OPTIMAL POWER FLOW WITH EXPECTED SECURITY COSTS

By

PARNJIT DAMRONGKULKAMJORN

Bachelor of Engineering
Kasetsart University
Bangkok, Thailand
May, 1991

Master of Science
Oklahoma State University
Stillwater, Oklahoma
May, 1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1999

OPTIMAL POWER FLOW WITH

EXPECTED SECURITY COSTS

Thesis Approved:

Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my deep sincere appreciation to my advisor Dr. Thomas W. Gedra for his guidance in all aspects of this research. I appreciate his time and commitment required to work with a doctoral student and would like to thank him for not letting me settle for anything less than my best. Because of him, I have become a stronger and better person. In addition, I would especially like to thank Dr. Gedra for supporting me with a research assistantship to pursue and complete my Ph.D.

I would also like to thank all of my committee members, Dr. Arthur Breipohl (University of Oklahoma, Department of Electrical Engineering and Computer Science), Dr. Rama Ramakumar, and Dr. Wade Brorsen for time and assistance. To Dr. Breipohl, I give a very special thanks for his diligent commitment in coming to OSU for my meetings regarding my progress in the Ph.D. program.

My special appreciation also goes to Dr. Hermann Burchard from the Department of Mathematics for his assistance regarding the computational method involving in this research. My deep gratitude is extended to my colleague John Condren for assisting me in the technical writing aspects of my thesis.

I would like to take this opportunity to thank my parents, Priyapas and Waewphan Damrongkulkamjorn, my two brothers, Panithi and Pakarn, for their enduring love, support, and encouragement. I gratefully dedicate this dissertation to my father, Priyapas, who has always believed in me and my work.

To my fiancé and best friend, Krit Jarinto, I truly appreciate all the love, support, and understanding you have given me during this time. You have been providing me with a great motivation to finish my study.

To my Thai friends at OSU, Kamolwan Lohsiwanont, Kullapapruk Piewthongngam, and Napaporn Laotaweesub, I would like to say "thank you" for your friendship, and

for making my stay at OSU a memorable experience. The time we spent together helped me go through the tough time of my study. I also owe a special thanks to another dear friend of mine, Theeraphol Sathapanawat, for his encouragement and support in every way.

Finally, I would like to thank the Department of Electrical and Computer Engineering for their support during my study at OSU.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$(\cdot)^0$	Pre-contingency state operating points
$(\cdot)^k$	Post-contingency state operating points
$\Delta^+_{\max} P_{Gi}$	Maximum amount of ramping up real power generation at bus i
$\Delta^-_{\max} P_{Gi}$	Maximum amount of ramping down real power generation at bus i
$\Delta^-_{\max} P_{Li}$	Maximum amount of interruptible real load of customer at bus i
λ	Lagrange multiplier for equality constraints
μ	Lagrange multiplier for inequality constraints
μ_d	Lagrange multiplier for coupling constraints
ν	Barrier parameter
ω^k	Penalty function for contingency k
π^0	Probability of system operating at normal state
π^k	Probability of contingency k
ρ	Penalty variables
AC	Actual (nonlinear) power flow equations
$B_{Ii}(P_{Li})$	Benefit curve at interruptible load bus i
$B_{Ui}(P_{Li})$	Benefit curve at uninterruptible load bus i
$C_{Gi}(P_{Gi})$	Cost of generation at generator bus i
$C_{Ii}(P^0_{Li} - P^k_{Li})$	Cost of load interruption

$C_{Ri}(\left\|P_{Gi}^k - P_{Gi}^0\right\|)$	Cost of ramping real power generator up or down
$d(Y^k, Y^0)$	Coupling constraints
DC	Linearized version of power flow equations
$\mathbf{e}$	Column vector $(1, 1, \ldots, 1)^T$
ESCOPF	Expected Security-Cost Optimal Power Flow
$f(Y)$	Inequality constraints
$\mathcal{G}$	Set of generator buses
$h(Y)$	Equality constraints
H	Hessian matrix
$\mathbf{I}$	Identity matrix
$\mathcal{I}$	Set of interruptible load buses
$\|I_{ij}\|$	Current flow in line ij
$I_{ij_{\max}}$	Maximum limit of line flows
J	Jacobian matrix
K	Total number of credible contingencies
$\mathcal{L}$	Lagrange function
$\mathbf{M}$	Diagonal matrix constructed from μ
MC_{Gi}	Marginal cost of generator i
MW	10^3 Watt
$NVCS$	Normalized violated constraint set
OPF	Optimal Power Flow
P_{ij}	Real power flow in line ij

P_{Gi}	Real power generator at bus i
$\left\|P_{Gi}^{k} - P_{Gi}^{0}\right\|$	Amount of real power generator being ramped up or down
$P_{Gi_{\max}}$	Maximum limits of real power generator
$P_{Gi_{\min}}$	Minimum limits of real power generator
P_{Li}	Real load at bus i
$P_{Li}^{0} - P_{Li}^{k}$	Amount of real load interruption
Q_{Gi}	Reactive power generator at bus i
$Q_{Gi_{\max}}$	Maximum limits of reactive power generator
$Q_{Gi_{\min}}$	Minimum limits of reactive power generator
Q_{Li}	Reactive load at bus i
s	Vector of slack variables
$\mathbf{S}$	Diagonal matrix constructed from s
S^{k}	Security cost of contingency k
SCOPF	Security-Constrained Optimal Power Flow
$\mathcal{U}$	Set of uninterruptible load buses
X	Vector of control variables:
	voltage magnitudes and angles, tap ratios, and phase shifters
$X_{\max}$	Maximum limits of control variables
$X_{\min}$	Minimum limits of control variables
Y	Vector of control and state variables: $(X, P_{Gi}, Q_{Gi}, P_{Li})^{T}$
z	Vector of all control and state variables, and multipliers

CHAPTER 1

INTRODUCTION

Security in a power system is its ability to continue providing power when there is a disturbance (or *contingency*) in the system. In the history of power system security, there have been some significant power failures or *blackouts* in the United States, such as in the WSCC on July 2-3, 1996 [1] (and again on August 10, 1996), and the Northeast blackout in 1965 [2] that caused significant economic losses. Studies of power system security must be done frequently in order to prevent such disasters from happening again.

A contingency would be the loss of one or more transmission lines if the lines were short-circuited or damaged, then were automatically disconnected from the rest of the system by the system protective relays. Another common contingency is the loss of on-line generators due to equipment failures or action by protective devices. The contingency causes changes in power flows in the remaining lines, and voltage magnitudes and angles of the remaining buses. Frequently, the changes in system state as a result of such a contingency are within the acceptable operational limits due to the redundancy in designing systems with several generators and transmission

lines. Under these circumstances, the system is able to continue serving loads and is considered *secure*. However, if the post-contingency limits and the pre-contingency operating points prevent the system from being redispatched to survive the contingency, cascading outages throughout the system may result. During the normal operating state, system security can be maintained by keeping enough spinning reserves on generators and the proper amount of transmission reserves on hand to ensure that the system will be able to be rescheduled to be within operational limits after any credible contingency which might arise.

From recent studies, a system that is operating at the least cost strategy and taking into account the additional constraints for system security is referred to as operating with *security-constrained optimal power flow* (SCOPF) [3, 4, 5, 6]. The SCOPF problem can be formulated by including the operating limits (or *security constraints*) on the post-contingency state into the *standard* optimal power flow (OPF) model. A more economically efficient SCOPF can be accomplished by additionally taking into account the capabilities of rescheduling system operating parameters after the contingencies [4]. The capabilities of rescheduling mainly refer to generator rescheduling. Starting at a given pre-contingency state, there is a chance that the post-contingency systems may not have feasible solutions even if the post-contingency rescheduling is allowed. In other words, the system may not necessarily survive every credible contingency that might occur in the system at the given normal operating state. If such cases arise, the pre-contingency state is considered insecure, and thus a new optimal pre-contingency state must be computed. In [4], infeasibility in the post-contingency system was mathematically avoided by adding some *penalty variables* to

the constraints of the post-contingency systems, along with associated *penalty costs.* Because of the difficulty of solving the problem, an iterative procedure is used. The total penalty costs are minimized subject to the post-contingency constraints, given the approximate solutions of the pre-contingency state. If the minimized penalty costs are not zero, the penalty variables are nonzero and that means the post-contingency state operating points stay outside the feasible region. In other words, the current estimate of the pre-contingency state violates the security constraints, and must be modified. This continues iteratively until the SCOPF is solved. The review of the power system security, the optimal power flow (OPF) problem, and the SCOPF problem with and without post-contingency rescheduling are summarized in Chapter 2.

In this research, we propose a new model for optimal power flow which takes into account system security. The proposed model is called *expected security-cost optimal power flow* (ESCOPF), since security is handled as an economic cost instead of as a constraint which has been done in the SCOPF. Each contingency has a security cost, which depends on the pre-contingency state, and it is multiplied by the contingency's probability to obtain the expected security-cost due to that contingency. The ESCOPF model takes into considering post-contingency rescheduling of generators, and also includes the ability to interrupt customer loads with relatively high interruption costs to ensure the feasible solutions of post-contingency states. The security cost for surviving a contingency k is the minimum cost of total power generation during the contingency including the cost of load interruption, and the cost of rescheduling the generators, and thus can be found by solving an OPF problem for contingency k, which we call the contingency subproblem.

The multipliers resulting from the ESCOPF problem can be used to study the *marginal value of spinning reserve* and the *marginal value of interruptible load* for the power system considering system security. One unit of the spinning reserve in power system refers to the ability of the system to ramp the generator up by one unit when it is needed. The marginal value of spinning reserve of a generator is interpreted as how much the objective function is improved when the binding constraint on the maximum limits of the generator is relaxed by one unit. The maximum limits of a generator include the ramping-up limit and the maximum capacity limit. Likewise, the marginal value of interruptible load can be interpreted as how much the objective function is improved when an additional unit of interruptible load is allowed. These marginal values are derived from the corresponding Lagrange multipliers by applying the *envelope theorem* as described in detail in Chapter 4.

The ESCOPF model is described in detail in Chapter 3 and its solution methodology is discussed in Chapter 4. The model has been successfully applied to three test cases: 3-bus system, 5-bus system, and IEEE 14-bus system. The program of the ESCOPF problem is written in MatLab and is included in Appendix A. Discussions of results are presented in Chapter 5. The input data for the three test cases are presented in the IEEE CDF format and included in Appendix B.

CHAPTER 2

LITERATURE REVIEW

This chapter introduces several optimization techniques in power systems. An early attempt at minimizing total generation cost in power systems was known as an *economic dispatch* problem. It was done by starting supplying load from the most efficient plant. As load increased until reaching the maximum capacity of that plant, the next most efficient plant came to service. The third efficient plant would not be in service until the second plant ran out of capacity. The economic dispatch focused on minimizing total fuel cost of the overall generation output. However, this method fails to take into account the transmission network of the power systems.

The capacity of the transmission lines and the network connection may prevent the optimal economic scheduling of generators. The most efficient generator may not be able to reach its maximum capacity without overloading the real power flow in the connected transmission lines. Therefore, the line flows must be computed once the generation scheduling is obtained from the economic dispatch to ensure the feasibility of operation. Another effect of the transmission network is the transmission losses. Transmission losses may cause an increase in the total generation and demand.

In order to study the effect of the transmission line flows and incremental losses, we must solve the *power-flow* problem along with the economic dispatch. The power-flow solution provides information on the real and reactive power flowing in each transmission line and the network losses. If the complete model of the transmission network is evaluated as an integral part of the optimal generation scheduling, the problem is known as *optimal power flow* or OPF. If security is taken into account in the OPF problem, the problem becomes *security-constrained optimal power flow* or SCOPF.

2.1 Economic Dispatch

The economic distribution of load among various generation units has been studied by determining the *incremental operating cost* of each unit as a function of power output. Let C_i be the operating cost of generator unit i in \$/hr, and P_i be the real power output of this generator. C_i includes fuel cost, which is the main part of operating cost for a fossil-fuel generator, as well as other variable costs such as maintenance cost. It is a piecewise continuous function of generator output P_i. Fixed costs, such as the generator installation (capital) cost, are not included in C_i since they do not depend on generator output P_i. Conventionally, C_i is sometimes expressed in terms of *heat rate* with units of BTU/hr because the heat rate is more precise in determining the characteristics of the generator unit. The heat rate can easily be converted to \$/hr by multiplying it with the cost of fuel in \$/BTU, which may vary over a period of time. The *incremental operating cost* of unit i is then obtained by taking the derivative of

C_i with respect to P_i. The criterion for optimal economic distribution of load among N generators is that all generators should generate output at the same incremental operating cost. That is

$$\frac{dC_1}{dP_1} = \frac{dC_2}{dP_2} = \ldots = \frac{dC_N}{dP_N}. \tag{2.1}$$

This criterion can be explained as follows. Suppose that the total load of a system is supplied by two generators. Let the distribution of load between the two generators be such that the incremental operating cost of one unit is higher than that of the other. Now suppose that some of the load from the higher incremental operating cost unit is transferred to the generator with a lower incremental operating cost. The reduction in operating cost caused by decreasing the load at the higher incremental operating cost unit is greater than the additional cost resulting from increasing the load on the other unit. As a result, the total operating cost of the system is reduced. The load transferred from the generator with higher incremental operating cost to the one with lower incremental operating cost can continue, and hence reduce the system operating cost until the incremental operating costs of the two generators are equal. The above discussion is the intuitive explanation of *economic dispatch* neglecting transmission losses and generator limits.

The ideal economic dispatch problem is stated by minimizing total operating cost subject to satisfying the total load demand. Mathematical formulation of lossless economic dispatch can then be expressed as

$$\min_{P_i} \quad \sum_{i=1}^{N} C_i \tag{2.2}$$

subject to:

$$P_1 + P_2 + \ldots + P_N = P_{load}. \tag{2.3}$$

If the generator operating limits are included in the economic dispatch, the problem can be expressed as

$$\min_{P_i} \quad \sum_{i=1}^{N} C_i \tag{2.4}$$

subject to:

$$P_1 + P_2 + \ldots + P_N = P_{load} \tag{2.5}$$

$$P_{i_{\min}} \leq P_i \leq P_{i_{\max}} \qquad : i = 1, \ldots, N. \tag{2.6}$$

If the limits on some generators are binding, the generators may not be able to generate output at the same incremental operating cost. Taking the example of the system with two generators mentioned earlier, suppose the transfer of load from the higher incremental operating cost generator is interrupted because the generator with the lower incremental operating cost has reached its maximum capacity. In this case, the incremental operating cost of the generator that hits its maximum limit will be lower than that of the other generator.

On the other hand, suppose the generator that has higher incremental operating cost reaches its minimum limit. No more load can be transferred to the other generator because the first generator must generate the required minimum capacity in order to stay on-line. That is, the transfer of load is interrupted when the incremental operating cost of the generator that hits its minimum limit is still higher than that of the other generator. The optimal criterion for lossless economic dispatch with generator limits can be summarized as [7]

$$\frac{dC_i}{dP_i} = \lambda \qquad \text{for} \quad P_{i_{\min}} < P_i < P_{i_{\max}}$$

$$\frac{dC_i}{dP_i} \leq \lambda \qquad \text{for} \quad P_i = P_{i_{\max}} \tag{2.7}$$

$$\frac{dC_i}{dP_i} \geq \lambda \qquad \text{for} \quad P_i = P_{i_{\min}}$$

where λ is a constant value representing the incremental operating cost of each generator that is operating within its limits.

Economic Dispatch with Transmission Losses

The classical economic dispatch problem taking into account system transmission losses is expressed as follows.

$$\min_{P_i} \quad \sum_{i=1}^{N} C_i \tag{2.8}$$

subject to:

$$P_1 + P_2 + \ldots + P_N = P_{load} + P_{loss} \tag{2.9}$$

$$P_{i_{\min}} \leq P_i \leq P_{i_{\max}} \qquad i = 1, \ldots, N. \tag{2.10}$$

The cost objective function in (2.8) is minimized subject to generator output P_i, that is

$$\sum_{i=1}^{N} \frac{\partial C_i}{\partial P_i}\, dP_i = 0. \tag{2.11}$$

Since the total load P_{load} is constant, $dP_{load} = 0$, and we can write (2.9) as:

$$\sum_{i=1}^{N} dP_i - dP_{loss} = 0. \tag{2.12}$$

Transmission loss P_{loss} is a function of generator outputs P_i. Therefore, dP_{loss} can be expressed as:

$$dP_{loss} = \sum_{i=1}^{N} \frac{\partial P_{loss}}{\partial P_i}\, dP_i. \tag{2.13}$$

By substituting dP_{loss} in (2.12), multiplying by λ, and subtracting the result form (2.11), we obtain

$$\sum_{i=1}^{N} \left(\frac{\partial C_i}{\partial P_i} + \lambda \frac{\partial P_{loss}}{\partial P_i} - \lambda \right) dP_i = 0, \tag{2.14}$$

or simply

$$\frac{\partial C_i}{\partial P_i} + \lambda \frac{\partial P_{loss}}{\partial P_i} - \lambda = 0; \qquad i = 1, \ldots, N. \tag{2.15}$$

The multiplier λ can be computed from (2.15) as:

$$\lambda = \frac{\partial C_i}{\partial P_i} \cdot \frac{1}{1 - \partial P_{loss}/\partial P_i}. \tag{2.16}$$

The second term on the right-hand side in (2.16), denoted by L_i, is called the *penalty factor* for generator i, and can be expressed as:

$$L_i = \frac{1}{1 - \partial P_{loss}/\partial P_i}. \tag{2.17}$$

Note that the multiplier λ is derived based on the assumption

$$P_{i_{\min}} < P_i < P_{i_{\max}}; \qquad \forall i.$$

Recall that with the same assumption, the lossless economic dispatch problem yields

$$\lambda = \frac{\partial C_i}{\partial P_i}, \tag{2.18}$$

which is the incremental operating cost of generator i.

Therefore the multiplier λ for economic dispatch with losses included is the incremental operating cost of generator i multiplied by its penalty factor.

Note that economic dispatch simply reduces the entire load flow equations into one power balance equation saying that the total real power generation must

be equal to the total real load (plus losses if considered), and ignores the reactive power flows in the system. Furthermore, it does not take into account the effect of the *transmission network* of power systems. Historically, analysis of the transmission network is accomplished by solving the power-flow problem (or *load flow*).

2.2 The Power-Flow Problem

The power-flow problem (or *load-flow* problem) is the network solution that provides the magnitude and phase angle of the voltage at each bus and the real and reactive power flowing in each line of the power system network, given that the system is in the normal operating state. Another important quantity that comes from the power-flow solution is the network losses. Therefore a power-flow study provides the relationship of losses in a power system network to economic dispatch. In power-flow problems, all transmission lines are represented by π-equivalent circuits, and the transformers are represented by ideal voltage transformers in series with impedances. The π-equivalent circuit of a transmission line is shown in Figure 2.1, where z_{ij} is the series impedance of the line between buses i and j, and y_c is the total line-charging admittance. The total current injected at bus i, denoted by I_i is equal to the sum of the current flowing in line ij and the line-charging current, and can be expressed in terms of bus voltages V_i and V_j as follows:

$$\begin{aligned} I_i &= I_{ij} + I_c \\ &= \frac{(V_i - V_j)}{z_{ij}} + V_i \cdot \frac{y_c}{2} \\ &= (V_i - V_j) \cdot y_{ij} + V_i \cdot \frac{y_c}{2}, \end{aligned}$$

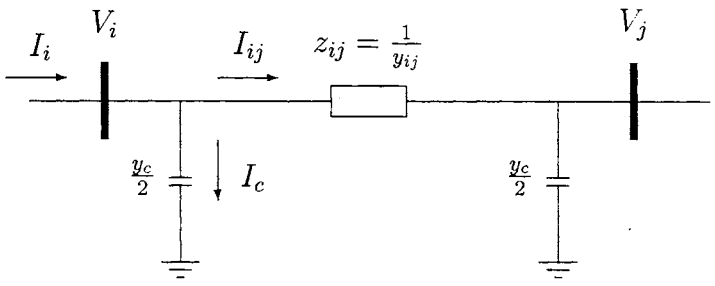

Figure 2.1: π-equivalent circuit for a transmission line.

$$I_i = (y_{ij} + \frac{y_c}{2}) \cdot V_i - y_{ij} \cdot V_j. \tag{2.19}$$

Similarly for bus j,

$$I_j = (y_{ij} + \frac{y_c}{2}) \cdot V_j - y_{ij} \cdot V_i. \tag{2.20}$$

The above relationships can be written in a matrix form as:

$$\begin{bmatrix} I_i \\ I_j \end{bmatrix} = \begin{bmatrix} (y_{ij} + \frac{y_c}{2}) & -y_{ij} \\ -y_{ij} & (y_{ij} + \frac{y_c}{2}) \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix}, \tag{2.21}$$

or simply

$$\begin{bmatrix} I_i \\ I_j \end{bmatrix} = \begin{bmatrix} Y_{ii} & Y_{ij} \\ Y_{ji} & Y_{jj} \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix}, \tag{2.22}$$

where

Y_{ii} : the sum of all admittance connected to bus i,

Y_{jj} : the sum of all admittance connected to bus j,

Y_{ij},Y_{ji} : negative of the admittance between buses i and j.

In general, a power system consists of more than two buses. Therefore, the relationship between bus currents and voltages in an N bus system can be expressed

as:

$$\begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_N \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1N} \\ Y_{21} & Y_{22} & \cdots & Y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N1} & Y_{N2} & \cdots & Y_{NN} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{bmatrix}, \tag{2.23}$$

which is simply

$$\mathbf{I} = \mathbf{Y}_{\text{bus}}\mathbf{V}. \tag{2.24}$$

The matrix $\mathbf{Y}_{\text{bus}}$ is the *bus admittance matrix*. The simple rules to form a $\mathbf{Y}_{\text{bus}}$ matrix are [8]:

- The diagonal element Y_{ii} is the sum of all the admittance *directly* connected to bus i.
- The off-diagonal element Y_{ij} is the *negative* of the admittance connected between buses i and j.

An element Y_{ij} and a bus voltage V_i can be expressed in polar coordinates as:

$$Y_{ij} = |Y_{ij}|\angle\delta_{ij} = |Y_{ij}|(\cos\delta_{ij} + j\sin\delta_{ij}), \tag{2.25}$$

$$V_i = |V_i|\angle\theta_i = |V_i|(\cos\theta_i + j\sin\theta_i). \tag{2.26}$$

From (2.23), the total current injected at any bus i is the summation

$$I_i = Y_{i1}V_1 + Y_{i2}V_2 + \cdots + Y_{iN}V_N = \sum_{j=1}^{N} Y_{ij}V_j. \tag{2.27}$$

The *complex* power consisted of *real* and *reactive* power (P_i and Q_i, respectively) is computed as:

$$S_i = P_i + jQ_i = V_i I_i^*, \tag{2.28}$$

where I_i^* is the complex conjugate of the current at bus i. Thus the complex conjugate of the total power injected at bus i is

$$\begin{aligned} P_i - jQ_i &= V_i^* I_i \\ &= V_i^* \sum_{j=1}^{N} Y_{ij} V_j. \end{aligned} \tag{2.29}$$

Substituting Y_{ij} and V_i with (2.25) and (2.26), we obtain

$$P_i - jQ_i = \sum_{j=1}^{N} |V_i Y_{ij} V_j| \angle(\delta_{ij} + \theta_j - \theta_i). \tag{2.30}$$

Expanding this equation and equating real and imaginary parts, we obtain

$$P_i = \sum_{j=1}^{N} |V_i Y_{ij} V_j| \cos(\delta_{ij} + \theta_j - \theta_i), \tag{2.31}$$

$$Q_i = -\sum_{j=1}^{N} |V_i Y_{ij} V_j| \sin(\delta_{ij} + \theta_j - \theta_i). \tag{2.32}$$

Equations (2.31) and (2.32) are called the *power flow equations.* They are *calculated* values of the net real and reactive power entering bus i.

Let P_{Gi} and P_{Li} denote the real power generated and the real load demanded at bus i, respectively. Then $P_{i,sch} = P_{Gi} - P_{Li}$ is the *scheduled* real power at bus i. Similarly, the scheduled reactive power at bus i is $Q_{i,sch} = Q_{Gi} - Q_{Li}$, where Q_{Gi} and Q_{Li} are reactive power generated and reactive load at bus i, respectively.

The *mismatch* real power at any bus i, ΔP_i, is defined as the difference between the scheduled real power $P_{i,sch}$ and the calculated value P_i, i.e.

$$\Delta P_i = P_{i,sch} - P_i = (P_{Gi} - P_{Li}) - P_i. \tag{2.33}$$

Likewise, the mismatch reactive power is given as:

$$\Delta Q_i = Q_{i,sch} - Q_i = (Q_{Gi} - Q_{Li}) - Q_i. \tag{2.34}$$

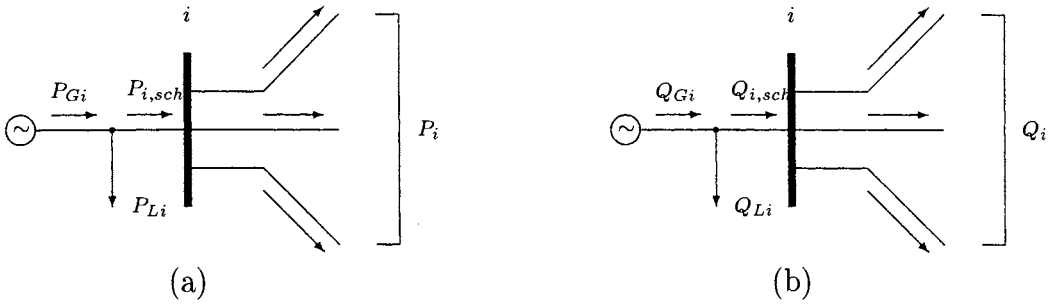

Figure 2.2: Notations for (a) real and (b) reactive power at bus i.

If the calculated powers, P_i and Q_i, are equal to their scheduled values, then the mismatches are zero, and we obtain the *power balance equations* at bus i as follows:

$$\Delta P_i = 0 \implies P_i = P_{Gi} - P_{Li}, \tag{2.35}$$

$$\Delta Q_i = 0 \implies Q_i = Q_{Gi} - Q_{Li}. \tag{2.36}$$

If there is no generator or load at bus i, the corresponding terms (P_{Gi} and Q_{Gi}, or P_{Li} and Q_{Li}) are equal to zero. The notations of power mismatches are illustrated in Figure 2.2. If the scheduled real and reactive powers are known at any bus i, then the voltage magnitude $|V_i|$ and angle θ_i can be calculated from the power flow equations (2.31) and (2.32) such that P_i and Q_i satisfy the power balance equations (2.35) and (2.36).

At each bus i, two out of four quantities P_i, Q_i, V_i, and θ_i are specified and the other two are to be calculated. The specified quantities are chosen based on the following three bus types:

1. *Slack bus.* There is only one slack bus in a power system network. The slack bus can be any bus in the system that has a generator connected to it. It is widely common to number the slack bus 1. The angle of the slack bus is set to be a reference for other voltage angles in the network. Normally, the angle of the slack bus is chosen to be zero for convenience, that is $\theta_1 = 0$. The voltage magnitude of the slack bus is normally set as $|V_1| = 1$. The two specified quantities $|V_1|$ and θ_1 are used to compute P_1 and Q_1 from power flow equations (2.31) and (2.32). The power balance equations are not defined at the slack bus.

2. *Voltage-controlled bus.* At any bus to which a generator is connected, the voltage magnitude can be maintained at a constant value by adjusting the generator excitation, and thus the bus is called a voltage-controlled bus. Moreover, at any generator bus, real power generation P_{Gi} is known. If there is also a load connected at this bus, then P_{Li} is known, otherwise P_{Li} is zero. Either way, P_i is known from the power balance equation (2.35). Therefore, a generator bus is sometimes called a *P-V bus* because P_i and $|V_i|$ are specified. The other two quantities Q_i and θ_i at the *P-V* bus are left to be computed.

3. *Load bus.* At a nongenerator bus, both P_{Gi} and Q_{Gi} are zero. The real and reactive power drawn from the bus by the load (P_{Li}, Q_{Li}) are known from measurement. From these specifications, P_i and Q_i are known from power balance equations (2.35) and (2.36). Then $|V_i|$ and θ_i are to be computed. The load bus is also known as a *P-Q bus.*

Bus type	Specified quantities	Unknowns
Slack bus; i=1	θ_1,$\|V_1\|$	P_1,Q_1
Voltage-controlled bus	P_i,$\|V_i\|$	θ_i,Q_i
Load bus	P_i,Q_i	θ_i,$\|V_i\|$

Table 2.1: Summary of three bus types in power-flow problem.

The above discussion is summarized in Table 2.1. The functions of P_i and Q_i in (2.31) and (2.32) are nonlinear functions of state variables θ_i and $|V_i|$. Therefore, solving power flow equations requires iterative methods such as the Gauss-Seidel and Newton-Raphson procedures. We will examine the Newton-Rapshon method in some detail.

Suppose there are g generator buses in an N-bus power system. Assuming that bus 1 is the slack bus, buses 2 through g are voltage-controlled buses, and buses $g+1$ through N are load buses. The power-flow problem for this system can be summarized as follows:

	Slack bus	P-V bus	P-Q bus
Knowns	θ_1	$P_2,\ldots,P_g$	$P_{g+1},\ldots,P_N$
	$\|V_1\|$	$\|V_2\|,\ldots,\|V_g\|$	$Q_{g+1},\ldots,Q_N$
Unknowns	P_1	$\theta_2,\ldots,\theta_g$	$\theta_{g+1},\ldots,\theta_N$
	Q_1	$Q_2,\ldots,Q_g$	$\|V_{g+1}\|,\ldots,\|V_N\|$

The unknown variables P_1, and Q_1 through Q_g can be computed from (2.31) and (2.32), once θ_2 through θ_g are known. The number of unknown state variables are $(2N-g-1)$, from which $(N-1)$ are unknown angles, and $(N-g)$ are unknown voltage magnitudes. We start solving this power-flow problem by guessing the unknown

angles $\tilde{\theta}_2, \ldots, \tilde{\theta}_N$, and the unknown voltage magnitudes $|\tilde{V}_{g+1}|, \ldots, |\tilde{V}_N|$. We then use these guesses to compute estimated power flow equations $\tilde{P}_i$ and $\tilde{Q}_i$ and compare the calculated values with the known P_i and Q_i. There are $(N-1)$ known P_i, and $(N-g)$ known Q_i as shown above. Therefore, we can write $(2N-g-1)$ mismatch equations as:

$$\begin{aligned}
\Delta P_2 &= P_2 - \tilde{P}_2(\theta_1, \tilde{\theta}_2, \ldots, \tilde{\theta}_N, |V_1|, |V_2|, \ldots, |V_g|, |\tilde{V}_{g+1}|, \ldots, |\tilde{V}_N|) \\
&\vdots \quad \vdots \\
\Delta P_N &= P_N - \tilde{P}_N(\theta_1, \tilde{\theta}_2, \ldots, \tilde{\theta}_N, |V_1|, |V_2|, \ldots, |V_g|, |\tilde{V}_{g+1}|, \ldots, |\tilde{V}_N|) \\
\Delta Q_{g+1} &= Q_{g+1} - \tilde{Q}_{g+1}(\theta_1, \tilde{\theta}_2, \ldots, \tilde{\theta}_N, |V_1|, |V_2|, \ldots, |V_g|, |\tilde{V}_{g+1}|, \ldots, |\tilde{V}_N|) \\
&\vdots \quad \vdots \\
\Delta Q_N &= Q_N - \tilde{Q}_N(\theta_1, \tilde{\theta}_2, \ldots, \tilde{\theta}_N, |V_1|, |V_2|, \ldots, |V_g|, |\tilde{V}_{g+1}|, \ldots, |\tilde{V}_N|)
\end{aligned}$$

We can now solve for $(2N-g-1)$ voltage magnitudes and angles by setting the mismatches to zero. Since the mismatch equations are nonlinear functions, solving for state variables in this nonlinear problem requires an iterative method described as follows.

Suppose there are n nonlinear functions with n state variables $\mathbf{x}$, and one controlled variable u, expressed as:

$$\mathbf{g}(\mathbf{x}, u) = \mathbf{f}(\mathbf{x}, u) - \mathbf{c} \quad \text{where} \quad \mathbf{f}: \mathbf{R}^n \rightarrow \mathbf{R}^n \tag{2.37}$$

For a given value of u, let us define the actual solutions $\mathbf{x}^*$ in terms of the estimated values $\mathbf{x}^{(0)}$ as $\mathbf{x}^* = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}$, where $\Delta\mathbf{x}^{(0)}$ is the *correction step*. Therefore,

the above function can be written as:

$$\mathbf{g}(\mathbf{x}^*, u) = \mathbf{g}(\mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}, u) \; = \; 0 \tag{2.38}$$

We need to solve for the correction steps $\Delta\mathbf{x}^{(0)}$ by expanding (2.38) in Taylor's series around the estimated values

$$\mathbf{g}(\mathbf{x}^*, u) = \mathbf{g}(\mathbf{x}^{(0)}, u) + \nabla\mathbf{g} \mid_{\mathbf{x}^{(0)}} \Delta\mathbf{x}^{(0)} + \ldots \; = \; 0. \tag{2.39}$$

By ignoring the partial derivatives of order greater than 1 in Taylor's series, (2.39) can be rewritten as:

$$\nabla\mathbf{g} \mid_{\mathbf{x}^{(0)}} \Delta\mathbf{x}^{(0)} = -\mathbf{g}(\mathbf{x}^{(0)}, u). \tag{2.40}$$

The correction steps $\Delta\mathbf{x}^{(0)}$ can be computed and added to the current estimates to obtain the new and better estimated solutions

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}. \tag{2.41}$$

The process is repeated until the correction steps become so small in magnitude, that is $|\Delta\mathbf{x}^*|$ values are less than a chosen $\varepsilon > 0$. Then we say that the updated $\mathbf{x}^*$ are the solutions of the nonlinear problems expressed in (2.37).

2.3 Optimal Power Flow (OPF)

Once the optimal generation scheduling is obtained from the economic dispatch, a power-flow study must be done to ensure the feasibility of the transmission lines and to obtain the incremental losses of the network. The information on the incremental losses as stated in (2.17) is used to recompute the economic dispatch problem.

If a transmission line is overloaded, the economic dispatch must shift the optimal generation scheduling to relax the flow in the overloaded line. Then the power-flow problem is solved again using the generation scheduling from the updated economic dispatch. A new set of overloaded lines may be detected, and the economic dispatch must optimally release those flows. The computation goes back and forth between the economic dispatch and the power-flow problem until the optimal dispatch without violating line flow constraints is obtained. To avoid the cycling between the economic dispatch and the power-flow solution, the complete model of the transmission network is embedded into the optimal generation scheduling by means of the *optimal power flow* (OPF) problem. The OPF problem can be viewed as solving power-flow solution by adjusting control variables to satisfy the transmission network constraints while minimizing a specified objective function, which may be the total cost of generation or the total losses of the system.

The OPF problem was first stated in the 1960s as a constrained optimization problem [9], or a problem of minimizing an objective function over some variables subject to sets of constraints. Mathematically, the OPF problem can be written in a general form as:

$$\begin{aligned} &\min_{Y} \quad C(Y) \\ &\text{subject to:} \\ &\qquad h_i(Y) = 0; \qquad i = 1, \ldots, n, \\ &\qquad f_{Li} \leq f_i(Y) \leq f_{Hi}; \quad i = 1, \ldots, m, \end{aligned} \tag{2.42}$$

where

Y : a vector of control and state variables

$C(Y)$: an objective function

$h(Y)$: a set of equality constraint functions, which are power flow equations

$f(Y)$: a set of inequality constraint functions, such as voltage limits, generator capacity limits, and line flow limits

Initially, control variables include only voltage magnitudes and voltage phase angles [10]. Transformer tap ratios, and phase shifter angles are added to the set of control variables in later work [11, 12, 13]. Real and reactive power generations, and transmission line flows may be treated as functions of control variables or as independent state variables.

Constraints involved in the OPF problem consist of sets of equality and inequality constraints. Equality constraints are power flow equations. These equations say that the total real/reactive power injected at any bus is equal to the total real/reactive power generated at that bus plus the total real/reactive load consumed at that bus. Inequality constraints are system operating limits, such as voltage limits, tap ratio and phase shifter limits, and generator capacity limits. Some authors [11] include line flow limits in the set of inequality constraints for standard OPF, while others treat OPF with line flow limits as *security-constrained* optimal power flow.

Usually the objective function of OPF problem is the total cost of power system generation or total losses in power system. However, other types of objective functions for the OPF problem are possible depending on the intended application. Multi-objective OPF problems have also been studied. For example: the amount of

pollutants generated from power plants are minimized along with the total cost of power generation in [14]; and the objective function in [15] combines both fuel costs and system losses. OPF may be used as part of power system planning by adjusting the objective function to minimize costs of additional VAR sources corresponding to system losses [16, 17].

The historical review of OPF is summarized in [9], where Huneault and others arrange OPF literatures in 1960s, 1970s and 1980s into four groups based on their problem formulations, and five groups based on their solution methodologies: equal incremental cost (EICC) method, linear programming, quadratic programming, P-Q decomposition methods, and penalty function methods. The most widely used methods for solving OPF are gradient methods based on linear programming.

Since the work in this research is based on solving the proposed optimization problem by *Newton's* method and *Primal-Dual Interior-Point* (PDIP) method using *logarithmic barrier function*, the discussion of Newton's method and PDIP will be done in some details in Sections 2.3.1 and 2.3.2 as part of the research background.

Applications of OPF

OPF has been used as a tool to improve power system planning, and operation by adjusting the objective function and/or the constraints. From the power system planning point of view, OPF can be used to determine the optimal types, sizes, settings, capital costs, and locations of FACTS (Flexible AC Transmission System) devices, regarding system real losses and voltage/VAR control [17, 18, 19, 20, 21,

22, 23]. For the planning purpose stated above, the objective function of OPF is minimizing real power losses in the system or total costs of additional VAR sources or FACTS devices.

From the power system operation point of view, OPF is used to optimally dispatch real and reactive power considering system security. It can also be used to determine short-run electric pricing (i.e. spot pricing) [3, 24] and transmission pricing in power systems by means of Lagrange multipliers.

2.3.1 OPF by Newton's Approach

This section discusses Newton's method for OPF following the discussions in [11, 12, 16]. The constrained optimization problem stated in (2.42) can be solved by forming the *Lagrange* function,

$$\begin{aligned}\mathcal{L}(Y,\lambda,\mu_H,\mu_L) &= C(Y)+\sum_{i=1}^{n}\lambda_i h_i(Y)\\ &+ \sum_{i\in\mathcal{A}_H}\mu_{Hi}(f_i(Y)-f_{Hi})+\sum_{i\in\mathcal{A}_L}\mu_{Li}(f_{Li}-f_i(Y)),\end{aligned} \tag{2.43}$$

where λ_i is the *Lagrange multiplier* for the i^{th} equality constraint. Assuming that we know which inequality constraints are binding at the upper limits and which ones are binding at the lower limits, and put them in the sets $\mathcal{A}_H$ and $\mathcal{A}_L$, respectively, then the inequality constraints in (2.42) are now enforced as equality constraints, and can be written as:

$$f_i(Y)-f_{Hi} = 0; \quad i\in\mathcal{A}_H \tag{2.44}$$

$$f_{Li}-f_i(Y) = 0; \quad i\in\mathcal{A}_L. \tag{2.45}$$

Thus μ_{Hi} and μ_{Li} have the same interpretation as λ_i: they are the Lagrange multipliers for *binding* inequality constraints. However, we need $\mu_{Hi} \geq 0$ and $\mu_{Li} \geq 0$ for every i [16]. We can drop the inequality constraints that are not binding since their μ's are known to be zero by *complementary slackness* condition [16]. That is

$$\begin{aligned} f_i(Y) = f_{Hi} &\implies \mu_i \geq 0, \\ f_i(Y) = f_{Li} &\implies \mu_i \geq 0, \\ f_{Li} \leq f_i(Y) \leq f_{Hi} &\implies \mu_i = 0. \end{aligned} \tag{2.46}$$

Therefore, only binding inequality constraints are included in the Lagrange function shown in (2.43) with corresponding nonzero μ's. The set $\mathcal{A} = \mathcal{A}_H \cup \mathcal{A}_L$ is therefore the set of binding constraints, and known as the *active set.*

Solution of a constrained optimization problem can be solved by adjusting control and state variables, and Lagrange multipliers to satisfy the following *first-order necessary optimality conditions*:

$$g(z) = \nabla_z \mathcal{L}(z) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial Y} \\ \frac{\partial \mathcal{L}}{\partial \lambda} \\ \frac{\partial \mathcal{L}}{\partial \mu_H} \\ \frac{\partial \mathcal{L}}{\partial \mu_L} \end{bmatrix} = 0, \tag{2.47}$$

where z is a vector of $[Y, \lambda, \mu_H, \mu_L]^T$.

Equation (2.47) is also called the *Karush-Kuhn-Tucker* (KKT) conditions. The second, third, and fourth terms in the vector $g(z)$ can be interpreted as follows. All equality constraints must be satisfied, functions that are binding at the upper limits

are equal to their upper limits, and functions that are binding at the lower limits are equal to their lower limits, i.e.

$$\begin{aligned} h_i(Y) &= 0; \qquad i = 1, \ldots, n, \\ f_i(Y) - f_{Hi} &= 0; \qquad i \in \mathcal{A}_H, \\ f_{Li} - f_i(Y) &= 0; \qquad i \in \mathcal{A}_L. \end{aligned}$$

From the above conditions, it is not possible for each inequality constraint function to be binding at its upper and lower limits simultaneously, that is $\mathcal{A}_H \cap \mathcal{A}_L = \emptyset$.

To solve the KKT conditions, $g(z) = 0$, the Newton method is applied by applying the Taylor's series expansion around a current point z^p as:

$$g(z) = g(z^p) + \left.\frac{d\,g(z)}{dz}\right|_{z=z^p} \cdot (z - z^p) + \underbrace{\frac{1}{2!}\left.\frac{d^2 g(z)}{dz^2}\right|_{z=z^p} \cdot (z - z^p)^2 + \cdots}_{\text{H.O.T.}} \tag{2.48}$$

The current point z^p can either be an initial guess in the first iteration of the computation, or the estimate solution from the prior iteration.

Recall that we want $g(z) = 0$. By ignoring the high order terms (H.O.T.) and defining $\Delta z = z - z^p$, the above equation can now be rewritten as:

$$\left.\frac{d\,g(z)}{dz}\right|_{z=z^p} \cdot \Delta z = -g(z^p). \tag{2.49}$$

The quantity Δz is the update vector, or the *Newton step* [16], and it tells how far and in which direction the variables and multipliers should move from this current point z^p to get closer to the solution. Since $g(z)$ is the gradient of the Lagrange function $\mathcal{L}(z)$, equation (2.49) can be written in terms of the Lagrange function $\mathcal{L}(z)$ as:

$$\left.\frac{\partial^2 \mathcal{L}(z)}{\partial z^2}\right|_{z=z^p} \cdot \Delta z = -\left.\frac{\partial \mathcal{L}(z)}{\partial z}\right|_{z=z^p}. \tag{2.50}$$

Or simply

$$W \cdot \Delta z = -g, \tag{2.51}$$

where W denotes the second order derivatives (or the *Hessian matrix*) of $\mathcal{L}(z)$ with respect to z, and g is the gradient of the Lagrange function evaluated at the current point. Equation (2.51) can be written in matrix form as:

$$\begin{bmatrix} H & J^{\mathrm{T}} & A_H^{\mathrm{T}} & A_L^{\mathrm{T}} \\ J & 0 & \cdots & 0 \\ A_H & \vdots & \ddots & \vdots \\ A_L & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \Delta Y \\ \Delta\lambda \\ \Delta\mu_H \\ \Delta\mu_L \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial Y} \\ \frac{\partial \mathcal{L}}{\partial \lambda} \\ \frac{\partial \mathcal{L}}{\partial \mu_H} \\ \frac{\partial \mathcal{L}}{\partial \mu_L} \end{bmatrix}, \tag{2.52}$$

where the *Hessian* matrix: H, and *Jacobian* matrices: J, A_H and A_L are given as follows:

$$\begin{aligned} H &= \tfrac{\partial^2 \mathcal{L}(z)}{\partial Y^2}, \\ J &= \tfrac{\partial^2 \mathcal{L}(z)}{\partial \lambda\, \partial Y} = \tfrac{\partial h_i(Y)}{\partial Y}; \qquad i = 1, \ldots, n, \\ A_H &= \tfrac{\partial^2 \mathcal{L}(z)}{\partial \mu_H\, \partial Y} = \tfrac{\partial f_i(Y)}{\partial Y}; \qquad i \in \mathcal{A}_H, \\ A_L &= \tfrac{\partial^2 \mathcal{L}(z)}{\partial \mu_L\, \partial Y} = \tfrac{\partial f_i(Y)}{\partial Y}; \qquad i \in \mathcal{A}_L. \end{aligned}$$

The Newton step can be obtained from solving

$$\Delta z = W^{-1} \cdot (-g), \tag{2.53}$$

where W^{-1} is computed by doing LU factorization and forward-backward substitution. Then a vector of estimate solution for the next iteration is updated as:

$$z^{p+1} = z^p + \Delta z. \tag{2.54}$$

After obtaining an updated set of variables and multipliers, a new set of binding inequality constraints (or the active set) can be determined as follows:

- If the updated μ's of the constraint functions in the current active set are zero or become negative, then the corresponding constraints must be released from the current active set because after the update, they no longer violate the limits.

- If other constraint functions evaluated at the updated variables violate their limits, then those constraints must be included in the new active set.

Once the active set has been updated, $g(z^{p+1})$ is checked for convergence. There are several criteria for checking convergence of the Newton OPF. The convergent tolerance may be set on the maximum absolute value of elements in $g(z)$ [25], or on its norm [16]. Another convergence criterion may be specified in terms of Δz [16]. If the updated z^{p+1} does not meet the desired convergence criterion, the Newton step calculation is repeated. The flow chart of the Newton OPF is illustrated in Figure 2.3.

Improvements of Newton's Method for OPF

Even though Newton method is considered a relatively fast method for solving OPF problem since the required computational work does not depend on the number of system control variables [25], some improvements for the method are still needed [26]. One of the most challenging of Newton's approach on OPF is the problem of identifying the correct set of binding inequality constraints [25]. In practice, we have found that "cycling" behavior often occurs with the active set methods, in which a small number of constraints repeatedly enter and leave the active set, without resulting in convergence. Several ways to improve the convergence of Newton's method by means of correctly identify the set of binding constraints have been studied and tested on

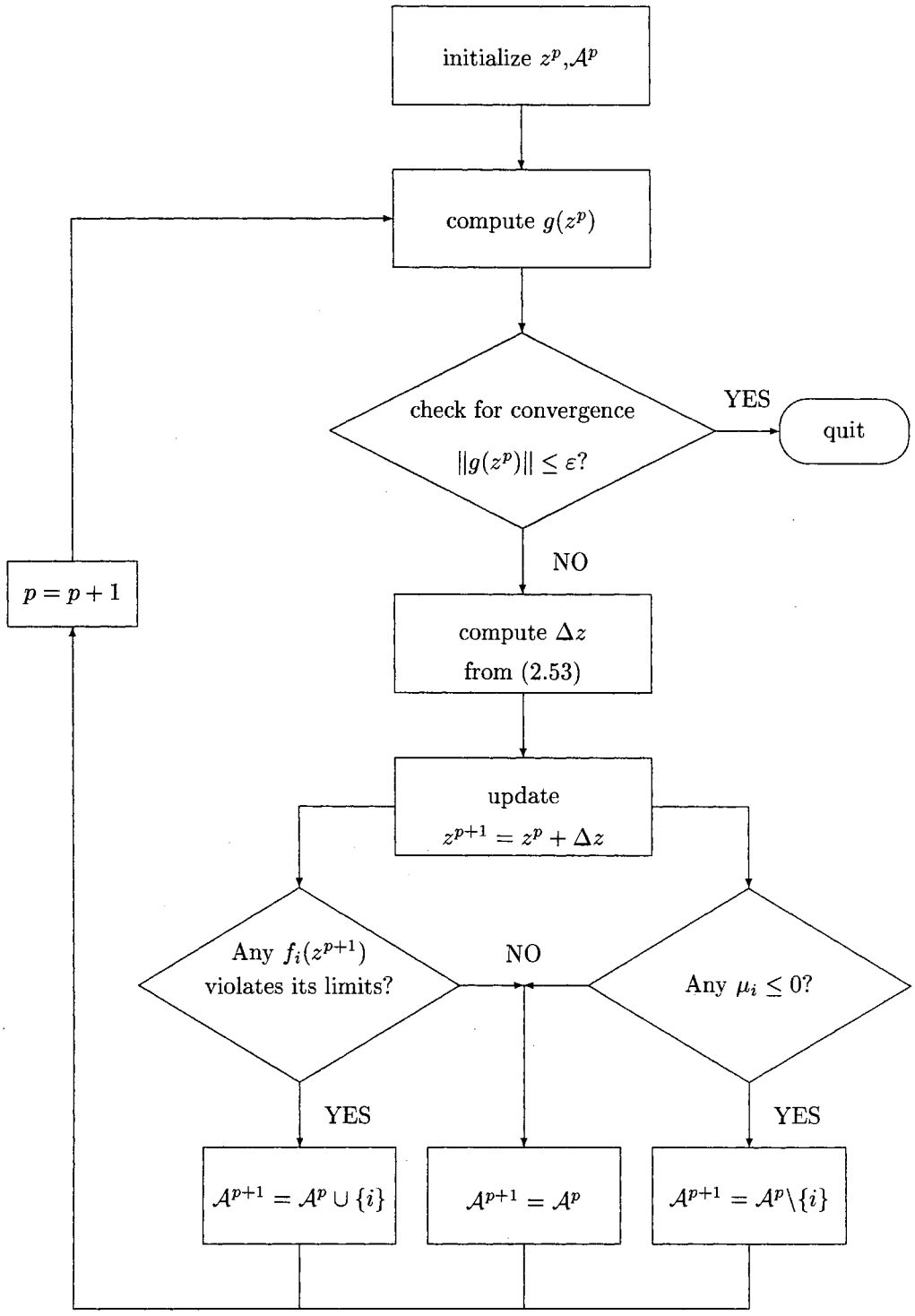

Figure 2.3: Flow chart of OPF by Newton's method.

some real power systems. These include the works in [13, 25, 27, 28]. The Linear programming has been used to identify the set of binding inequality constraints in [28]. However, this technique is claimed to be too slow when being applied to a large-scale system [25].

In [13], the Newton's method is used to solve for the optimality conditions in OPF, where the equality constraints are still represented by Lagrange multipliers, but the inequality constraints are handled by using the *penalty function* method. By this method, binding inequality constraints are added to the objective function as penalty terms. The OPF problem with the penalty method modified from problem (2.42) can be written as:

$$\begin{aligned} \min_{Y} \quad & C(Y) + \sum_{i \in \mathcal{A}_H \cup \mathcal{A}_L} \psi_i \\ \text{subject to:} & \\ & h_i(Y) = 0; \qquad i = 1, \ldots, n, \end{aligned} \tag{2.55}$$

where ψ_i is the penalty term corresponding to the i^{th} binding inequality constraints.

Santos and Costa [29] combine Newton's method and the augmented Lagrangian method in order to avoid the difficulty of identifying the set of binding constraints. The augmented Lagrangian method introduces *slack variables* in the inequality constraints, and adds penalty functions for inequality constraints in the objective function. The modified OPF model based on this method is expressed as:

$$\begin{aligned} \min_{Y} \quad & C(Y) + \frac{\upsilon}{2}\sum_{i=1}^{m}\{f_i(Y) - f_{Hi} + s_{Hi}\}^2 + \frac{\upsilon}{2}\sum_{i=1}^{m}\{f_{Li} - f_i(Y) + s_{Li}\}^2 \\ & \text{subject to:} \end{aligned} \tag{2.56}$$

$$
\begin{aligned}
h_i(Y) &= 0; \qquad i = 1, \ldots, n, \\
f_i(Y) - f_{Hi} + s_{Hi} &= 0; \qquad i = 1, \ldots, m, \\
f_{Li} - f_i(Y) + s_{Li} &= 0; \qquad i = 1, \ldots, m, \\
s_{Hi}, s_{Li} &\geq 0, \qquad \forall i
\end{aligned}
$$

where

s_i : slack variable corresponding to inequality constraint $f_i(Y)$,

υ : penalty factor.

The corresponding Lagrange function is

$$
\begin{aligned}
\mathcal{L}(Y, \lambda, \mu, s, \upsilon) &= C(Y) + \frac{\upsilon}{2}\sum_{i=1}^{m}\{f_i(Y) - f_{Hi} + s_{Hi}\}^2 + \frac{\upsilon}{2}\sum_{i=1}^{m}\{f_{Li} - f_i(Y) + s_{Li}\}^2 \\
&+ \sum_{i=1}^{n}\lambda_i h_i(Y) + \sum_{i=1}^{m}\mu_{Hi}(f_i(Y) - f_{Hi} + s_{Hi}) + \sum_{i=1}^{m}\mu_{Li}(f_{Li} - f_i(Y) + s_{Li}). \qquad (2.57)
\end{aligned}
$$

The Newton method is then applied to solve for the KKT conditions corresponding to the above Lagrange function. The algorithm of iteratively updating multipliers λ and μ, and penalty factor υ is described in detail in [29].

Crisan and Mohtadi [25] introduce an efficient technique to update the set of binding inequality constraints for the iterative Newton's method. This method is claimed to be fast and have less cycling of the binding constraints entering and leaving the active set. They release the nonbinding constraints from the active set by determining the signs of associating μ's. However, the addition of the new inequality constraints to the active set must be done more carefully. Suppose we consider only the inequality constraint on a set of control variables $x_i \leq x_{Hi}$. At the beginning of the p^{th} iteration, x_i^p was not in the active set, but suppose the Newton step calculated

at that iteration made it violate its upper limit x_{Hi} (i.e. $x_i^{p+1} > x_{Hi}$). The distance between x_i^p and its limit is denoted by:

$$\Delta x_{vi} = x_{Hi} - x_i^p; \qquad i = 1, \ldots, m_v, \tag{2.58}$$

where m_v is the number of such violations. Note that this Δx_{vi} is not the same as the Newton step for x calculated from (2.53). It is defined as the distance between the current operating point x_i^p and the its limit x_{Hi}. The increment of the Lagrange function due to Δx_{vi} is defined as:

$$\Delta \mathcal{L}_{vi} = \left| \frac{\partial \mathcal{L}}{\partial x_i} \cdot \Delta x_{vi} \right|. \tag{2.59}$$

If $\Delta \mathcal{L}_{vi}$ is large, the effect of Δx_{vi} upon the change of the Lagrange function is strong. The biggest $\Delta \mathcal{L}_{vi}$ in the set of $i \in \{1, ..., m_v\}$ is used to normalize the set of increment of the Lagrange function as follows:

$$NVCS = \frac{(\Delta \mathcal{L}_{vi}, \ldots, \Delta \mathcal{L}_{vm_v})}{\max\{\Delta \mathcal{L}_{vi}, \ldots, \Delta \mathcal{L}_{vm_v}\}}, \tag{2.60}$$

where $NVCS$ defines the "normalized violated constraint set". The violated constraints that will be added to the active set are the ones corresponding to the elements in $NVCS$ which are bigger than a specified value r, where $0 \leq r \leq 1$. The larger the value of r, the fewer the constraints are being added to the active set, and this will slow down the computational process. However, if r is too small, too many constraints are being added at the same time and it may cause the problem to move away from the actual solution. Crisan and Mohtadi suggest that r be 0.5.

2.3.2 OPF by Primal-Dual Interior-Point Method

Due to the difficulty of identifying the set of binding inequality constraints, or the active set in the Newton method, the Primal-Dual Interior-Point (PDIP) method for solving optimization problems in power systems has been developed [30, 31, 32, 33, 34, 35, 36]. The interior point method for optimization has been widely known since the publication of Karmarkar in 1984 [37].

To apply the primal-dual algorithm to the OPF problem, we rewrite the problem stated in (2.42) in a simpler form as follows:

$$\begin{aligned} \min_{Y} \quad & C(Y) \\ \text{subject to:} & \\ & h_i(Y) = 0; \quad i = 1, \ldots, n, \\ & f_i(Y) \leq 0; \quad i = 1, \ldots, 2m. \end{aligned} \tag{2.61}$$

The inequality constraints are now eliminated by introducing the *logarithmic barrier function* as:

$$\mathcal{B}(Y, \lambda, \nu) = C(Y) + \sum_{i=1}^{n} \lambda_i h_i(Y) - \nu \sum_{i=1}^{2m} \ln |f_i(Y)|, \tag{2.62}$$

where $\nu > 0$ is the *barrier parameter* that will be iteratively adjusted toward zero in order to approach the solution. The adjustment of the barrier parameter will be discussed later on.

The gradient of the barrier function with respect to control and state variables Y, and multipliers λ can be written as:

$$\nabla_Y \mathcal{B}(Y, \lambda, \nu) = \nabla_Y C(Y) + \sum_{i=1}^{n} \lambda_i \nabla_Y h_i(Y) - \nu \left[\sum_{i=1}^{2m} \frac{1}{f_i(Y)} \nabla_Y f_i(Y) \right], \tag{2.63}$$

$$\nabla_\lambda \mathcal{B}(Y,\lambda,\nu) \;=\; h(Y). \tag{2.64}$$

By introducing *slack variables* s and another set of multipliers for inequality constraints μ as

$$s_i \;=\; -f_i(Y), \tag{2.65}$$

$$\mu_i \;=\; -\frac{\nu}{f_i(Y)} \;=\; \frac{\nu}{s_i}, \tag{2.66}$$

the first order necessary optimality conditions for the barrier function $\mathcal{B}$ are

$$\nabla_Y C(Y) + \sum_{i=1}^{n} \lambda_i \nabla_Y h_i(Y) + \sum_{i=1}^{2m} \mu_i \nabla_Y f_i(Y) \;= 0, \tag{2.67}$$

$$h_i(Y) \qquad = 0; \qquad i = 1,\ldots,n, \tag{2.68}$$

$$f_i(Y) + s_i \qquad = 0; \qquad i = 1,\ldots,2m, \tag{2.69}$$

$$\mu_i s_i - \nu \qquad = 0; \qquad i = 1,\ldots,2m, \tag{2.70}$$

where

$$\mu_i \;\geq\; 0, \tag{2.71}$$

$$s_i \;\geq\; 0. \tag{2.72}$$

Equations (2.67)-(2.70) can be denoted by $g_{\mathcal{B}}(z) = 0$, where $z = [Y, \lambda, \mu, s]^T$, and can be solved by the Newton-Raphson method described as follows. By applying the Taylor's series expansion around a current point z^p, the first-order optimality conditions can be written as:

$$0 = g_{\mathcal{B}}(z) = g_{\mathcal{B}}(z^p) + \left.\frac{d\, g_{\mathcal{B}}(z)}{dz}\right|_{z=z^p} \cdot (z - z^p) + \text{H.O.T.}, \tag{2.73}$$

where H.O.T is the term with the derivatives of the orders higher than one, and can be ignored. Therefore, by defying $\Delta z = z - z^p$, the first-order optimality conditions

can be rewritten as:

$$\left.\frac{d\, g_{\mathcal{B}}(z)}{dz}\right|_{z=z^p} \cdot \Delta z = -g_{\mathcal{B}}(z^p), \tag{2.74}$$

or

$$W_{\mathcal{B}} \cdot \Delta z = -g_{\mathcal{B}}, \tag{2.75}$$

which is the same as (2.49) in Section 2.3.1. However, the difference between solving an optimization problem with Newton's method and with the primal-dual interior-point method is that the Hessian matrix (or W matrix) of the Newton method is *symmetric*, but the Hessian matrix of the PDIP method ($W_{\mathcal{B}}$) is not. In order to verify $W_{\mathcal{B}}$ matrix, let us rewrite $g_{\mathcal{B}}(z)$ in matrix form as:

$$g_{\mathcal{B}}(z) = \begin{bmatrix} \frac{\partial \mathcal{B}}{\partial Y} \\ h(Y) \\ f(Y) + s \\ \mu\, s - \nu \end{bmatrix} = 0. \tag{2.76}$$

Therefore, the first-order optimality conditions for PDIP algorithm in (2.74) can be expressed in matrix form as:

$$\begin{bmatrix} H & J^{\mathrm{T}} & A^{\mathrm{T}} & 0 \\ J & 0 & 0 & 0 \\ A & 0 & 0 & \mathbf{I} \\ 0 & 0 & \mathbf{S} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \Delta Y \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathcal{B}}{\partial Y} \\ h(Y) \\ f(Y) + s \\ \mu\, s - \nu \end{bmatrix}, \tag{2.77}$$

where

$$H = \frac{\partial^2 \mathcal{B}(z)}{\partial Y^2},$$

$$J = \frac{\partial h_i(Y)}{\partial Y}; \quad i = 1, \ldots, n,$$

$$A = \frac{\partial f_i(Y)}{\partial Y}; \quad i = 1, \ldots, 2m,$$

and

I : an identity matrix

S : a diagonal matrix constructed from $(s_1, s_2, \ldots, s_{2m})$

M: a diagonal matrix constructed from $(\mu_1, \mu_2, \ldots, \mu_{2m})$

Assuming that we start out with feasible set of initial variables and positive slacks and multipliers, the new approximation is determined by

$$z^{p+1} = z^p + \alpha \Delta z, \tag{2.78}$$

where $\Delta z = [\Delta Y, \Delta \lambda, \Delta \mu, \Delta s]^T$ is the Newton step solved from (2.75), and α is the *step length* chosen to satisfy the non-negativity conditions on μ and s in (2.71) and (2.72), respectively. The evaluation of α is done by applying the *minimum ratio test*:

$$\alpha^\dagger = \min \left\{ \frac{-\mu_i}{\Delta \mu_i}, \frac{-s_i}{\Delta s_i} \right\}; \qquad \forall i : \Delta \mu_i, \Delta s_i < 0, \tag{2.79}$$

and then

$$\alpha = \min \left\{ \sigma \alpha^\dagger, 1 \right\}, \tag{2.80}$$

where σ is the number chosen to be a little less than 1 to prevent μ and s from being zero without interfering with the step length. Recent studies have suggested σ be 0.9995 [33, 35]. Since σ forces the variables to stay inside the bound without touching

it, the variables are *strictly feasible* with respect to inequality constraints, that is $\forall i$, $\mu_i > 0$ and $s_i > 0$ at every iteration. This property yields the term *interior-point*, and the above primal-dual method becomes the *primal-dual interior-point* method.

For each $\nu > 0$, the set of solution of (2.67)-(2.70) computed by the Newton method is $z_\nu = (Y_\nu, \lambda_\nu, \mu_\nu, s_\nu)$. The points z_ν, where $0 < \nu < \infty$, are called the *central path* of the barrier function. Once ν is small enough to be considered approximately zero, (2.70) becomes $\mu_i s_i \simeq 0$, which implies that for each $i = 1, \ldots, 2m$, only one of μ_i or s_i must be approximately zero. This condition is called the *complementarity condition* because μ_i and s_i are nonzero in complementary locations [38].

Compared to the Newton method, the complementarity condition in primal-dual interior-point method can be used to determine the active set of inequality constraints once ν is close enough to zero. If μ_i is positive, $s_i \simeq 0$ (due to the interior-point property), the i^{th} inequality constraint is considered binding at its limit (even though it is not exactly at the limit) because (2.69) becomes $f_i(Y) \simeq 0$. If s_i is positive, then $f_i(Y) < 0$ and the i^{th} inequality constraint is not binding, and μ_i has to be approximately zero.

The set of solution obtained from driving ν toward zero, $(Y^*, \lambda^*, \mu^*, s^*)$, is considered the solution of the problem stated in (2.61).

The Path-Following Method

For $\nu = 0$, the point z_ν is the exact solution of the KKT condition. However, we should start out with fairly large $\nu > 0$ and tend to reduce ν toward zero as we

go along the central paths. Once we are close to a central path, the solution set of (2.67)-(2.70) is obtained for a positive ν, we then reduce the value of ν by multiplying it with $0 < r < 1$, and go toward the new central path.

The method used to consider if we are close enough to the central path is called the *path following method.* The path following method introduces two most interesting *neighborhoods* of the central path: the 2-norm neighborhood $\mathcal{N}_2(\vartheta)$, and the one-side ∞-norm neighborhood $\mathcal{N}_{-\infty}(\zeta)$. Both types of neighborhood are defined as follows [38]:

$$\mathcal{N}_2(\vartheta) = \left\{(Y,\lambda,\mu,s) \in \mathcal{F}^0 \,\middle|\, \| \mathbf{S}\mathbf{M}e - \nu e \|_2 \leq \vartheta\nu\right\};\ \exists\vartheta \in (0,1), \quad (2.81)$$

$$\mathcal{N}_{-\infty}(\zeta) = \left\{(Y,\lambda,\mu,s) \in \mathcal{F}^0 \,\middle|\, \mu_i s_i \geq \zeta\nu \quad :\forall i\right\};\ \exists\zeta \in (0,1). \quad (2.82)$$

where $\mathcal{F}^0$ is the primal-dual *strictly feasible set* defined as

$$\mathcal{F}^0 = \{(Y,\lambda,\mu,s) \mid g_B = 0,\ (\mu,s) > 0\}. \quad (2.83)$$

The typical values of the parameters are $\vartheta = 0.5$ and $\zeta = 10^{-3}$.

2.4 Power System Security

So far, we have seen several methods of minimizing generation cost in power systems operating at the normal operating state. However, the optimal dispatch of the system may not be a *secure* one. Security in a power system is the ability of the system to continue supplying power to load when there is a disturbance (or *contingency*) in the system. A contingency might be the loss of one or more transmission lines if the lines were short-circuited or damaged; such lines are automatically disconnected from the

rest of the system by the system protection relays. Another common contingency is the loss of on-line generators due to equipment failures or action by protective devices. Each contingency causes changes in power flows in the remaining lines, and voltage magnitudes and angles of the remaining buses. Frequently, the changes in system state as a result of every such contingency are within the acceptable operational limits due to the redundancy in designing generators and transmission lines. Under these circumstances, the system is able to continue serving loads and is considered *secure*. However, if the post-contingency limits and the pre-contingency operating points prevent the system from being redispatched to survive the contingency, cascading outages throughout the system may result. This phenomenon is referred to as a *blackout* in power systems.

During the normal operating state, system security can be maintained by keeping enough spinning reserves on generators and the proper amount of transmission reserves on hand to ensure that the system will be able to be rescheduled to be within operational limits after any credible contingency which might arise.

Stott et al [5] illustrate the levels of static security in power systems as shown in Figure 2.4, in which the arrows represent the transitions between the levels. The transitions are due to the changes of loads and generation, and the rescheduling actions allowed in the systems. Levels 1 and 2 represent systems that are operating at normal operating state and secure. Level 1 represents the *ideal* security in power system. Systems in Level 1 survive any credible contingencies without depending on the post-contingency corrective rescheduling. Systems in Level 2 have post-contingency rescheduling ability to remove any operating constraint violations caused by credible

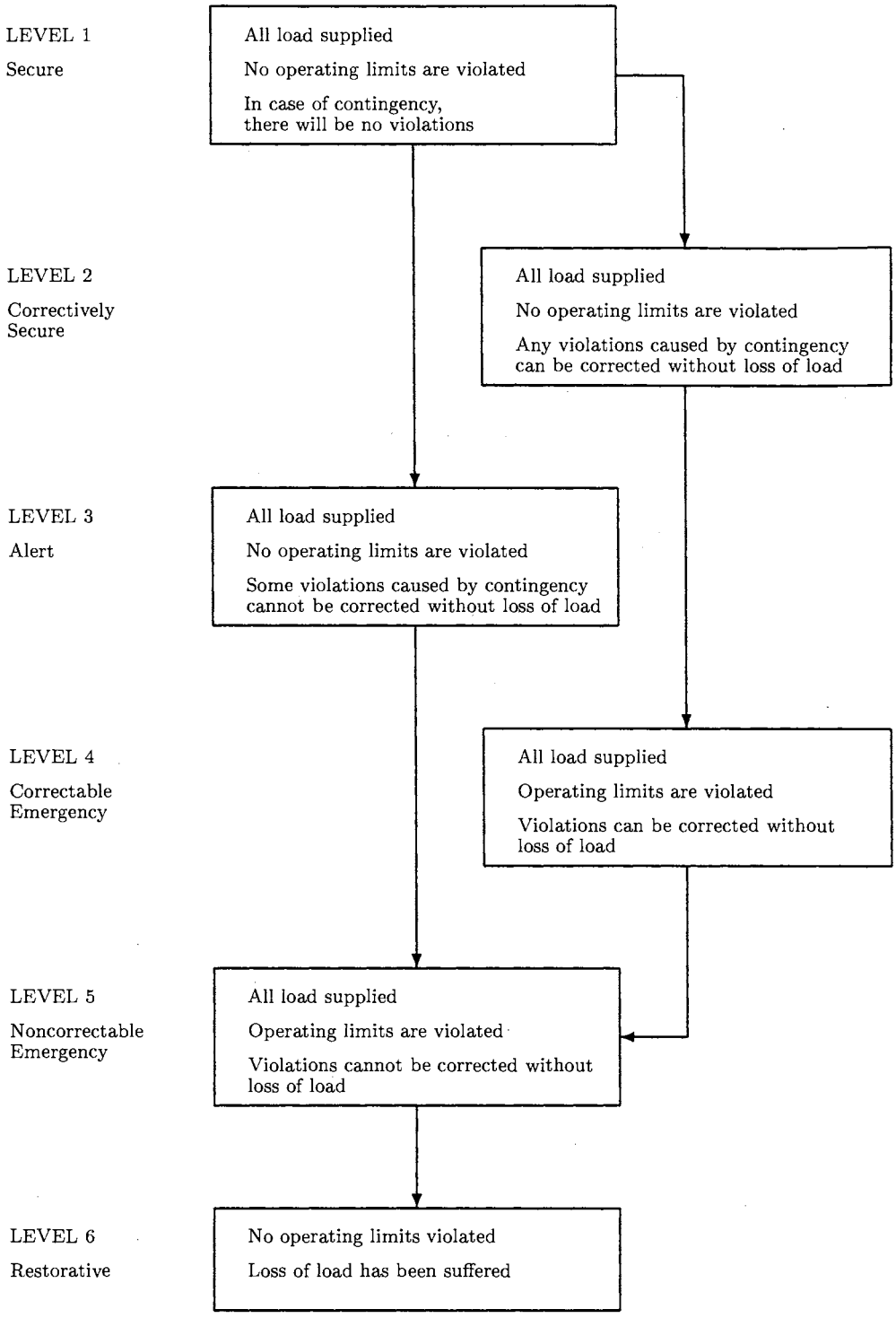

Figure 2.4: Static security levels in power systems.

contingencies within a specified period of time. Thus it is more economical than Level 1. Level 3 is not considered a secure operating system due to the loss of load if some contingencies occur. The system in Level 3 can be brought to Level 1 or 2 by performing the generator rescheduling, which is considered the *preventive rescheduling.* The "loss of load" we are referring to in this research is the customer load that is interrupted without advance information and/or agreement. In case the customers are willing to be interrupted when contingencies occur, the *interruptible load* is not considered as loss of load in a power system. Instead, it is considered the *demand side rescheduling* to which the system has access in case of emergencies.

A power system is in an *emergency* condition when some of the operating limits are violated (Levels 4 and 5). However, the system in Level 4 can be brought back to Level 2 by activating some required *corrective rescheduling.* A power system may fall into Level 4 and 5 because of the contingencies.

Wood and Wollenberg [7] specify three functions of the operation control for system security as follows:

1. System monitoring: This function provides the system operator the on-line information of the power system. The critical system quantities that must be measured and sent to the control center are voltages, currents, power flows in transmission lines, the status of the circuit breakers, and switches at every substation (or *bus*) in the system. The incoming information must be checked with their limits. The limits violation should set off the alarm for the system operator to take actions.

2. Contingency analysis: The contingency analysis is an important tool for determining the emergency states whether or not they should take place in a power system. The contingency analysis should be done frequently at different normal on-line operating states. At each normal operating state, the contingencies are ranked based on their severity. The ones that cause emergency states must be included in the list of *credible* contingencies. The power system operator may respond to the credible contingency by changing the pre-contingency operating state to eliminate the emergency state if the probability of the contingency is significant. Another way of taking care of the emergency state resulting from a credible contingency is by developing a post-contingency control strategy that will reschedule the the system to recover the emergency. However if the probability of the contingency is unlikely, the operator may choose to do nothing at the pre-contingency state. Note that the approach of the contingency analysis is not in the scope of this research. The further reading on this subject is in Section II of [5].

3. Security-constrained optimal power flow: The security-constrained OPF can be performed by combining the optimal power flow with the response from the contingency analysis. The detail of this function is described in the next section.

2.5 Security-Constrained Optimal Power Flow (SCOPF)

In order to study the security-constrained optimal power flow, Wood and Wollenberg [7] divide the power system operations into four states:

	Gen 1	Gen 2
Marginal cost	\$1/MW	\$2/MW
Minimum P_{Gi}	50 MW	0 MW
Maximum P_{Gi}	200 MW	120 MW
	Line 1	Line 2
Maximum line flow	100 MW	120 MW

Table 2.2: Characteristics of two-bus system.

- Optimal dispatch: This state takes place prior to any contingency. The system is operating under normal operating states at the minimum generation cost.

- Post contingency: This is the state just after a contingency occurs. As a result, the system limits such as power flows in the remaining lines might be violated.

- Security-constrained dispatch: The system is operating under normal conditions (i.e. no contingencies occur), but the system variables are adjusted from the optimal dispatch condition so that there are no system violations when a contingency occurs.

- Security-constrained post contingency: During a contingency, the system is operating within the system operating limits since the security-constrained dispatch was applied prior to the contingency.

The relationship of the four operating states described above are shown in Figure 2.5. A simple 2-bus system shown in Figure 2.6 is used to illustrate the SCOPF problem. This example is taken from [4]. Table 2.2 shows the characteristics of the generators, and the flow limits of the transmission lines for the 2-bus system.

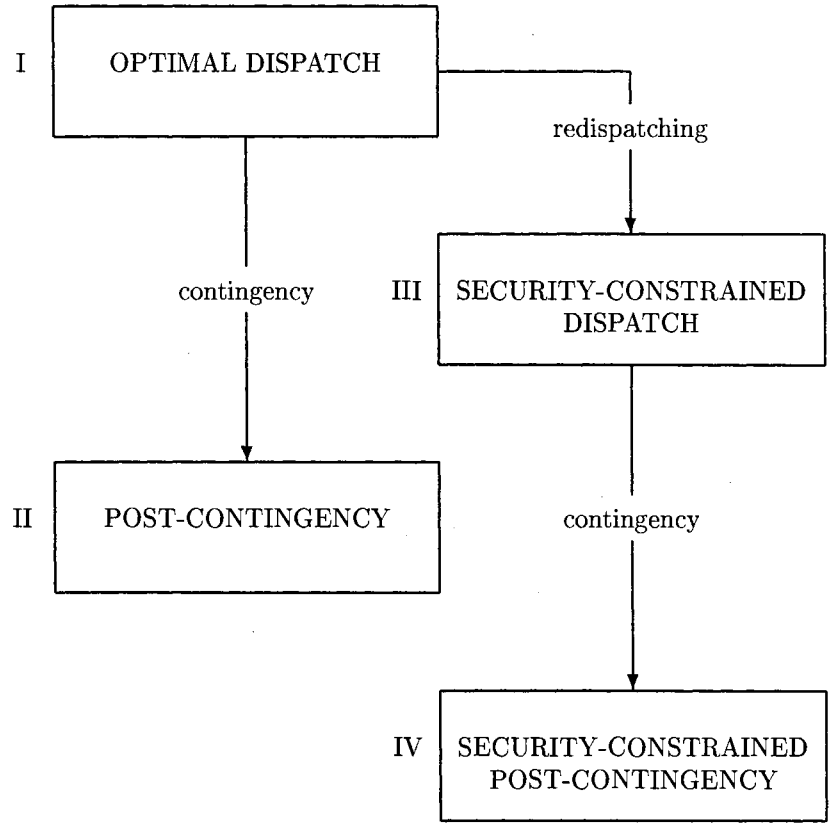

Figure 2.5: Operating states in power system regarding system security.

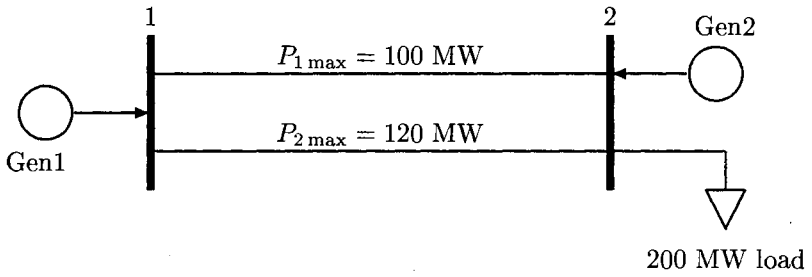

Figure 2.6: Two-bus system

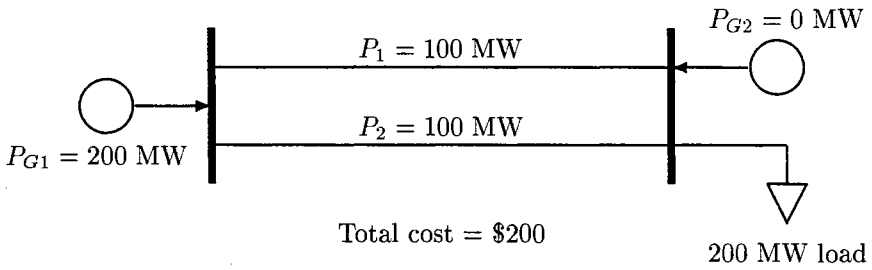

Figure 2.7: Optimal dispatch for two bus system.

Since the marginal cost of the generator at bus 1 (generator 1) is lower than that of the generator at bus 2 (generator 2), the standard optimal dispatch is as shown in Figure 2.7 where generator 1 picks up the entire load without violating its maximum capacity limit. The optimal cost of generation is therefore (200 MW)(\$1/MW) = \$200. Assuming that both transmission lines (line 1 and line 2) in the system are identical even though they have different maximum flow limits, then 200 MW power generated from generator 1 is split equally in both lines as shown. Suppose that only contingencies on transmission lines are of concern in this system. There are two possible and hence *credible* contingencies in this system: the outages on line 1 and line 2. However, if the system can survive the outage on line 2, then it will certainly survive the outage on line 1. This is because line 2 has more power flow capacity than line 1. Therefore, we will discuss only the contingency on line 2.

By suddenly losing line 2 from the system for any reason, line 1 must carry the entire 200 MW generated from generator 1, that is $P_1 = 200$ MW, should the optimal dispatch based on the marginal costs be dispatched prior to the contingency, as shown in Figure 2.8. This state is considered the *post-contingency state* in the power system. However, it is obvious that line 1 is overload since its maximum flow

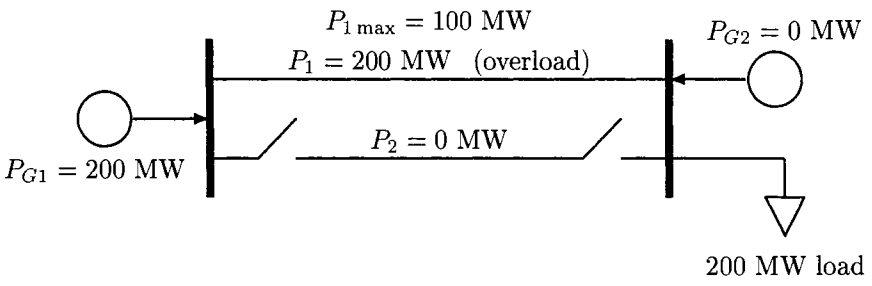

Figure 2.8: Post-contingency state.

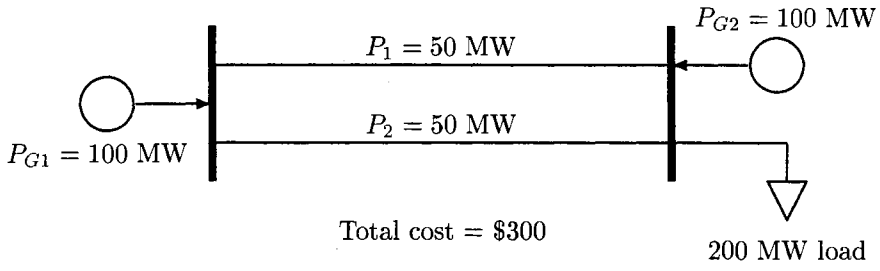

Figure 2.9: Optimal security-constrained dispatch.

limit is only 100 MW. If we let the transmission line be overloaded for quite some time, it will be damaged from overheating and the breakers on both sides of the line will trip the line out of the system. That results in generator 1 being disconnected from the rest of the system. Unfortunately, generator 2 alone is not able to serve 200 MW load due to its maximum capacity. As a result, the breaker of generator 2 will trip and disconnect the generator from the bus to prevent a severe damage on the generator equipment. Finally, the system completely fails to serve the entire load. This situation is considered an *insecure* power system since the system loses the ability to serve load after a credible contingency.

If the security of the power system is taken into account, when either line 1 or line 2 is out of service, the entire 200 MW load should still be provided without damaging the remaining transmission line and on-line generators. In order to survive

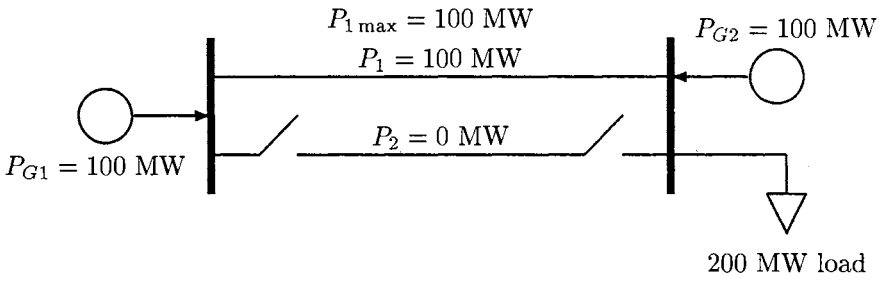

Figure 2.10: Post-contingency state of security-constrained dispatch.

contingency on line 2 (and hence survive contingency on line 1), generator 1 should not generate more than 100 MW during the normal operating state because the tightest flow limit is 100 MW on line 1. Since the total load is 200 MW, generator 2 has to pick up another 100 MW. This is considered the security-constrained dispatch and is shown in Figure 2.9. At this state, the system is operated in the way that it can normally serve the total load when any contingency (i.e. outage on line 1 or line 2) occurs. Thus the system operating in Figure 2.9 is considered secure. The system generation cost is now (100 MW)($1/MW)+ (100 MW)($2/MW) = $300. The operating cost is higher than that of the optimal dispatch system without security shown in Figure 2.7 because generator 2 which is the more expensive generator, has to pick up some load in order for the system to stay secure at all time. Figure 2.10 shows the post-contingency state of the secure dispatch with outage on line 2. This system stays secure since there is no overload on the remaining line or generators.

Mathematical Model of SCOPF

From the recent study [4], the security-constrained optimal power flow (SCOPF) model is developed from the standard optimal power flow (OPF) model by adding

the operating constraints for credible contingencies. The additional constraint function for contingency k is denoted by $f_m^k(Y^0)$, which is a function of pre-contingency variables Y^0. Assuming that the total number of credible contingencies is K, the model for SCOPF is given as follows.

$$\begin{aligned} \min_{Y^0} \quad & C(Y^0) \\ \text{subject to:} & \\ & h_i^0(Y^0) = 0, \qquad i = 1, \ldots, n^0, \\ f_{Li}^0 \le & f_i^0(Y^0) \le f_{Hi}^0, \quad i = 1, \ldots, m^0, \\ & h_i^k(Y^0) = 0, \qquad i = 1, \ldots, n^k, \; k = 1, \ldots, K, \\ f_{Li}^k \le & f_i^k(Y^0) \le f_{Hi}^k, \quad i = 1, \ldots, m^k, \; k = 1, \ldots, K. \end{aligned} \tag{2.84}$$

The superscripts $(\cdot)^0$ and $(\cdot)^k$ denote the normal operating state and the post-contingency state, respectively.

2.6 SCOPF with Post-Contingency Rescheduling

It is often considered that the system operated under the optimal strategy taking into account the system security is not as economical as the one operated under the standard optimal dispatch. However, studies have stated that the security-constrained dispatch described above is too conservative in the sense that the system operator does not consider the fact that the system can be partly adjusted (or corrected) after the contingencies [3, 4, 5, 6]. The corrections that can be done after the contingencies include generation rescheduling, switching, overload rotation [4], and partially drop-

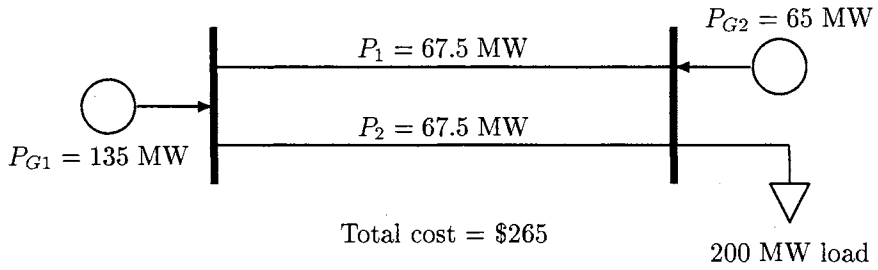

Figure 2.11: Optimal security-constrained dispatch with corrective rescheduling.

ping customer loads [3]. In [4], the study of SCOPF with post-contingency corrective rescheduling on generators is introduced. The capability of generator rescheduling depends on the spinning reserves and generator ramping rates.

	Gen 1	Gen 2
$\Delta^{+}_{\max} P_{Gi}$	40 MW	35 MW
$\Delta^{-}_{\max} P_{Gi}$	40 MW	35 MW

Table 2.3: Ramping limit capacities of two-bus system generators.

The *ramping rate* is the maximum total change of the real power generation within a specific period of time[1], which depends on the setting time frame of the protective devices. If the generator rescheduling were considered in the system with the ramping limits shown in Table 2.3, where $\Delta^{+}_{\max} P_{Gi}$ is the maximum ramping up rate and $\Delta^{-}_{\max} P_{Gi}$ is the maximum ramping down rate, the security-constrained dispatch with corrective rescheduling would be as shown in Figure 2.11.

The secure post-contingency dispatch requires that generator 2 generate 100 MW when line 2 is out. Since the generator ramping rate of generator 2 allows the generator to increase the output by 35 MW in a short time period. the security-

[1]This is considered an *emergency time limit.* It must not exceed the time period the breakers need in order to detect any emergency limit and trip the on-line equipment out of service.

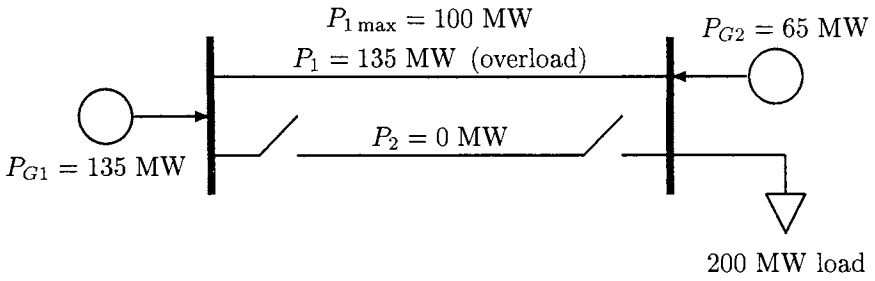

Figure 2.12: Post-contingency emergency state.

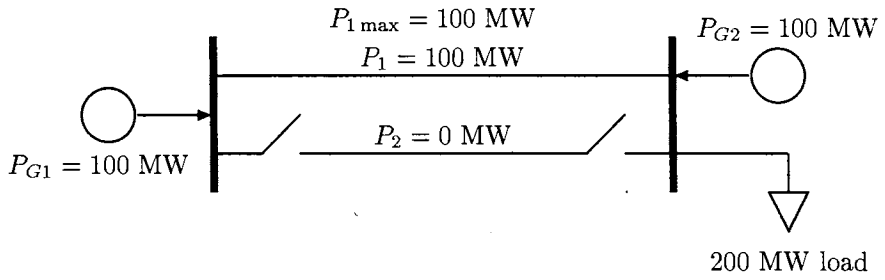

Figure 2.13: Post-contingency state of SCOPF with rescheduling.

constrained dispatch taking into account the corrective rescheduling results in generator 2 being scheduled for 65 MW. Consequently, generator 1 must be generating 135 MW in order to serve the total load at the pre-contingency state. When the contingency on line 2 occurs, generator 1 is generating 135 MW which will overload line 1 (see Figure 2.12). However, the capability of generator rescheduling allows generator 1 to drop from 135 MW to 100 MW within the emergency time limit before the breakers trip line 1 out due to the overload. Meanwhile, generator 2 successfully ramps from 65 MW to 100 MW to cover the rest of the load. The total generation cost for this operating system is, therefore, (135 MW)($1/MW)+(65 MW)($2/MW) = $265, which is lower than the total generation cost of the security-constrained dispatch without corrective rescheduling. The total generation costs of the 2-bus system for three cases discussed above are summarized in Table 2.4.

Dispatch Types	Total Generation Cost
Optimal dispatch	\$200
SCOPF dispatch	\$300
SCOPF with rescheduling	\$265

Table 2.4: Total generation costs for three different dispatches.

Mathematical Model of SCOPF with Post-Contingency Rescheduling

The SCOPF model presented in Section 2.5 does not take into account the ability of rescheduling the operating state variables. If the system were allowed to adjust some parameters within the time limit, the modified SCOPF model should be expressed as:

$$\begin{aligned} \min_{Y^0, Y^k} \quad & C(Y^0) \\ \text{subject to:} \quad & \\ & h_i^0(Y^0) = 0, \qquad i = 1, \ldots, n^0, \\ & f_{Li}^0 \leq f_i^0(Y^0) \leq f_{Hi}^0, \quad i = 1, \ldots, m^0, \\ & h_i^k(Y^k) = 0, \qquad i = 1, \ldots, n^k, \ k = 1, \ldots, K, \\ & f_{Li}^k \leq f_i^k(Y^k) \leq f_{Hi}^k, \quad i = 1, \ldots, m^k, \ k = 1, \ldots, K, \\ & d_i^k(Y^k, Y^0) \leq 0, \qquad i = 1, \ldots, m^{k0}, \ k = 1, \ldots, K, \end{aligned} \tag{2.85}$$

where the *coupling constraint*, $d^k(Y^k, Y^0)$, represents the set of rescheduling capabilities of the operating points at contingency k.

The post-contingency state Y^k may be changed from the pre-contingency state Y^0 by the amount of Δ^k as expressed in the last constraint in (2.85). The relationships

between the two operating states are considered the *coupling* constraints. The system is now able to optimally dispatch the pre-contingency state Y^0 based on the fact the it may be adjusted later on to satisfy the contingency constraints. Therefore the optimal pre-contingency system can be secure without being too conservative.

Monticelli [4] states that the coupling constraints act like the joints between standard OPF problem (2.42) and the SCOPF problem (2.84).

In [3], in addition to the generator rescheduling, the rescheduling can also come from the customer side as the amount of interruptible loads during contingencies. By this system security scheme, not only do the generators have available additional capacity for rescheduling, but the customers also allow the system to interrupt some of their loads to maintain system security. This security concept is known as the "stand-by capacity requirement". However, the authors in [3] feel that it may be unfair to the participants due to the unknown pattern of compensated costs for both generators and customers.

Infeasibility of SCOPF with Post-Contingency Rescheduling

In [4], the *penalty variables* are added to the post-contingency constraints of (2.85) in order to ensure the mathematical feasibility of the post-contingency state at any given pre-contingency state. If the post-contingency operating point is feasible, these variables are zero. Thus the *penalty costs* associated with the constraint violations can be used as an objective function to be minimized. The penalty for feasibility of the post-contingency operating state at contingency k is defined as follows.

$$\omega^k(Y^0) = \min_{\rho} \quad C_\rho \cdot \rho$$

subject to: (2.86)

$$
\begin{aligned}
h_i^k(Y^k) &= 0, && i = 1, \ldots, n^k, \\
f_{Li}^k - \rho_{Li} \leq f_i^k(Y^k) &\leq f_{Hi}^k + \rho_{Hi}, && i = 1, \ldots, m^k, \\
d_i^k(Y^k, Y^0) &\leq \rho_{di}, && i = 1, \ldots, m^{k0},
\end{aligned}
$$

where $C_\rho \gg 0$ is the penalty cost coefficient, and ρ is a vector of penalty variables, which consists of $(\rho_{Hi}, \rho_{Li}, \rho_{di})$. The cost coefficients of the penalty variables are assumed to be the same and equal to C_ρ for convenience.

The penalty function ω^k is a function of the pre-contingency operating point Y^0. If $\omega^k(Y^0) = 0$, the post-contingency state is feasible. On the other hand, if $\omega^k(Y^0) > 0$, the post-contingency is not feasible. The problem stated in (2.85) can then be solved by a decomposition method as follows:

- For a given Y^0, the penalty function $\omega^k(Y^0)$ is obtained by solving equation (2.86).

- The SCOPF problem in (2.85) can be written in terms of Y^0 as:

$$
\begin{aligned}
&\min_{Y^0} && C(Y^0) \\
&\text{subject to:} && \\
&h_i^0(Y^0) = 0, && i = 1, \ldots, n^0, \\
&f_{Li}^0 \leq f_i^0(Y^0) \leq f_{Hi}^0, && i = 1, \ldots, m^0, \\
&\omega^k(Y^0) \leq 0 && k = 1, \ldots, K.
\end{aligned}
\tag{2.87}
$$

If the information of $\omega^k(Y^0)$ for every contingency k is presented for a given Y^0, the SCOPF with post-contingency rescheduling problem in (2.87) can be solved

for Y^0 that satisfies all the constraints. The new Y^0 must be used to solve the $\omega^k(Y^0)$ again. This process repeats until the optimality conditions of (2.86) and (2.87) are satisfied simultaneously.

CHAPTER 3

PROBLEM FORMULATION

The OPF problem taking into account system security proposed in this research handles system security as an *economic cost* instead of a *constraint* as has been done in previous studies (see problems (2.84) and (2.85) in Section 2.5). When a credible contingency occurs, the power system must redispatch from the current normal operating points (or pre-contingency state) to survive the contingency as described in Section 2.5. Consequently, redispatch of the power system at the post-contingency state yields a new total cost of system operation, which is considered the *security cost* in this research. Therefore, each contingency has a security cost that depends on the pre-contingency state.

> The security cost of a particular contingency k is the minimum cost of the system operating within operational limits at the post-contingency state given the system has been operating at the pre-contingency state Y^0 prior to the contingency.

At the post-contingency state, system operational limits may be *emergency limits* instead of normal operating limits. Moreover, the system takes into account the

ability of post-contingency rescheduling. The available post-contingency rescheduling methods considered in this research are *generator rescheduling* and *customer load interruption.* Generator rescheduling can be done by gradually ramping generators up or down within a short time period (considered an emergency time limit) in order to survive the contingency. Generator rescheduling may require some additional cost other than the regular operation cost (or fuel cost).

We have the ability to interrupt some load when the contingency occurs. Interrupting load is considered *demand side rescheduling.* We can only interrupt the group of customers who have agreed to participate in demand side rescheduling. Interruptible customers must be compensated by the system for each unit of load that is interrupted. Both load interruption cost and generator rescheduling cost are included in the system operating cost at the post-contingency state.

If security costs of all credible contingencies had been taken into account in the optimal pre-contingency state, the system might not have optimally dispatched the way it did when the information of security costs was not provided. Instead, it would have dispatched at a pre-contingency state such that the security costs would be as low as possible when contingencies occur. However, the information on security costs alone is not sufficient to determine the optimal pre-contingency state because during the normal operating state, one does not know which contingency will occur. Each contingency has a different probability of occurrence. A contingency that causes high security cost may not be likely to happen, while a contingency with lower security cost may be more likely. The pre-contingency optimal dispatch process should consider the significance of contingencies by looking at both post-contingency security costs

and their probabilities of occurrence. Therefore, the security costs taken into account in the pre-contingency state must be incorporated in terms of *expected* security costs by multiplying each of them with its corresponding *contingency probability.* The above discussion can be accomplished by introducing an OPF model that can handle security constraints at post-contingency states in terms of economic costs.

The problem formulation introduced in this research is an optimization problem that minimizes a cost objective function including expected security costs subject to sets of nonlinear equality and inequality constraints. Sets of equality constraints are power balance equations (see Section 2.2). Sets of inequality constraints are system operating limits, which include

- voltage limits,
- limits on tap ratios and phase shifters,
- capacity limits on real and reactive power generations,
- line flow limits,
- constraints on real power load consumed by customers.

The multipliers for these constraints have important economic interpretations. The multipliers on real power balance equations are the optimal *bus prices* of the power system. The multipliers on the capacity limits on the real power generation at the post-contingency state can be used to compute the *marginal value of spinning reserve* for that generator. Likewise, the multipliers on the constraints on real power load at the post-contingency state can be used to compute the *marginal value of interruptible load.* The detail of the economic interpretations of these multipliers will be discussed later when we introduce them in the solution methodology in Chapter 4.

Then, the marginal values of spinning reserve and interruptible load will be calculated when we obtain the results from the test cases in Chapter 5.

The model, expressed in (3.1)-(3.6), represents an OPF problem taking into account system security in terms of expected security costs, or the *expected security-costs optimal power flow* (ESCOPF) problem.

3.1 Main Problem

The *main problem* of ESCOPF can be expressed in (3.1), (3.2), and (3.3), as functions of the pre-contingency operating state Y^0, where Y^0 consists of

- vector of control variables X^0, which includes voltage magnitudes $|V_i^0|$, voltage angles θ_i^0, transformer tap ratios T_{ij}^0, and phase shifters ϕ_{ij}^0,
- vector of independent state variables, which includes real power generations P_{Gi}^0, reactive power generations Q_{Gi}^0, line current magnitudes $I_{ij}^0 = |I_{ij}(X^0)|$, and real loads P_{Li}^0.

The probability of the system operating at the normal operating state is $\pi^0 = [1 - \sum_{k=1}^{K} \pi^k]$, where π^k is the probability of contingency k, and K is the total number of credible contingencies.

The objective function of the main problem, also called the *expected cost of system operation* (3.1), consists of two parts:

1. generation costs and customer benefits evaluated at the normal operating state (or pre-contingency state), multiplied by the probability of the normal operating state π^0,

$$\min_{Y^0}\left\{[1-\sum_{k=1}^{K}\pi^k]\left[\sum_{i\in\mathcal{G}}C_{Gi}(P^0_{Gi})-\sum_{i\in\mathcal{I}}B_{Ii}(P^0_{Li})-\sum_{i\in\mathcal{U}}B_{Ui}(P^0_{Li})\right]+\sum_{k=1}^{K}\pi^k S^k(P^0_{Gi},P^0_{Li})\right\} \tag{3.1}$$

subject to

$$\begin{aligned} P^0_i(X^0) &= P^0_{Gi}-P^0_{Li} && :\forall\, i,\\ Q^0_i(X^0) &= Q^0_{Gi}-Q^0_{Li}(P^0_{Li}) && :\forall\, i, \end{aligned} \tag{3.2}$$

$$\begin{aligned} X^0_{\min} \le\ & X^0 \le X^0_{\max}, && \\ P^0_{Gi_{\min}} \le\ & P^0_{Gi} \le P^0_{Gi_{\max}} && :\forall\, i\in\mathcal{G},\\ Q^0_{Gi_{\min}} \le\ & Q^0_{Gi} \le Q^0_{Gi_{\max}} && :\forall\, i\in\mathcal{G},\\ & |I_{ij}(X^0)| \le I^0_{ij_{\max}} && :\forall\, ij,\\ 0 \le\ & P^0_{Li} && :\forall\, i\in\{\mathcal{I}\cup\mathcal{U}\}, \end{aligned} \tag{3.3}$$

where

$$\begin{aligned} S^k(P^0_{Gi},P^0_{Li}) = \min_{Y^k}\ & \left\{\sum_{i\in\mathcal{G}}\left[C_{Gi}(P^k_{Gi})+C_{Ri}(|P^k_{Gi}-P^0_{Gi}|)\right]\right.\\ & \left.+\sum_{i\in\mathcal{I}}\left[C_{Ii}(P^0_{Li}-P^k_{Li})-B_{Ii}(P^k_{Li})\right]-\sum_{i\in\mathcal{U}}B_{Ui}(P^0_{Li})\right\} \end{aligned} \tag{3.4}$$

subject to

$$\begin{aligned} P^k_i(X^k) &= P^k_{Gi}-P^k_{Li} && :\forall\, i,\\ Q^k_i(X^k) &= Q^k_{Gi}-Q^k_{Li}(P^k_{Li}) && :\forall\, i, \end{aligned} \tag{3.5}$$

$$\begin{aligned} X^k_{\min} \le\ & X^k \le X^k_{\max}, && \\ \max\left(P^k_{Gi_{\min}},P^0_{Gi}-\Delta^-_{\max}P_{Gi}\right) \le\ & P^k_{Gi} \le \min\left(P^k_{Gi_{\max}},P^0_{Gi}+\Delta^+_{\max}P_{Gi}\right) && :\forall\, i\in\mathcal{G},\\ Q^k_{Gi_{\min}} \le\ & Q^k_{Gi} \le Q^k_{Gi_{\max}} && :\forall\, i\in\mathcal{G},\\ & |I_{ij}(X^k)| \le I^k_{ij_{\max}} && :\forall\, ij,\\ \max\left(0,P^0_{Li}-\Delta^-_{\max}P_{Li}\right) \le\ & P^k_{Li} \le P^0_{Li} && :\forall\, i\in\mathcal{I}. \end{aligned} \tag{3.6}$$

2. expected security cost, which is the sum over all contingencies of the security cost $S^k(Y^0)$ of each contingency k, multiplied by π^k, the corresponding probability of contingency.

There are two sets of customer benefits in the objective function, indicated by the willingness to be interrupted when contingencies occur:

- the set $\mathcal{I}$ denotes the group of customers who are willing to be interrupted if needed, and have the benefit function B_I,
- the set $\mathcal{U}$ denotes the uninterruptible customers with the benefit function B_U.

We assume that customer benefit curves for both types of customers, B_I and B_U, are given, and the customers have enough time and pricing information to react to the optimal bus prices during the normal operating state. Customer benefits are included as negative costs in the objective function. Therefore, the objective function can be viewed as the negative expected *social welfare*[1], and the whole problem is then interpreted as "minimizing negative expected social welfare", which is equivalent to "maximizing expected social welfare" in economic terms, subject to power flow equations and system operating constraints.

Equation (3.2) represents the set of real and reactive power flow (or power balance) equations for every bus i at the pre-contingency state. The real power injected at bus i (P_i), which is a function of control variables X, must be equal to the real power generated at bus i (P_{Gi}) less the real load drawn from bus i (P_{Li}). The same concept applies to the reactive power flow equation. The reactive power load

[1]Social welfare is defined as total benefits minus total costs.

drawn at any load bus is a function of the real power load at that bus, i.e. $Q_{Li}(P_{Li})$, due to the power factor of the load.

Equation (3.3) is the set of power system operating constraints at the normal operating state. The system control and state variables can be anything between their lower and upper bounds in order to minimize the objective function as well as satisfy power flow equations. For both interruptible and uninterruptible customers, their real power consumption must not be negative.

3.2 Subproblems

The security cost of post-contingency state, S^k in (3.1), is obtained by allowing the system to optimally redispatch from the current pre-contingency operating state when contingency k occurs. It is equivalent to solving the optimal power flow (OPF) problem for the post-contingency system[2] given the pre-contingency state. Therefore each S^k can be evaluated separately as shown in (3.4), (3.5), and (3.6). The evaluation of each S^k is called the *contingency subproblem* of the ESCOPF problem.

We choose to incorporate the post-contingency state security costs in our objective function as expected costs because we do not know which contingency will occur, only the probabilities of occurrence. Because we use expected security costs, a high-cost contingency that is rare may have the same expected security cost as a relatively low-cost contingency that is more likely. Hence their effects should be equally important in the pre-contingency state.

[2]The structure (or characteristics) of the post-contingency system might be different from that of the pre-contingency system due to the outage of on-line equipment.

The objective function of S^k is slightly different from that of the main problem since the system allows post-contingency rescheduling by ramping the generators up and down, and by interrupting customer loads. However, the customers should not be interrupted without compensation from the system, and ramping the generators up and down may have additional costs other than the change in steady-state cost due to the change in the generator output. Therefore, interruption costs and additional ramping costs must be included in the post-contingent objective function to be minimized.

Generator Ramping Cost

It is possible to assume that ramping a generator up or down rapidly when contingency occurs has additional cost other than the cost of generation at post-contingency state $C_{Gi}(P_{Gi}^k)$. This could be due to equipment maintenance and labor cost. The generator ramping cost included in the objective function of a subproblem S^k is a function of the change in generator output from pre-contingency to post-contingency state. For convenience, we assume that, in this model, ramping a generator up or down has the same additional cost function denoted by $C_{Ri}(|P_{Gi}^k - P_{Gi}^0|)$ in (3.4).

We may as well introduce another state variable for the amount of generator output ramped, $P_{Ri}^k = |P_{Gi}^k - P_{Gi}^0|$, and the generator ramping cost can be written as $C_{Ri}(P_{Ri}^k)$. However, this set up will add another equation in the set of equality constraints for defining $P_{Ri}^k = |P_{Gi}^k - P_{Gi}^0|$.

Load Interruption Cost

The interruption cost at the interruptible load bus $i \in \mathcal{I}$, C_{Ii}, is a function of the amount of the *load interruption*, which is the difference between pre-contingency and post-contingency load, $P_{Li}^0 - P_{Li}^k$. The interruption cost function appears only in the set of interruptible customers $\mathcal{I}$.

The interruption cost can be expressed in another simple form as $C_{Ii}(P_{Ii}^k)$, where $P_{Ii}^k = P_{Li}^0 - P_{Li}^k$. Although writing the interruption cost in this form makes the objective function of S^k look simpler, it requires an additional equation stating that $P_{Ii}^k = P_{Li}^0 - P_{Li}^k$ in the set of equality constraints. Therefore, we choose to express C_{Ii} as a function of P_{Li}^0 and P_{Li}^k as shown in (3.4).

Coupling Constraints

The subproblem still needs to satisfy power flow equations and operating constraints in the post-contingency state as shown in (3.5) and (3.6). The system operating constraints in the post-contingency state can be different from the ones in the pre-contingency state. In addition to the fact that the system operating limits at post-contingency state are emergency limits (denoted by superscript k), the limits of real power generator P_{Gi}^k and real load P_{Li}^k depend on their pre-contingent values P_{Gi}^0 and P_{Li}^0 due to the ability of post-contingency rescheduling.

During post-contingency states, generators can be ramped up and down within a few minutes after the contingency as part of the post-contingency rescheduling. Real power generation is allowed to move within the maximum ramping rates of the

generators. The maximum rates of ramping generator i up and down are denoted by $\Delta^{+}_{\max} P_{Gi}$ and $\Delta^{-}_{\max} P_{Gi}$, respectively. The upper limit of the real power generation at bus i can be the initial level of generation plus the maximum amount that the generator can be ramped up, $P^{0}_{Gi} + \Delta^{+}_{\max} P_{Gi}$, or the emergency capacity limit on the generator itself, $P^{k}_{Gi_{\max}}$, depending on which one is hit first. The lower limit of real power generation is determined similarly.

We assume that the responses to contingencies must take place very quickly (in a matter of minutes), so the customers do not have time to react to any new bus prices corresponding to post-contingency states. Therefore, the uninterruptible loads stay the same as they were at the pre-contingency state, while the interruptible customer load may change if the least-cost response to the contingency requires load interruption. However, interruptible customers cannot be interrupted by more than what they have been currently consuming, and not more than the maximum amount they have agreed to. The maximum amount of load interruption for customer $i \in \mathcal{I}$ is defined as $\Delta^{-}_{\max} P_{Li}$.

Since some pre-contingent state variables, such as P^{0}_{Gi} and P^{0}_{Li}, enter into the constraints of the subproblems, these constraints are called the *coupling constraints* of the ESCOPF problem. All other constraints in the ESCOPF problem involve only Y^0 or Y^k. The ESCOPF can then be solved by a decomposition method because the problem can be expressed as a main problem and a set of subproblems related to each other by these coupling constraints.

3.3 Linearized ("DC") ESCOPF Problem

The linearized approximation of a power system, the so-call "DC" approximation, has been applied as a pilot test for the formulation due to its linear and simple nature. The following assumptions are made for the DC approximation of a power system:

- bus voltage magnitudes are approximately 1 pu.,
- differences in bus voltage angles are small,
- transmission lines are purely reactive.

From the above assumptions, the real power losses in the transmission lines, and the reactive power flows are, therefore, zero. The real power flow in MW on a transmission line ij is written as:

$$P_{ij} = -b_{ij}(\theta_i - \theta_j), \tag{3.7}$$

where θ_i is the voltage angle at bus i in degrees and b_{ij} is the per-unit line susceptance. Therefore, the total real power injected at bus i is expressed as:

$$P_i(\theta) = \sum_j P_{ij} = \sum_j -b_{ij}(\theta_i - \theta_j). \tag{3.8}$$

The DC version of the ESCOPF problem is expressed as (3.9)-(3.14).

$$\min_{Y^0} \left\{ [1 - \sum_{k=1}^{K} \pi^k] \left[\sum_{i \in \mathcal{G}} C_{Gi}(P^0_{Gi}) - \sum_{i \in \mathcal{I}} B_{Ii}(P^0_{Li}) - \sum_{i \in \mathcal{U}} B_{Ui}(P^0_{Li}) \right] + \sum_{k=1}^{K} \pi^k S^k(P^0_{Gi}, P^0_{Li}) \right\} \tag{3.9}$$

subject to

$$P^0_i(\theta^0) \;=\; P^0_{Gi} - P^0_{Li} \qquad : \forall\, i, \tag{3.10}$$

$$
\begin{aligned}
P^0_{Gi_{\min}} &\leq P^0_{Gi} \leq P^0_{Gi_{\max}} && : \forall\, i \in \mathcal{G}, \\
-P^0_{ij_{\max}} &\leq P^0_{ij} \leq P^0_{ij_{\max}} && : \forall\, ij, \\
0 &\leq P^0_{Li} && : \forall\, i \in \{\mathcal{I} \cup \mathcal{U}\},
\end{aligned}
\tag{3.11}
$$

where

$$
\begin{aligned}
S^k(P^0_{Gi}, P^0_{Li}) = \min_{Y^k} \Bigg\{ & \sum_{i \in \mathcal{G}} \left[C_{Gi}(P^k_{Gi}) + C_{Ri}(\left| P^k_{Gi} - P^0_{Gi} \right|) \right] \\
& + \sum_{i \in \mathcal{I}} \left[C_{Ii}(P^0_{Li} - P^k_{Li}) - B_{Ii}(P^k_{Li}) \right] - \sum_{i \in \mathcal{U}} B_{Ui}(P^0_{Li}) \Bigg\}
\end{aligned}
\tag{3.12}
$$

subject to

$$
P^k_i(\theta^k) = P^k_{Gi} - P^k_{Li} \qquad : \forall\, i, \tag{3.13}
$$

$$
\begin{aligned}
\max\left(P^k_{Gi_{\min}}, P^0_{Gi} - \Delta^-_{\max} P_{Gi}\right) \leq P^k_{Gi} \leq \min\left(P^k_{Gi_{\max}}, P^0_{Gi} + \Delta^+_{\max} P_{Gi}\right) && : \forall\, i \in \mathcal{G}, \\
-P^k_{ij_{\max}} \leq P^k_{ij} \leq P^k_{ij_{\max}} && : \forall\, ij, \\
\max\left(0, P^0_{Li} - \Delta^-_{\max} P_{Li}\right) \leq P^k_{Li} \leq P^0_{Li} && : \forall\, i \in \mathcal{I}.
\end{aligned}
\tag{3.14}
$$

The DC problem is similar to the problem in "AC" power system, except there are less variables and constraints, and the constraints are all linear as expressed in (3.7) and (3.8).

CHAPTER 4

SOLUTION METHODOLOGY

4.1 The Decomposition ("Decoupled") Method

The decomposition (or *decoupled*) method can be applied to the ESCOPF problem because the problem can be decoupled into a main and several subproblems. In order to simplify our discussion of the decoupled ESCOPF solution methodology, the model in Chapter 3 can be written in a much more compact notation as:

$$\begin{aligned} \min_{Y^0} \quad & \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k S^k(Y^0) \\ \text{subject to} \quad & \pi^0 h^0(Y^0) = 0, \\ & \pi^0 f^0(Y^0) \leq 0, \end{aligned} \tag{4.1}$$

where

$$\begin{aligned} S^k(Y^0) \quad = \quad & \min_{Y^k} \quad C^k(Y^k, Y^0) \\ \text{subject to} \quad & h^k(Y^k) = 0, \\ & f^k(Y^k) \leq 0, \\ & d^k(Y^k, Y^0) \leq 0, \end{aligned} \tag{4.2}$$

where π^0 is the probability of system operating at normal operating state, $C^0(Y^0)$ represents the total costs of generation in the main problem, which includes customer benefits, and $C^k(Y^k, Y^0)$ is the objective function of subproblem k. The equality constraints $h^0(Y^0)$ and $h^k(Y^k)$ denote the power flow equations in the main problem and subproblems, respectively. The set of inequality constraints in the main ESCOPF problem is now denoted by $f^0(Y^0)$, while $f^k(Y^k)$ is the corresponding set of inequality constraints in the subproblem, which do not involve the pre-contingent state Y^0. The set of coupling constraints for contingency k is denoted by $d^k(Y^k, Y^0)$, and appears only in the contingency subproblem.

Adding the probability π^0 to the equality and inequality constraints of the main problem in (4.1) does not change the problem mathematically. However, it changes the units and meanings of the multipliers of those constraints to be the same as the multipliers of the standard OPF problem. In the standard OPF, the Lagrange multiplier of the power flow equation represents the *bus price* in \$/MWh, which is equal to the *marginal cost* of the generator at that bus if none of the generator limits is binding. In the ESCOPF problem, the pre-contingency state only happens with the probability of π^0. By adding π^0 to the power flow equations of the ESCOPF main problem, the multipliers are the bus prices within that pre-contingency state.

On the other hand, the subproblem in (4.2) has the same structure as a standard OPF problem: the objective function of the subproblem does not take into account the contingency probability. Thus, the multipliers of the subproblem already have the same interpretations as those of the OPF problem.

The first step of the problem solution methodology is to decouple our problem

into a main problem and several contingent subproblems. Actually, we have already done this when we presented our model in Chapter 3. To see what the problem would look like without the decoupling step, we can use the simplified notation introduced above to write our overall problem as:

$$\begin{aligned} \min_{Y^0, Y^1, \ldots, Y^K} \quad & \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k C^k(Y^k, Y^0) \\ \text{subject to} \quad & \\ & \pi^k h^k(Y^k) = 0; \qquad k = 0, \ldots, K, \\ & \pi^k f^k(Y^k) \leq 0; \qquad k = 0, \ldots, K, \\ & \pi^k d^k(Y^k, Y^0) \leq 0; \qquad k = 1, \ldots, K. \end{aligned} \tag{4.3}$$

Note that the contingency probabilities π^k, where $k = 1, \ldots, K$, are multiplied by the subproblem constraints when the ESCOPF problem is written as an integrated problem in (4.3) because the objective function of the subproblem $C^k(Y^k, Y^0)$ is now taking into account the contingency probability π^k.

Although the problem, when stated in this way, looks simpler than the decoupled formulation in (4.1) and (4.2), it is actually considerably harder to solve, primarily because of the size of the problem. Problem (4.3) has $K + 1$ sets of variables, Y^0 and Y^1 through Y^K, and each set of variables is itself rather large. There are a large number of constraints within each vector Y^k, which we denote as $h^k(Y^k)$ and $f^k(Y^k)$, as well as a much smaller number of coupling constraints $d^k(Y^k, Y^0)$, which link the variables Y^0 with the variables Y^k. In fact, the coupling constraints only link a few of the variables in Y^0 with a few of the variables in Y^k. Furthermore, the objective function in problem (4.3) is separable, in that it consists of a sum of

costs which each depend only on Y^0 or one of the Y^k. Thus, our problem is, by its very nature, almost completely decoupled to start with. We merely take advantage of this structure and make it explicit when we write the overall problem as the main problem in (4.1) and the K subproblems in (4.2). Each of the $K+1$ individual OPF problems (the main problem and K subproblems) are $1/(K+1)$ as large as the overall problem, thus resulting in considerable speed improvements.

The price we pay for the more manageable decoupled formulation is that we must coordinate the results of each subproblem with that of the main problem. Each time a subproblem is solved for a particular value of Y^0, the value of $S^k(Y^0)$ must be passed back to the main problem. In addition, the main problem must know not only the value of the security cost $S^k(Y^0)$, but how it depends on Y^0 so that the security costs of various contingencies can be traded off against each other and against the pre-contingent cost of operation. In other words, we also need the derivative dS^k/dY^0.

The Lagrange function of the main problem can be written as:

$$\mathcal{L}(Y^0, \lambda^0, \mu^0) = \pi^0 \left[C^0(Y^0) + \lambda^{0^T} h^0(Y^0) + \mu^{0^T} f^0(Y^0) \right] + \sum_{k=1}^{K} \pi^k S^k(Y^0),$$

and the first-order conditions for optimality are

$$\begin{aligned}
0 = \frac{\partial \mathcal{L}}{\partial Y^0}(Y^0, \lambda^0, \mu^0) &= \pi^0 \left[\frac{\partial C^0}{\partial Y^0}(Y^0) + \lambda^{0^T} \frac{\partial h^0}{\partial Y^0}(Y^0) + \mu^{0^T} \frac{\partial f^0}{\partial Y^0}(Y^0) \right] + \sum_{k=1}^{K} \pi^k \frac{\partial S^k}{\partial Y^0}(Y^0), \\
0 = \frac{\partial \mathcal{L}}{\partial \lambda^0}(Y^0, \lambda^0, \mu^0) &= \pi^0 h^0(Y^0), \\
0 = \frac{\partial \mathcal{L}}{\partial \mu^0}(Y^0, \lambda^0, \mu^0) &= \pi^0 f^0(Y^0).
\end{aligned}$$

In order to satisfy the optimality conditions above, we must know the gradient $\partial S^k/\partial Y^0$ at Y^0 for each subproblem k. But this gradient is already available from

the optimal values of the Lagrange multipliers of the coupling constraints which we obtain each time we solve the subproblems at the value Y^0. To see this, we can apply the *envelope theorem* [39] to each subproblem in order to obtain the gradient. The Lagrange function of a subproblem is written as follows:

$$\mathcal{L}^k(Y^k, Y^0, \lambda^k, \mu^k, \mu_d^k) = C^k(Y^k, Y^0) + \lambda^{k^T} h^k(Y^k) + \mu^{k^T} f^k(Y^k) + \mu_d^{k^T} d^k(Y^k, Y^0).$$

At the solution, the optimal decision variables and multipliers can be expressed as $Y^{k^*}(Y^0)$, $\lambda^{k^*}(Y^0)$, $\mu^{k^*}(Y^0)$, and $\mu_d^{k^*}(Y^0)$, and thus the solution of subproblem k is

$$S^k(Y^0) = \mathcal{L}^k\left(Y^{k^*}(Y^0), \lambda^{k^*}(Y^0), \mu^{k^*}(Y^0), \mu_d^{k^*}(Y^0), Y^0\right).$$

Assuming that $S^k(Y^0)$ and $Y^{k^*}(Y^0)$ are differentiable, the derivative of $S^k(Y^0)$ with respect to Y^0 can be written as:

$$\begin{aligned}
\frac{dS^k}{dY^0}(Y^0) &= \frac{d}{dY^0}\mathcal{L}^k\left(Y^{k^*}(Y^0), \lambda^{k^*}(Y^0), \mu^{k^*}(Y^0), \mu_d^{k^*}(Y^0), Y^0\right) \\
&= \left[\frac{\partial \mathcal{L}^k}{\partial Y^k} \cdot \frac{dY^{k^*}}{dY^0}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \lambda^k} \cdot \frac{d\lambda^{k^*}}{dY^0}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \mu^k} \cdot \frac{d\mu^{k^*}}{dY^0}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \mu_d^k} \cdot \frac{d\mu_d^{k^*}}{dY^0}\right] + \frac{\partial \mathcal{L}^k}{\partial Y^0}.
\end{aligned}$$

At the optimal solution, the first-order conditions for optimality are satisfied. That is

$$\frac{\partial \mathcal{L}^k}{\partial Y^k} = \frac{\partial \mathcal{L}^k}{\partial \lambda^k} = \frac{\partial \mathcal{L}^k}{\partial \mu^k} = \frac{\partial \mathcal{L}^k}{\partial \mu_d^k} = 0.$$

Thus the gradient of the security cost for contingency k is

$$\begin{aligned}
\frac{dS^k}{dY^0}(Y^0) &= \frac{\partial \mathcal{L}^k}{\partial Y^0}(Y^{k^*}(Y^0), \lambda^{k^*}(Y^0), \mu^{k^*}(Y^0), \mu_d^{k^*}(Y^0), Y^0) \\
&= \frac{\partial C^k}{\partial Y^0}(Y^{k^*}(Y^0), Y^0) + \mu_d^{k^{*T}}(Y^0) \cdot \frac{\partial d^k}{\partial Y^0}(Y^{k^*}(Y^0), Y^0).
\end{aligned}$$

The envelope theorem simply says that $d\mathcal{L}/dY^0 = \partial\mathcal{L}/\partial Y^0$ at the optimum. The derivative $\partial C^k/\partial Y^0$ is either the marginal cost of ramping or the marginal cost of

interruption because the terms involving Y^0 in the subproblem objective function are ramping costs and interruption costs. Furthermore, because the parameter Y^0 enters linearly into the coupling constraints, the derivatives $\partial d^k / \partial Y^0$ are constants, usually $+1$ or -1.

Given Y^0, which is the pre-contingent operating point from the main problem, security costs $S^k(Y^0)$ and gradients of these security costs can be computed, and passed to the main problem so that the main problem can obtain an improved estimate of the optimal Y^0. However, this improved estimate of Y^0 is based on subproblems which were solved using the old value of Y^0: once the new Y^0 becomes available, the security costs for each contingency must be re-evaluated based on the new pre-contingent operating point Y^0 to see if the security costs or their gradients have changed. The subproblem solutions at the latest estimate of the optimal Y^0 is then used in the main problem to improve Y^0 still further. This continues until all optimality conditions in both main and sub-problems are satisfied simultaneously.

Several methods, such as the *Newton* approach [24] and the *primal-dual interior point* (PDIP) method [38], can be used to solve these optimization problems. The Newton method is an *active set method*, and requires that we correctly identify the set of active constraints in order to obtain the optimal solution. The set of active constraints may change iteratively until we get close to the actual solution. In practice, we have found that "cycling" behavior often occurs, in which a small number of constraints repeatedly enter and leave the active set, without resulting in convergence. We have developed several methods in order to efficiently identify the binding constraints and avoid the cycling problem during the iterative process. Although some

of these methods provide solutions in some cases, for other cases the cycling process continues to be a problem. Thus, we explored the use of the primal-dual interior-point method, which did not suffer from this particular problem. The algorithm worked very well with the decoupled 3-bus case, which is the smallest case in this research. The results of the 3-bus case are illustrated and discussed in Chapter 5.

4.1.1 Decoupled Formulation Using Primal-Dual Interior-Point (PDIP) Method

The barrier function of the main problem stated in (4.1), can be expressed as:

$$\mathcal{B}^0(Y^0,\lambda^0,\nu^0) = \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k S^k(Y^0) + \lambda^{0^T}\pi^0 h^0(Y^0) - \nu^0\pi^0 \left[\ln|f^0(Y^0)|\right], \quad (4.4)$$

where ν^0 is the barrier parameter of the main problem. The gradients of the main-problem barrier function with respect to variable Y^0 and multiplier λ^0 are:

$$\begin{aligned}\nabla_{Y^0}\mathcal{B}^0(Y^0,\lambda^0,\nu^0) &= \pi^0\frac{\partial C^0(Y^0)}{\partial Y^0} + \sum_{k=1}^{K}\pi^k\frac{\partial S^k(Y^0)}{\partial Y^0} + \lambda^{0^T}\pi^0\frac{\partial h^0(Y^0)}{\partial Y^0} \\ &\quad -\nu^0\pi^0\left[\frac{1}{f^0(Y^0)}\frac{\partial f^0(Y^0)}{\partial Y^0}\right], \quad (4.5)\\ \nabla_{\lambda^0}\mathcal{B}^0(Y^0,\lambda^0,\nu^0) &= \pi^0 h^0(Y^0). \quad (4.6)\end{aligned}$$

The slack variable and multiplier for inequality constraints of the main problem are defined as follows:

$$\begin{aligned}s^0 &= -f^0(Y^0), \quad (4.7)\\ \mu^0 &= -\frac{\nu^0}{f^0(Y^0)} = \frac{\nu^0}{s^0}. \quad (4.8)\end{aligned}$$

By substituting the multiplier μ^0 into the gradient in (4.5), and rewriting the conditions in (4.7) and (4.8), we obtain the first-order optimality conditions for the main

problem as follows:

$$\pi^0 \left[\frac{\partial C^0(Y^0)}{\partial Y^0} + \lambda^{0^T} \frac{\partial h^0(Y^0)}{\partial Y^0} + \mu^{0^T} \frac{\partial f^0(Y^0)}{\partial Y^0} \right] + \sum_{k=1}^{K} \pi^k \frac{\partial S^k(Y^0)}{\partial Y^0} = 0, \quad (4.9)$$

$$h^0(Y^0) = 0, \quad (4.10)$$

$$f^0(Y^0) + s^0 = 0, \quad (4.11)$$

$$\mathbf{M}^0 \mathbf{S}^0 \mathbf{e}^0 - \nu^0 = 0, \quad (4.12)$$

and

$$\mu^0 \geq 0, \quad (4.13)$$

$$s^0 \geq 0, \quad (4.14)$$

where

$\mathbf{S}^0$: a diagonal matrix constructed from s^0

$\mathbf{M}^0$: a diagonal matrix constructed from μ^0

$\mathbf{e}^0$: a column vector $[1, 1, \ldots, 1]^T$ that has the same size as s^0 and μ^0

The probability π^0 can be omitted from the equality constraint in (4.10) for simplification because it does not change the mathematical solution of the problem.

Following the Newton-Raphson algorithm discussed in Section 2.3.2 for solving the above first-order optimality conditions, we can write the update equation for the main problem as:

$$\underbrace{\begin{bmatrix} H^0 & J^{0^{\mathrm{T}}} & A^{0^{\mathrm{T}}} & 0 \\ J^0 & 0 & 0 & 0 \\ A^0 & 0 & 0 & \mathbf{I} \\ 0 & 0 & \mathbf{S}^0 & \mathbf{M}^0 \end{bmatrix}}_{W_B^0} \underbrace{\begin{bmatrix} \Delta Y^0 \\ \Delta\lambda^0 \\ \Delta\mu^0 \\ \Delta s^0 \end{bmatrix}}_{\Delta z^0} = - \underbrace{\begin{bmatrix} \frac{\partial \mathcal{B}^0}{\partial Y^0} \\ h^0(Y^0) \\ f^0(Y^0) + s^0 \\ \mu^0\, s^0 - \nu^0 \end{bmatrix}}_{g_B^0}, \tag{4.15}$$

where

$$\begin{aligned} H^0 &= \tfrac{\partial^2 \mathcal{B}^0(z)}{\partial (Y^0)^2}, \\ J^0 &= \tfrac{\partial h^0(Y^0)}{\partial Y^0}, \\ A^0 &= \tfrac{\partial f^0(Y^0)}{\partial Y^0}, \end{aligned}$$

and

$\mathbf{I}$: an identity matrix

The update step Δz^0 is then computed from (4.15), and the step length α is chosen based on the minimum ratio test to make sure that μ^0 and s^0 are positive. Then, the updated vector z^0 for the next iteration of the main problem can be obtained. The process of finding an updated vector z^0 continues until the first-order optimality conditions are satisfied for that particular ν^0. Then, we decrease the parameter ν^0 and obtain a new set of the optimality conditions. We continue finding the variables and multipliers that satisfy the optimality constraints for the reduced ν^0 until ν^0 is zero or close enough. The set of variables and multipliers that satisfy the optimality constraints is the optimal solution of the main problem.

Note that the matrix W_B^0 of the PDIP method in (4.15) is not symmetric, while the Hessian matrix (W) of the Newton method is symmetric as discussed in

Section 2.3.1. The matrix W in the Newton method is symmetric because it is the matrix of the second-order derivatives. However, the matrix $W_{\mathcal{B}}^0$ has a symmetric part that looks like W matrix in (2.52) as shown in the dashed box in the left-hand corner. The diagonal matrices $\mathbf{I}$, $\mathbf{S}^0$, and $\mathbf{M}^0$ outside the dashed box correspond to the slack variables s^0 and the barrier parameter ν^0.

Similarly for the subproblem, the barrier function of the subproblem stated in (4.2) is written as:

$$\mathcal{B}^k(Y^K, \lambda^k, \nu^k) = C^k(Y^k, Y^0) + \lambda^{k^T} h^k(Y^k) - \nu^k \left[\ln|f^k(Y^k)| + \ln|d^k(Y^k, Y^0)|\right] \quad (4.16)$$

The gradients of the barrier function of the subproblem with respect to variables Y^k and multipliers λ^k are:

$$\begin{aligned} \nabla_{Y^k}\mathcal{B}^k(Y^k, \lambda^k, \nu^k) &= \frac{\partial C^k(Y^k, Y^0)}{\partial Y^k} + \lambda^{k^T}\frac{\partial h^k(Y^k)}{\partial Y^k} - \nu^k\left[\frac{1}{f^k(Y^k)}\frac{\partial f^k(Y^k)}{\partial Y^k}\right] \\ &\quad -\nu^k\left[\frac{1}{d^k(Y^k, Y^0)}\frac{\partial d^k(Y^k, Y^0)}{\partial Y^k}\right], \quad (4.17) \\ \nabla_{\lambda^k}\mathcal{B}^k(Y^k, \lambda^k, \nu^k) &= h^k(Y^k). \quad (4.18) \end{aligned}$$

Introducing the slack variables and multipliers for inequality constraints of the subproblem as

$$\begin{aligned} s^k &= -f^k(Y^k), \quad (4.19) \\ s_d^k &= -d^k(Y^k, Y^0), \quad (4.20) \\ \mu^k &= -\frac{\nu^k}{f^k(Y^k)} = \frac{\nu^k}{s^k}, \quad (4.21) \\ \mu_d^k &= -\frac{\nu^k}{d^k(Y^k, Y^0)} = \frac{\nu^k}{s_d^k}, \quad (4.22) \end{aligned}$$

the first-order optimality conditions of the subproblem are as follows:

$$\frac{\partial C^k(Y^k, Y^0)}{\partial Y^k} + \lambda^{k^T}\frac{\partial h^k(Y^k)}{\partial Y^k} + \mu^{k^T}\frac{\partial f^k(Y^k)}{\partial Y^k} + \mu_d^{k^T}\frac{\partial d^k(Y^k, Y^0)}{\partial Y^k} = 0, \quad (4.23)$$

$$h^k(Y^k) = 0, \quad (4.24)$$

$$f^k(Y^k) + s^k = 0, \quad (4.25)$$

$$d^k(Y^k, Y^0) + s_d^k = 0, \quad (4.26)$$

$$\mathbf{M}^k \mathbf{S}^k \mathbf{e}^k - \nu^k = 0, \quad (4.27)$$

$$\mathbf{M}_d^k \mathbf{S}_d^k \mathbf{e}_d^k - \nu^k = 0, \quad (4.28)$$

and

$$\mu^k,\, \mu_d^k \geq 0, \quad (4.29)$$

$$s^k,\, s_d^k \geq 0, \quad (4.30)$$

where

$\mathbf{S}^k$: a diagonal matrix constructed from s^k

$\mathbf{M}^k$: a diagonal matrix constructed from μ^k

$\mathbf{e}^k$: a column vector $[1, 1, \ldots, 1]^T$ that has the same size as s^k and μ^k

$\mathbf{S}_d^k$: a diagonal matrix constructed from s_d^k

$\mathbf{M}_d^k$: a diagonal matrix constructed from μ_d^k

$\mathbf{e}_d^k$: a column vector $[1, 1, \ldots, 1]^T$ that has the same size as s_d^k and μ_d^k

The update equation of the subproblem can then be expressed as:

$$\underbrace{\begin{bmatrix} H^k & J^{k^{\mathrm{T}}} & A^{k^{\mathrm{T}}} & 0 & A_d^{k^{\mathrm{T}}} & 0 \\ J^k & 0 & 0 & 0 & 0 & 0 \\ A^k & 0 & 0 & \mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{S}^k & \mathbf{M}^k & 0 & 0 \\ A_d^k & 0 & 0 & 0 & 0 & \mathbf{I} \\ 0 & 0 & 0 & 0 & \mathbf{S}_d^k & \mathbf{M}_d^k \end{bmatrix}}_{W_{\mathcal{B}}^k} \underbrace{\begin{bmatrix} \Delta Y^k \\ \Delta\lambda^k \\ \Delta\mu^k \\ \Delta s^k \\ \Delta\mu_d^k \\ \Delta s_d^k \end{bmatrix}}_{\Delta z^k} = - \underbrace{\begin{bmatrix} \frac{\partial \mathcal{B}^k}{\partial Y^k} \\ h^k(Y^k) \\ f^k(Y^k) + s^k \\ \mu^k\, s^k - \nu^k \\ d^k(Y^k, Y^0) + s_d^k \\ \mu_d^k\, s_d^k - \nu^k \end{bmatrix}}_{g_{\mathcal{B}}^k}, \qquad (4.31)$$

and

$$\begin{aligned} H^k &= \tfrac{\partial^2 \mathcal{B}^k(z)}{\partial (Y^k)^2}, \\ J^k &= \tfrac{\partial h^k(Y^k)}{\partial Y^k}, \\ A^k &= \tfrac{\partial f^k(Y^k)}{\partial Y^k}, \\ A_d^k &= \tfrac{\partial d^k(Y^k, Y^0)}{\partial Y^k}. \end{aligned}$$

The block in the dashed box has the same structure as the matrix $W_{\mathcal{B}}^0$ of the main problem. The last two rows and two columns of the matrix $W_{\mathcal{B}}^k$ correspond to the coupling constraints $d^k(Y^k, Y^0)$ that appear only in contingency subproblems. The pre-contingent parameters Y^0 in $d^k(Y^k, Y^0)$ come from the main problem.

The solution of the subproblem can be obtained in a way similar to that described for the main problem. In the decoupled method, each subproblem is solved separately until the parameter ν^k is driven toward zero before its optimal multipliers for the coupling constraints $\mu_d^{k^*}$ are passed to the main problem to compute an updated pre-contingency parameter Y^0. In the main problem, we start the problem with

a relatively large value of ν^0 and perform the iterative process until ν^0 is reduced to zero. The updated optimal decision variables Y^{0^*} are passed back to the subproblems again. This continues until the optimal decision variables in the main problem stop changing. Then we reach the optimal solution of the decoupled ESCOPF problem.

4.1.2 Infeasibility of Subproblems

It is possible that a post-contingency subproblem does not have a feasible solution for a given pre-contingency state. In order to obtain the solution for an infeasible subproblem, we add *penalty variables* (ρ) to the post-contingency inequality constraints. Furthermore, the fairly high *penalty costs* (C_ρ) associated with the penalty variables are added to the objective function of the subproblem to be minimized. The penalty costs are set to be much higher than the ramping cost and the cost of interruption. Otherwise the subproblem may converge to a solution which exceeds the limits instead of rescheduling the generator output or interrupting customer load. The post-contingency subproblem with penalty functions is expressed as follows.

$$\begin{aligned} S^k(Y^0) \;=\; \min_{Y^k,\rho} \quad & C^k(Y^k, Y^0) + C_\rho^T \cdot \rho + C_{\rho_d}^T \cdot \rho_d \\ \text{subject to} \quad & \\ & h^k(Y^k) = 0, \\ & f^k(Y^k) \le \rho, \\ & d^k(Y^k, Y^0) \le \rho_d, \\ & \rho, \rho_d \ge 0. \end{aligned} \tag{4.32}$$

An infeasible post-contingency subproblem will yield very high security cost

due to the nonzero penalty functions. Then the main problem will adjust the pre-contingency operating point so that the infeasibility is removed.

4.1.3 Difficulties of the Decoupled PDIP Method

When the decoupled PDIP algorithm is applied to a case study bigger than the 3-bus case, we have problems with convergence as follows. We start solving the subproblems of the decoupled ESCOPF with the decision variables Y^0 from the system base-case OPF, which is the optimal dispatch of the system without taking into account security. Then the desired Lagrange multipliers $\mu_d^{k^*}$ from all subproblems are gathered and returned to the main problem. The modified main problem is then solved for the improved solution Y^0 in the way that the effects of the coupling constraints multipliers $\mu_d^{k^*}$ are minimized in the post-contingency subproblems. When this updated Y^0 is used to solve the subproblems again, the subproblems yield different answers from the first time. Sometimes, the answers from the subproblems indicate that this pre-contingent state will not cause any trouble when contingencies occur. Thus the Lagrange multipliers from the subproblems (second round) are all zero. When we return these $\mu_d^{k^*} = 0$ to the main problem, we are back to the beginning of the problem, or the base-case OPF, where there is no effect of the subproblems. Then the results from this main problem will yield nonzero $\mu_d^{k^*}$'s again. The problem keeps cycling like this without converging for the case studies we tried which are bigger than the DC 3-bus case.

The cycling behavior in the decoupled PDIP results from identifying the active

set $\mu_d^{k^*}$ of each subproblem by driving its barrier parameter ν^k toward zero. The interior-point algorithm is based on not choosing which constraints are binding and which are not. It treats all the inequality constraints as equality constraints by adding the slack variables. By avoiding explicit identification of the set of active constraints, some of the convergence problems associated with active-set methods can be avoided. The decoupled algorithm takes away this advantage of the interior-point algorithm if the barrier parameters ν^k of the subproblems are allowed to go to zero (thus, in effect, identifying the active set of constraints) before the optimal solution of the whole problem is reached.

However, we believe that the decoupled method could be successfully used if the iteration process is adjusted so that the main and subproblems converge together. In the current decoupled method, the main and all subproblems are completely solved before the exchange of the information between the main and the subproblems takes place, resulting in the pre-selection of the active set $\mu_d^{k^*}$. Therefore, we should try performing only a few iterations on the main problem with a given ν^0 (without letting ν^0 go to zero) before passing the most recent Y^0 to the subproblems. Then, with the same value of the barrier parameters as the main problem, $\nu^k = \nu^0$, several iterations of subproblems are performed before sending the information back to the main problem. The complicated schedules for tuning $\nu^0 \to 0$ and $\nu^k \to 0$ therefore include the decision of how many iterations to perform before the information exchange takes place. The barrier parameters ν^0 and ν^k should not be decreased until the problems are within the central path as explained in Section 2.3.2. In this way, the barrier parameters of all problems are driven toward zero simultaneously. As a result, the

main and all subproblems can gradually converge together.

Up to this point, we have not applied the modified decoupled PDIP discussed above to the ESCOPF problem. We were more concerned that it would not work in general for our proposed ESCOPF model or that too many "tuning" would need to be done. We would like to isolate the difficulties of the decoupled formulation to see if the PDIP would work by itself. Therefore, we applied the PDIP method to our integrated ESCOPF model stated in (4.3). We discovered that the PDIP method worked for the 5-bus and the IEEE 14-bus cases. The results of these two cases are illustrated and discussed in Chapter 5. The successful integrated solution method is described as follows.

4.2 The Integrated Solution Method

The integrated solution method solves the main and subproblems of the ESCOPF model simultaneously. Therefore, the ESCOPF model must be formulated as one integrated problem stated in (4.3). Problem (4.3) is rewritten to explicitly express the main and subproblems as follows:

$$\begin{aligned} &\min_{Y^0,Y^k} \quad \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k C^k(Y^k, Y^0) \\ &\text{subject to} \\ \text{Main} \begin{cases} \pi^0 h^0(Y^0) &= 0, \\ \pi^0 f^0(Y^0) &\leq 0, \end{cases} \end{aligned} \tag{4.33}$$

$$\left.\begin{array}{rcll} \pi^k h^k(Y^k) & = & 0; & k=1,...,K, \\ \pi^k f^k(Y^k) & \leq & 0; & k=1,...,K, \\ \pi^k d^k(Y^k,Y^0) & \leq & 0; & k=1,...,K. \end{array}\right\} \text{Subproblem}$$

By applying the PDIP method, the barrier function can be written as:

$$\mathcal{B}(Y^0,Y^1,...,Y^K,\lambda^0,\lambda^1,...,\lambda^K,\nu) = \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k C^k(Y^k,Y^0) + \sum_{k=0}^{K} \lambda^{k^T} \pi^k h^k(Y^k)$$
$$-\nu\,\pi^k \sum_{k=0}^{K} \left[\ln|f^k(Y^k)|\right] - \nu\,\pi^k \sum_{k=1}^{K} \left[\ln|d^k(Y^k,Y^0)|\right] \quad (4.34)$$

Note that now the constraints of the main and subproblem are no longer expressed separately. The superscript $k = 0$ denotes the main problem, while $k = 1,\ldots,K$ still denotes subproblem 1 through K. However, there is only one barrier parameter ν in this integrated problem instead of ν^0 and ν^k for the main and subproblems as expressed in the decoupled method.

The gradients of the barrier function with respect to variables Y^0 and Y^k, and multipliers λ^0 and λ^k, where $k = 1,\ldots,K$, are:

$$\begin{aligned} \nabla_{Y^0}\mathcal{B} &= \pi^0 \frac{\partial C^0(Y^0)}{\partial Y^0} + \lambda^{0^T} \pi^0 \frac{\partial h^0(Y^0)}{\partial Y^0} - \nu\,\pi^0 \left[\frac{1}{f^0(Y^0)} \frac{\partial f^0(Y^0)}{\partial Y^0}\right] \\ &- \nu\,\pi^0 \left[\frac{1}{d^k(Y^k,Y^0)} \frac{\partial d^k(Y^k,Y^0)}{\partial Y^0}\right], \quad (4.35) \\ \nabla_{Y^k}\mathcal{B} &= \pi^k \frac{\partial C^k(Y^k,Y^0)}{\partial Y^k} + \lambda^{k^T} \pi^k \frac{\partial h^k(Y^k)}{\partial Y^k} - \nu\,\pi^k \left[\frac{1}{f^k(Y^k)} \frac{\partial f^k(Y^k)}{\partial Y^k}\right] \\ &- \nu\,\pi^k \left[\frac{1}{d^k(Y^k,Y^0)} \frac{\partial d^k(Y^k,Y^0)}{\partial Y^k}\right], \quad (4.36) \\ \nabla_{\lambda^0}\mathcal{B} &= \pi^0 h^0(Y^0), \quad (4.37) \\ \nabla_{\lambda^k}\mathcal{B} &= \pi^k h^k(Y^k). \quad (4.38) \end{aligned}$$

The slack variables and multipliers for inequality constraints are defined as follows:

$$s^k = -f^k(Y^k); \qquad k = 0, \ldots, K, \tag{4.39}$$

$$s_d^k = -d^k(Y^k, Y^0); \qquad k = 1, \ldots, K, \tag{4.40}$$

and

$$\mu^k = -\frac{\nu}{f^k(Y^k)} = \frac{\nu}{s^k}; \qquad k = 0, \ldots, K, \tag{4.41}$$

$$\mu_d^k = -\frac{\nu}{d^k(Y^k, Y^0)} = \frac{\nu}{s_d^k}; \qquad k = 1, \ldots, K. \tag{4.42}$$

By substituting multipliers μ^k and μ_d^k into the gradients in (4.35) and (4.36), and rewriting conditions (4.39)-(4.42), the first-order conditions for optimality of the integrated problem are:

$$\pi^0 \left[\frac{\partial C^0(Y^0)}{\partial Y^0} + \lambda^{0^T} \frac{\partial h^0(Y^0)}{\partial Y^0} + \mu^{0^T} \frac{\partial f^0(Y^0)}{\partial Y^0} + \mu_d^{k^T} \frac{\partial d^k(Y^k, Y^0)}{\partial Y^0} \right] = 0 \tag{4.43}$$

$$\pi^k \left[\frac{\partial C^k(Y^k, Y^0)}{\partial Y^k} + \lambda^{k^T} \frac{\partial h^k(Y^k)}{\partial Y^k} + \mu^{k^T} \frac{\partial f^k(Y^k)}{\partial Y^k} + \mu_d^{k^T} \frac{\partial d^k(Y^k, Y^0)}{\partial Y^k} \right] = 0 \tag{4.44}$$

and

$$h^k(Y^k) = 0; \qquad k = 0, \ldots, K, \tag{4.45}$$

$$f^k(Y^k) + s^k = 0; \qquad k = 0, \ldots, K, \tag{4.46}$$

$$d^k(Y^k, Y^0) + s_d^k = 0; \qquad k = 1, \ldots, K, \tag{4.47}$$

$$\mathbf{M}^k \mathbf{S}^k \mathbf{e}^k - \nu = 0; \qquad k = 0, \ldots, K, \tag{4.48}$$

$$\mathbf{M}_d^k \mathbf{S}_d^k \mathbf{e}_d^k - \nu = 0; \qquad k = 1, \ldots, K, \tag{4.49}$$

and

$$\mu^0, \mu^k, \mu_d^k \geq 0; \qquad k = 1, \ldots, K, \tag{4.50}$$

$$s^0, s^k, s_d^k \geq 0; \qquad k = 1, \ldots, K. \tag{4.51}$$

In order to solve the optimality conditions in (4.43)-(4.49), we need to compute the gradients of these equations with respect to all variables and multipliers. There are only few terms in those optimality conditions that have coupling variables: the coupling constraints in (4.43), (4.44) and (4.47), and the objective function of subproblem in (4.44). Therefore, the integrated problem is almost completely decoupled. The gradient of the optimality conditions can be expressed in a *bordered block-Hessian-like* matrix $W_{\mathcal{B}}$ as follows:

$$\underbrace{\begin{bmatrix} W_{\mathcal{B}}^0 & W_{\mathcal{C}}^{(01)} & \cdots & W_{\mathcal{C}}^{(0K)} \\ W_{\mathcal{C}}^{(10)} & W_{\mathcal{B}}^1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ W_{\mathcal{C}}^{(K0)} & 0 & 0 & W_{\mathcal{B}}^K \end{bmatrix}}_{W_{\mathcal{B}}} \underbrace{\begin{bmatrix} \Delta z^0 \\ \cdots \\ \Delta z^1 \\ \cdots \\ \vdots \\ \cdots \\ \Delta z^K \end{bmatrix}}_{\Delta z} = - \underbrace{\begin{bmatrix} g_{\mathcal{B}}^0 \\ \cdots \\ g_{\mathcal{B}}^1 \\ \cdots \\ \vdots \\ \cdots \\ g_{\mathcal{B}}^K \end{bmatrix}}_{g_{\mathcal{B}}}, \tag{4.52}$$

where

$W_{\mathcal{B}}^0$: Hessian-like matrix of the main problem as expressed in (4.15)

$W_{\mathcal{B}}^k$: Hessian-like matrix of subproblem $k = 1, \ldots, K$ as expressed in (4.31)

$W_{\mathcal{C}}^{(0k)}$: the effects on the main problem due to the changes in subproblem k

$W_{\mathcal{C}}^{(k0)}$: the effects on subproblem k due to the changes in the main problem

The matrices in the diagonal blocks are the Hessian-like matrices of the main and all subproblems. The effects of the coupling constraints appear only in the blocks

on the top and left borders of the matrix $W_{\mathcal{B}}$. These blocks have highly sparsity structures due to the very few coupling constraints in the optimality conditions. The other blocks of $W_{\mathcal{B}}$ are zero because there are no coupling constraints between the subproblems.

The first row of (4.52) can be written as:

$$W_{\mathcal{B}}^{0}\Delta z^{0} = -g_{\mathcal{B}}^{0} - W_{\mathcal{C}}^{(01)}\Delta z^{1} - \ldots - W_{\mathcal{C}}^{(0K)}\Delta z^{K}. \tag{4.53}$$

Therefore, the matrices $W_{\mathcal{C}}^{(01)}, \ldots, W_{\mathcal{C}}^{(0K)}$ can be interpreted as how the changes in the variables of subproblem k affects the changes in the variables of the main problem. On the other hand, if we rewrite any other row of (4.52), we obtain

$$W_{\mathcal{B}}^{k}\Delta z^{k} = -g_{\mathcal{B}}^{k} - W_{\mathcal{C}}^{(k0)}\Delta z^{0}; \qquad k = 1, \ldots, K. \tag{4.54}$$

The matrices $W_{\mathcal{C}}^{(k0)}$ for $k = 1, \ldots, K$ can be interpreted as how the changes in the variables of the main problem affects the changes in the variables of subproblem k.

4.3 Interpretation of Multipliers

A constraint multiplier tells us how much the objective function could be improved if that constraint were relaxed by one unit. Some multipliers in the ESCOPF problem have meaningful economic interpretations in power systems considering system security. The multipliers on the real power balance equations at the pre-contingency state are the *bus prices* of the system. The multipliers on the maximum limits on the real power generation at the post-contingency state can be used to determine the *marginal value of spinning reserve* for that generator. The generator maximum limits include the maximum capacity limit and the maximum ramping-up limit. Likewise, the *marginal value of interruptible load* is determined from the multiplier on the minimum constraint on real power load at the post-contingency state. The studies of these multipliers are described as follows.

4.3.1 Marginal Value of Spinning Reserve

Decoupled Problem Formulation

The economic interpretation of the multiplier on a constraint can be demonstrated using the *envelope theorem* [39, 40]. The *envelope theorem* as applied to our ESCOPF problem can be derived as follows.

For the decoupled formulation, the ESCOPF problem can be rewritten as:

$$\begin{aligned} S^0(Y^0,\delta) \;=\; \min_{Y^0} \quad & \pi^0 C^0(Y^0) + \sum_{k=1}^{K} \pi^k S^k(Y^0,\delta) \\ \text{subject to} \quad & \pi^0 h^0(Y^0) = 0, \end{aligned} \tag{4.55}$$

$$\pi^0 f^0(Y^0) \leq 0,$$

where

$$\begin{aligned} S^k(Y^0, \delta) \quad &= \quad \min_{Y^k} \quad C^k(Y^k, Y^0) \\ \text{subject to} \quad & \qquad h^k(Y^k) = 0, \\ & \qquad f^k(Y^k) \leq 0, \\ & \qquad d^k(Y^k, Y^0, \delta) \leq 0, \end{aligned} \tag{4.56}$$

where δ is a parameter corresponding to relaxing the coupling constraint in each subproblem $k = 1, \ldots, K$.

The Lagrange function of subproblem k can be written as:

$$\mathcal{L}^k(Y^k, Y^0, \lambda^k, \mu^k, \mu_d^k, \delta) = C^k(Y^k, Y^0) + \lambda^{k^T} h^k(Y^k) + \mu^{k^T} f^k(Y^k) + \mu_d^{k^T} d^k(Y^k, Y^0, \delta).$$

The inequality constraints we include in the Lagrange function are the ones in the active set (binding). At the solution, the optimal decision variables and multipliers of the subproblem can be expressed as $Y^{k^*}(\delta)$, $\lambda^{k^*}(\delta)$, $\mu^{k^*}(\delta)$, and $\mu_d^{k^*}(\delta)$, and thus the solution of the subproblem is

$$S^k(Y^0, \delta) = \mathcal{L}^k\left(Y^{k^*}(\delta), \lambda^{k^*}(\delta), \mu^{k^*}(\delta), \mu_d^{k^*}(\delta), \delta\right).$$

The derivative of $S^k(Y^0, \delta)$ with respect to δ can be written as:

$$\frac{dS^k}{d\delta}(Y^0, \delta) = \left[\frac{\partial \mathcal{L}^k}{\partial Y^k} \cdot \frac{dY^{k^*}}{d\delta}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \lambda^k} \cdot \frac{d\lambda^{k^*}}{d\delta}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \mu^k} \cdot \frac{d\mu^{k^*}}{d\delta}\right] + \left[\frac{\partial \mathcal{L}^k}{\partial \mu_d^k} \cdot \frac{d\mu_d^{k^*}}{d\delta}\right] + \frac{\partial \mathcal{L}^k}{\partial \delta}.$$

At the optimal solution, the first-order conditions for optimality are satisfied. That is

$$\frac{\partial \mathcal{L}^k}{\partial Y^k} = \frac{\partial \mathcal{L}^k}{\partial \lambda^k} = \frac{\partial \mathcal{L}^k}{\partial \mu^k} = \frac{\partial \mathcal{L}^k}{\partial \mu_d^k} = 0.$$

Thus the gradient of $S^k(Y^0, \delta)$ becomes

$$\begin{aligned}\frac{dS^k}{d\delta}(Y^0, \delta) &= \frac{\partial \mathcal{L}^k}{\partial \delta}(Y^{k^*}(\delta), \lambda^{k^*}(\delta), \mu^{k^*}(\delta), \mu_d^{k^*}(\delta), \delta) \\ &= \mu_d^{k^{*T}}(\delta) \cdot \frac{\partial d^k}{\partial \delta}(Y^{k^*}(\delta), Y^0, \delta).\end{aligned}$$

The derivative $\partial d^k / \partial \delta$ is either $+1$ or -1 because δ enters linearly into the coupling constraint. Therefore, the *envelope theorem* simply shows that the change of the subproblem objective function due to relaxing the coupling constraint by δ is equal to $\pm \mu_d^{k^*}$, where $\mu_d^{k^*}$ is the Lagrange multiplier on a coupling constraint. If the sign in front of the multiplier is negative, the objective function is improved.

The improvement of the main ESCOPF objective function due to relaxing the coupling constraint by δ can be found by applying another *envelope theorem* to the main problem in (4.55). We write down the Lagrange function of the main problem as:

$$\mathcal{L}^0(Y^0, \lambda^0, \mu^0, \delta) = \pi^0 \left[C^0(Y^0) + \lambda^{0^T} h^0(Y^0) + \mu^{0^T} f^0(Y^0) \right] + \sum_{k=1}^{K} \pi^k S^k(Y^0, \delta).$$

By applying the *envelope theorem* again, the gradient of the main objective function with respect to the parameter δ becomes

$$\begin{aligned}\frac{dS^0}{d\delta}(Y^0, \delta) &= \frac{\partial \mathcal{L}^0}{\partial \delta}(Y^{0^*}(\delta), \lambda^{0^*}(\delta), \mu^{0^*}(\delta), \delta) \\ &= \sum_{k=1}^{K} \pi^k \frac{\partial S^k}{\partial \delta}(Y^0, \delta). \qquad (4.57)\end{aligned}$$

Thus the second application of the *envelope theorem* says that when we relax the coupling constraint by δ, the improvement of the objective function of the ESCOPF problem is equal to the summation over all contingencies of the improvements of the subproblem objective functions multiplied by their contingency probabilities.

Since we have verified the interpretation of the multipliers on coupling constraints, the marginal value of spinning reserve can be derived as follows. For the decoupled ESCOPF, the binding ramping-up constraint on generator i at post-contingency k can be expressed as:

$$\left[P_{Gi}^{k} - \left(P_{Gi}^{0} + \Delta_{\max}^{+} P_{Gi}\right)\right] = 0; \qquad \text{where} \quad \mu_{\Delta P_{Gi}}^{k} > 0, \tag{4.58}$$

where $\mu_{\Delta P_{Gi}}^{k}$ is the Lagrange multiplier of the ramping-up constraint on generator i. An additional δ_i^k of ramping-up capacity at contingency k relaxes the ramping-up constraint by δ_i^k. As a result, the objective function of subproblem k will be improved by

$$\mu_{\Delta P_{Gi}}^{k} \delta_i^k.$$

Therefore, by applying equation (4.57) to the ramping-up constraint, if the ramping-up capacity on generator i is relaxed by δ_i in all contingencies $k = 1, \ldots, K$, the objective function of the ESCOPF problem would be improved by

$$\left[\sum_{k=1}^{K} \pi^k \mu_{\Delta P_{Gi}}^{k}\right] \delta_i.$$

Similarly, the binding constraint on maximum capacity of generator i at contingency k, with the multiplier $\mu_{P_{Gi}}^{k}$, can be expressed as:

$$\left[P_{Gi}^{k} - P_{Gi_{\max}}^{k}\right] = 0; \qquad \text{where} \quad \mu_{P_{Gi}}^{k} > 0. \tag{4.59}$$

The multiplier $\mu_{P_{Gi}}^{k}$ has an economic interpretation of how much the objective function of subproblem k will be improved if the maximum capacity constraint on generator i is relaxed by one unit. As discussed earlier, an additional δ_i of generator capacity

in every contingency will improve the subproblem objective function by $\mu^k_{P_{Gi}}\delta_i$, and therefore improve the total objective function of the ESCOPF problem by

$$\left[\sum_{k=1}^{K} \pi^k \, \mu^k_{P_{Gi}}\right] \delta_i.$$

The marginal value of spinning reserve of generator i can be interpreted as how much the objective function is improved when an addition unit of either ramping-up capacity or generator maximum capacity is given to the generator. Thus the marginal value of spinning reserve of generator i in the decoupled problem formulation can be expressed as:

$$\sum_{k=1}^{K} \pi^k \left(\mu^k_{\Delta P_{Gi}} + \mu^k_{P_{Gi}}\right).$$

Integrated Problem Formulation

For the integrated problem formulation, suppose in contingency k, which has the contingency probability π^k, the ramping-up constraint on generator i is binding, that is:

$$\pi^k \left[P^k_{Gi} - \left(P^0_{Gi} + \Delta^+_{\max} P_{Gi}\right)\right] = 0; \qquad \text{where} \quad \mu^k_{\Delta P_{Gi}} > 0. \tag{4.60}$$

The positive value $\mu^k_{\Delta P_{Gi}}$ is the multiplier corresponding to the ramping-up constraint. It can be interpreted as how much the objective function would be improved by having an additional unit of ramping-up capacity. Note that, from now on, we will always use the interpretation of the Lagrange multipliers derived from the *envelope theorem.*

If we are allowed access to an additional δ^k_i of ramping-up capacity at contingency k, we relax the ramping-up constraint by $\epsilon^k_i = \pi^k \, \delta^k_i$. Therefore, by the *envelope*

theorem, the objective function[1] will be improved by

$$\epsilon_i^k \, \mu_{\Delta P_{Gi}}^k = \pi^k \, \mu_{\Delta P_{Gi}}^k \, \delta_i^k. \tag{4.61}$$

If an additional δ_i of ramping-up capacity is allowed for generator i in all contingencies $k = 1, \ldots, K$, the objective function of the ESCOPF problem can be improved by

$$\left[\sum_{k=1}^{K} \pi^k \, \mu_{\Delta P_{Gi}}^k \right] \delta_i.$$

The improvement of the objective function of the integrated ESCOPF problem due to an additional δ_i of maximum capacity on generator i in all contingencies can be derived in a way similar to that of the constraint on ramping-up limit, and can be expressed as:

$$\left[\sum_{k=1}^{K} \pi^k \mu_{P_{Gi}}^k \right] \delta_i,$$

where $\mu_{P_{Gi}}^k$ is the multiplier on the maximum capacity constraint on generator i at post-contingency k.

Therefore, by the definition of the marginal value of spinning reserve discussed earlier, the marginal value of spinning reserve of generator i in the integrated ESCOPF problem cam be written as:

$$\sum_{k=1}^{K} \pi^k \left(\mu_{\Delta P_{Gi}}^k + \mu_{P_{Gi}}^k \right),$$

which is the same as that of the decoupled problem.

[1] There is only one objective function in the integrated problem formulation, which is the objective function of the ESCOPF problem.

4.3.2 Marginal Value of Interruptible Load

Decoupled Problem Formulation

The marginal value of interruptible load can be interpreted as how much the objective function of the ESCOPF problem can be improved when another unit of interruptible load is allowed. The binding constraint on the interruptible load at bus i at contingency k in the decoupled problem formulation can be expressed as:

$$\left[\left(P_{Li}^{0} - \Delta_{\max}^{-} P_{Li}\right) - P_{Li}^{k}\right] = 0; \qquad \text{where} \quad \mu_{\Delta P_{Li}}^{k} > 0. \tag{4.62}$$

The economic interpretation of the multiplier $\mu_{\Delta P_{Li}}^{k}$ is how much the objective function would be improved if we were allowed to have another unit of interruptible load.

With a similar verification described on the marginal value of spinning reserve in the decoupled formulation, if an additional δ_i of interruptible load is allowed at load bus i in all contingencies $k = 1, \ldots, K$, the objective function of the ESCOPF problem will be improved by

$$\left[\sum_{k=1}^{K} \pi^{k} \mu_{\Delta P_{Li}}^{k}\right] \delta_i.$$

Therefore, the marginal value of interruptible load in the decoupled problem formulation can be expressed as:

$$\sum_{k=1}^{K} \pi^{k} \mu_{\Delta P_{Li}}^{k}.$$

Integrated Problem Formulation

For the integrated formulation, the binding constraint on the interruptible load at bus i for contingency k is written as:

$$\pi^k \left[\left(P_{Li}^0 - \Delta_{\max}^- P_{Li} \right) - P_{Li}^k \right] = 0; \qquad \text{where} \quad \mu_{\Delta P_{Li}}^k > 0. \tag{4.63}$$

If an additional δ_i of interruptible load is allowed at load bus i in all contingencies $k = 1, \ldots, K$, the objective function of the integrated ESCOPF problem will be improved by

$$\left[\sum_{k=1}^{K} \mu_{\Delta P_{Li}}^k \right] \delta_i.$$

As a result, the marginal value of interruptible load in the integrated problem formulation can be expressed as:

$$\sum_{k=1}^{K} \mu_{\Delta P_{Li}}^k.$$

CHAPTER 5

SUMMARY OF RESULTS

5.1 3-bus Case

The model formulated in Chapter 3 was first tested on a contrived 3-bus system shown in Figure 5.1. The linearized approximation of the system, the so-called "DC" approximation, is assumed in this particular case. Thus the ESCOPF problem for this system is set up as described in Section 3.3. The algorithm used to solve this system is the *decomposition* method described in Section 4.1.

The characteristics of the 3-bus system are taken from [41]. This case is a very simple power system with two generators connected at buses 1 and 2, and load at bus 3, where C_{Gi} is the cost characteristic of generator i, B_{Li} is the benefit curve of customer load at bus i, b_{ij} is the susceptance of line ij, and $P_{ij\max}$ is the maximum capacity of line ij.

The generation costs of both generators C_{G1} and C_{G2} are as shown in Figure 5.1, and their additional data are given in Table 5.1, where $P_{Gi_{\min}}$ and $P_{Gi_{\max}}$ are the minimum and maximum capacity limits of generator i, respectively. For the

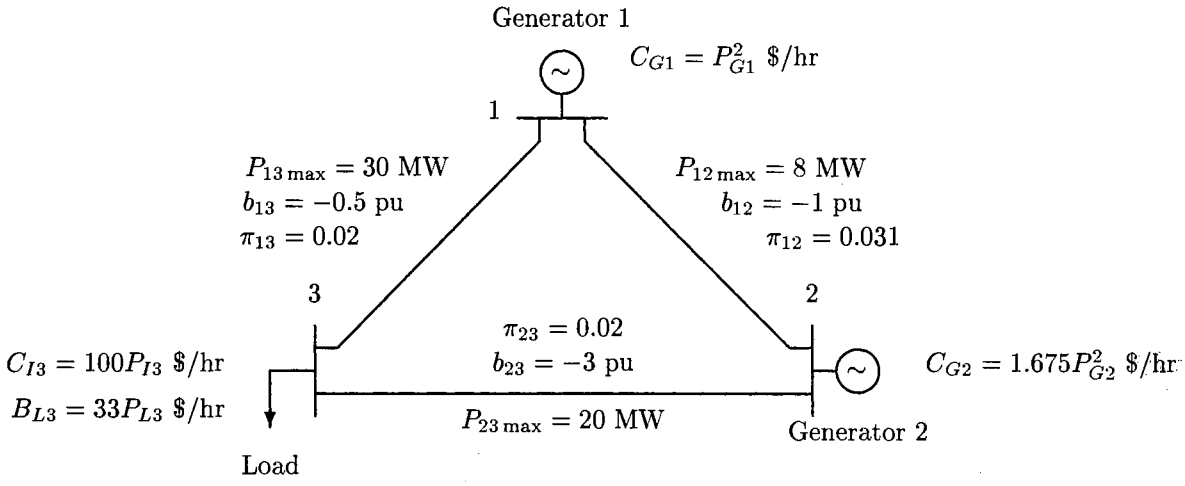

Figure 5.1: Characteristics of DC 3-bus system.

	Gen 1	Gen 2
Marginal Cost	$2P_{G1}$\$/MWh	$3.35P_{G2}$\$/MWh
$P_{Gi_{\min}}$	5 MW	5 MW
$P_{Gi_{\max}}$	35 MW	30 MW
$\Delta^{+}_{\max}P_{Gi}$	5 MW	1.5 MW
$\Delta^{-}_{\max}P_{Gi}$	8 MW	7 MW

Table 5.1: Characteristics of generators in 3-bus system at pre-contingency state.

study of the ESCOPF, some additional generator information such as the maximum rates of ramping generators up and down ($\Delta^{+}_{\max}P_{Gi}$, $\Delta^{-}_{\max}P_{Gi}$), is needed and is given in Table 5.1; additional costs corresponding to generator ramping rates are ignored in this example. Furthermore, the interruption cost of load (C_{Ii}), and the probabilities of line outage contingencies (π_{ij}), which do not appear in [41], are needed and are presented in Figure 5.1.

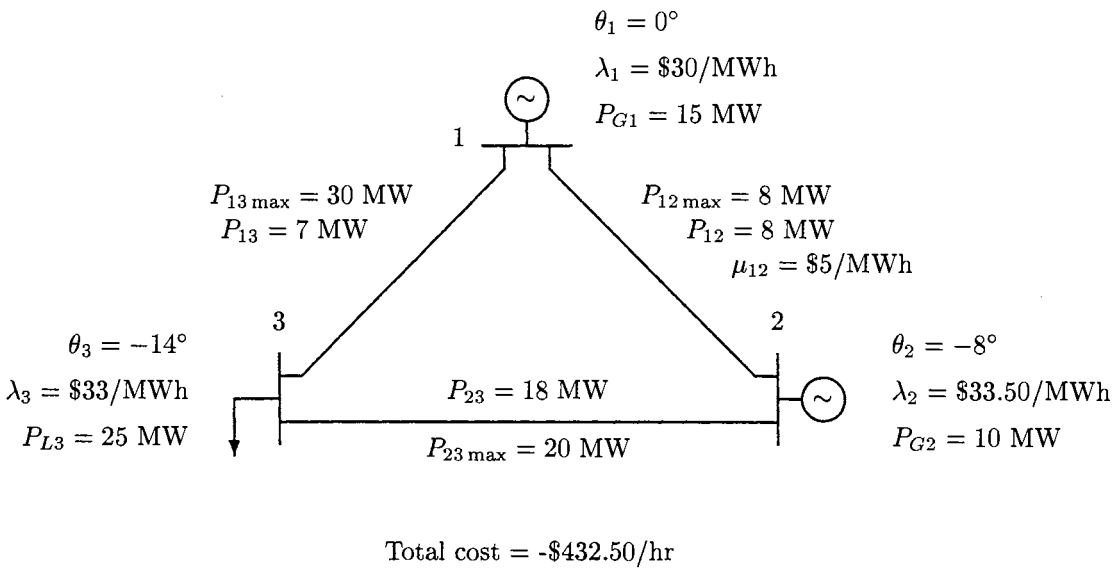

Figure 5.2: Base case OPF of DC 3-bus system, ignoring contingencies.

5.1.1 Standard (Base-Case) OPF of DC 3-bus Case

The initial pre-contingent operating state, which is the optimal power flow of the main problem without contingencies, is illustrated in Figure 5.2: λ_i is the Lagrange multiplier at bus i and is interpreted as the optimal bus price ignoring security considerations [24], and μ_{ij} is the multiplier of the binding real power flow limit on line ij. If the real power flow constraint were not binding, μ_{ij} would be zero. The interpretation of μ_{ij} is how much our objective function can be improved if we are allowed to relax the constraint by 1 MW. The voltage angle θ_1 is always equal to zero because bus 1 is chosen to be the slack bus, and thus serves as the angle reference for all other voltages in the system.

The optimal objective function for this OPF problem is $-\$432.50$/hr. Since the objective function that we are minimizing is the negative expected social welfare

	Gen 1	Gen 2
$P^k_{Gi_{\min}}$	7 MW	5 MW
$P^k_{Gi_{\max}}$	20 MW	11.50 MW

Table 5.2: Post-contingency generation limits given the initial estimate of the optimal pre-contingency state Y^0.

of the power system, the system with negative optimal objective function means that it has a positive value of maximum expected social welfare, and thus the system is considered worth operating.

As discussed in Section 4.1, the gradient $\partial S^k(Y^0)/\partial Y^0$ of every credible contingency k is needed to determine an improved pre-contingent operating point. In this illustrated example, there are three credible subproblems: outages on line 1-2, line 1-3, and line 2-3. All three subproblems are then solved starting with the state shown in Figure 5.2, which represents the optimal pre-contingent state ignoring the effects of contingencies.

As a result of the contingency, constraints on generators may change depending on their pre-contingent operating points and their ramping rates. From the given pre-contingency operating points shown in Figure 5.2 and ramping rates in Table 5.1, the generator limits shown in Table 5.2 are the post-contingency limits given the initial value of the pre-contingent state Y^0. Both generators have new upper limits at post-contingency state because generator 1 can not ramp up more than 5 MW from its initial pre-contingency operating point, which is 15 MW, and generator 2 can not ramp up more than 1.50 MW from its pre-contingency generation of 10 MW, regardless of their nominal post-contingency limits. Similarly, generator 1 cannot

ramp down by more than 8 MW from its pre-contingency generation of 15 MW, so given the initial estimate of the optimal Y^0, the post-contingency lower limit on generation is 7 MW instead of 5 MW. On the other hand, generator 2 can ramp down by 7 MW from its pre-contingent level of 10 MW, so its new post-contingent limit based on ramping is 3 MW. Since this is less than its minimum generation level of 5 MW, the post-contingency minimum level is 5 MW.

Assuming that customer load at bus 3 is an interruptible load, the load should remain 25 MW unless the system interrupts part of it in order to minimize the cost of surviving the contingency.

5.1.2 Subproblem 1 of DC 3-bus Case

Figure 5.3 illustrates the optimal redispatch when an outage of line 1-2 occurs. This solution is optimal with no binding constraints, still assuming a pre-contingency operating point of Y^0. By losing line 1-2 which has a line flow limit of only 8 MW, generator 1, which is the cheaper generator, can generate more power without hitting either its generator constraint, or line flow limits, and thus the total cost of system operation is lower compared to the total cost of the pre-contingency OPF. Security costs and gradients of subproblems are summarized in Table 5.3.

The derivatives of S with respect to P_{G1} and P_{G2} for contingency 1 are zero because generator constraints are not binding at the post-contingency state, and there are no additional costs for ramping generators up and down. Thus, changing the pre-contingent amount of real power generation does not result in improving the sub-

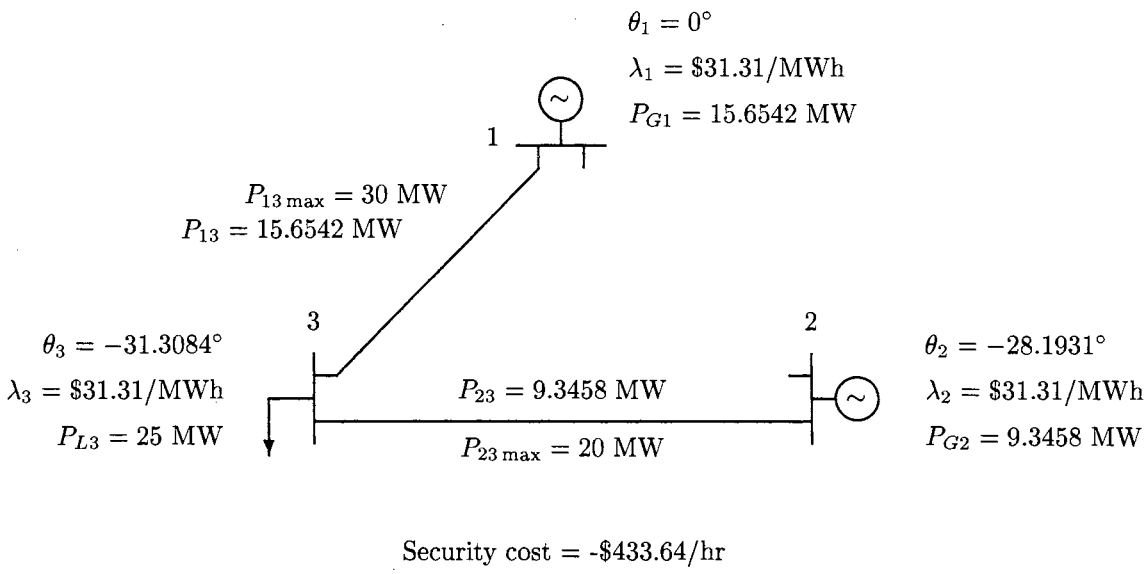

Figure 5.3: Solution of (DC) subproblem 1 given Y^0 from Figure 5.2.

	Subproblem 1	Subproblem 2	Subproblem 3
Security cost	−$433.64/hr	$192.02/hr	−$428.80/hr
dS/dP_{G1}	0	0	0
dS/dP_{G2}	0	−$94.48/MWh	0
dS/dP_{L3}	−$1.69/MWh	$100.00/MWh	$1.00/MWh

Table 5.3: Security costs and gradients of subproblems in Figures 5.3, 5.4, and 5.5, based on initial value of Y^0 from Figure 5.2.

problem. The derivative of S^1 with respect to P_{L3}, which is equal to $-\$1.69$/MWh, means that had the customer consumed one more MW before the contingency occurred, the cost of the subproblem would have been lowered by \$1.69/hr because the optimal Lagrange multiplier at bus 3 (λ_3) is now \$31.31/MWh instead of \$33/MWh, the marginal benefit of consumption at bus 3. So the derivative of S with respect to P_{L3} tell us that the system would like to interrupt a negative amount of load. In other words, the customer at bus 3 should consume more load when contingency 1 occurs.

As a result, the gradients from subproblem 1 tell the main problem to choose an improved pre-contingency state so that more load is consumed, but the gradients not provide any guidance about whether to raise or lower the pre-contingent levels of real power generation. However, this information about how to improve the overall objective function of the SCOPF is based only on information from contingency 1 and may be partially or totally contradicted by information from other subproblems.

5.1.3 Subproblem 2 of DC 3-bus Case

For the initial pre-contingent state Y^0, the solution of subproblem 2 is shown in Figure 5.4. During contingency 2, losing line 1-3 can cause the customer load at bus 3 to be interrupted because the only line connected to the load bus is line 2-3 which has only a 20 MW capacity, while the load was 25 MW at pre-contingency. Generator 2 can generate at most 11.50 MW due to its post-contingent limits. Meanwhile, generator 1 can not distribute more than 8 MW because of the limit on line 1-2.

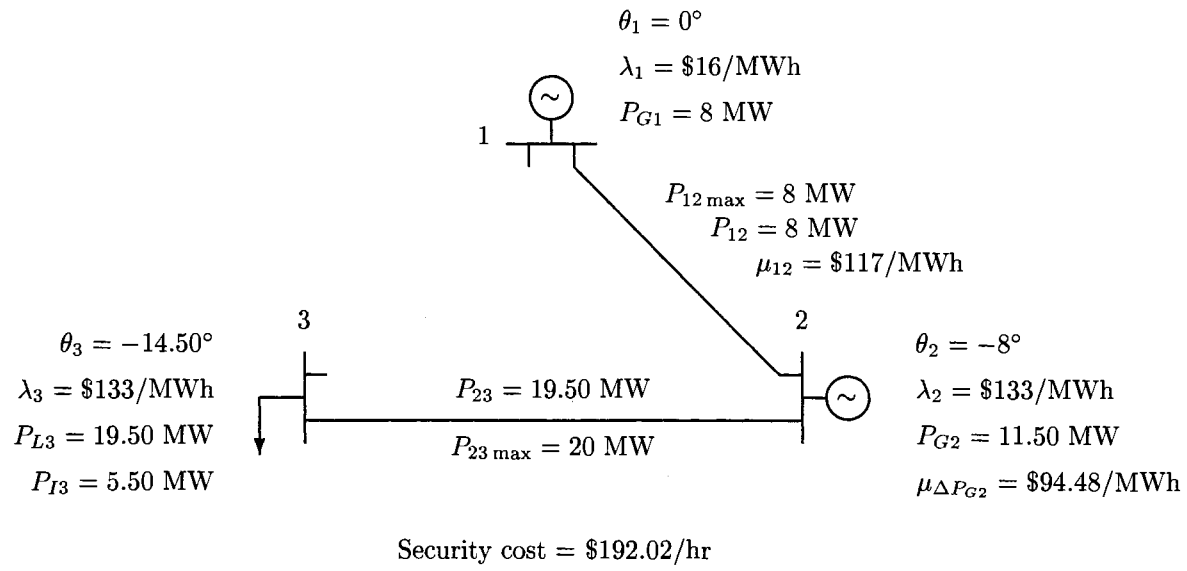

Figure 5.4: Solution of (DC) subproblem 2 given Y^0 from Figure 5.2.

Therefore, even though line 2-3 allows the flow of 20 MW, the total power generated from both generators is only 19.50 MW. This results in 5.50 MW load interruption. Due to the high marginal cost of interruption, which is \$100/MWh, the optimal cost of subproblem 2 becomes positive, which means the social welfare at post-contingency is negative. Had the system not allowed interruption of the customer at bus 3, this subproblem would not have had a feasible solution. The derivative of S with respect to P_{L3} of subproblem 2 in Table 5.3 is \$100/MWh, which has the interpretation that one more MW of load before the contingency would have resulted in one more MW interrupted at this post-contingency and would have cost the system another \$100/hr.

The negative gradient with respect to P_{G2} in subproblem 2 can be interpreted as how much the optimum post-contingency cost would have been reduced by increasing the output of generator 2 by 1 MW before the contingency. Since the limit on generator 2 depends on the pre-contingent real power generated at bus 2, increasing

P_{G2} by 1 MW before the contingency results in relaxing its post-contingency upper limit by 1 MW. This explains why the multiplier of the maximum generation constraint on generator 2, $\mu_{\Delta P_{G2}}$, has the same absolute value as the gradient with respect to P_{G2}, as is discussed in Section 4.1. As a result, generator 2 would have been able to generate more after the contingency and the amount of interrupted load would have been reduced. This is why the derivative dS^2/dP_{G2} is comparable in magnitude to the derivative dS^2/dP_{L3}.

From the point of view of subproblem 2, the derivative with respect to P_{G2} is the Lagrange multiplier on the upper generation limit, which is determined by the availability of spinning reserve. It is therefore the *marginal value of spinning reserve* at generator 2, which can be interpreted as how much another MW of real power generation is worth in post-contingency state 2. In the other subproblems, the spinning reserve on generator 2 is not exhausted, and spinning reserve on generator 1 is never exhausted in any subproblem, so the marginal values are zero in these cases. Note that this does not mean that the spinning reserve is worthless, it means only that there is no value associated with *additional* spinning reserve. In summary, the large values of gradients in subproblem 2 simply say that the main problem should increase P_{G2} and decrease load in order to reduce the security cost of contingency 2.

5.1.4 Subproblem 3 of DC 3-bus Case

For the last subproblem, an outage on line 2-3, the solutions are shown in Figure 5.5, along with corresponding gradients and security cost in Table 5.3. As in subproblem 1,

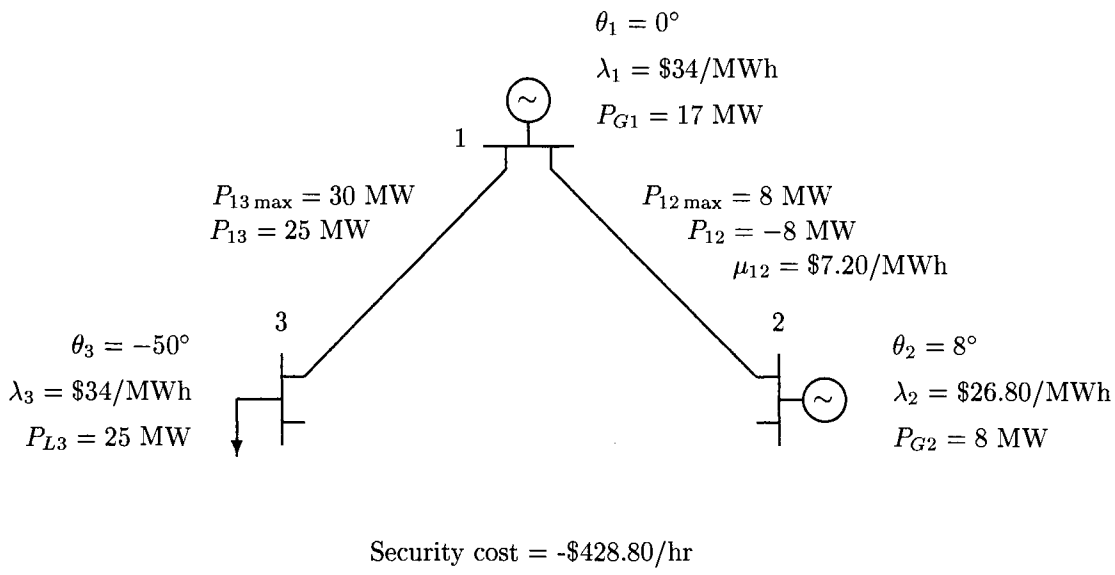

Figure 5.5: Solution of (DC) subproblem 3 given Y^0 from Figure 5.2.

pre-contingent real power generation does not have an impact on subproblem 3 since the gradients with respect to P_{G1} and P_{G2} are zero. That means the constraints on real power generation are not binding, and therefore the marginal values of spinning reserves for subproblem 3 are zero. The gradient with respect to P_{L3} is again the difference between the optimal post-contingent Lagrange multiplier λ_3 and the marginal benefit of consumption after the contingency. Since the optimal multiplier at the load bus of subproblem 3, which is \$34/MWh, is higher than \$33/MWh (the post-contingent marginal benefit), cutting down the pre-contingent consumption would have reduced the security cost of this contingency. By reducing 1 MW of load in the pre-contingent state, the security cost of contingency 3 would be reduced by \$1/hr, which is the gradient dS^3/dP_{L3}

However, in this particular example, we do not run out of interruptible load. Therefore, the *marginal values of interruptible load* are zero in all subproblems. If the

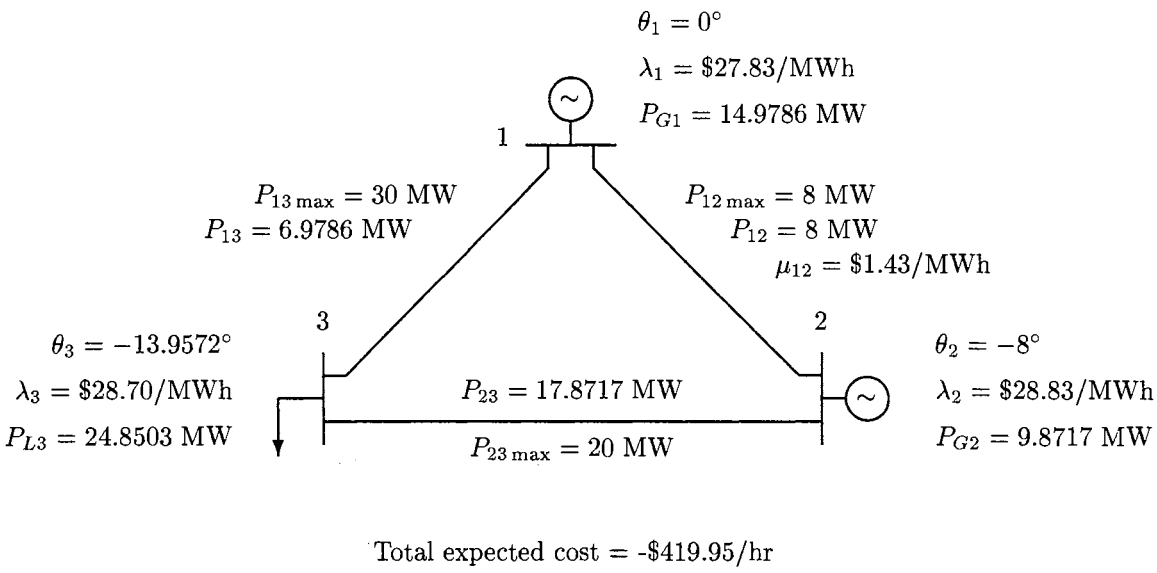

Figure 5.6: Second iteration solution of (DC) main problem incorporating information from subproblems in Figures 5.3, 5.4, and 5.5.

maximum interruptible load constraint were binding in a subproblem, the corresponding Lagrange multiplier would tell us how much an additional 1 MW of interruptible load would be worth, and is thus the *marginal value of interruptible load.* Again, although interruptible load does have value, its marginal value is zero because we have not used all the interruptible load we have, and so *additional* interruptible load has no value.

Since we have already obtained security costs and gradients for all subproblems, we are now ready to compute another main OPF problem that includes all contingencies and their probabilities to obtain a new pre-contingent state. We now recompute the solution of the main problem, including the information on the security costs and their gradients from each of the contingency subproblems we just solved (based on the old estimate of the optimal Y^0 from the previous solution of the

main problem). The results of the second solution of the main problem is shown in Figure 5.6. Load at bus 3 in this main OPF with contingencies is reduced from 25 MW due to the gradient with respect to P_{L3} in subproblem 2. Lagrange multipliers, λ_i, are not the optimal bus prices of the system anymore because the contingency probabilities are included, and the impact of pre-contingency on security costs causes these Lagrange multipliers to deviate from the optimal bus prices of the base case OPF ignoring contingencies. The total expected cost of OPF with contingencies is higher than the total cost of base case OPF ignoring contingencies because it includes security costs from subproblems along with associated contingency probabilities.

The new pre-contingent state Y^0 from the main OPF with contingencies is then used to solve all the subproblems again as described in Section 4. The main problem and the contingency subproblems are then repeated until all the optimality conditions (for both the main problem and subproblems) are satisfied. It is not instructive to show the results of every problem in this sequence of iterations. Therefore, we skip to the final (converged) solution of the main problem in Figure 5.7, along with the final (converged) solutions of the subproblems, shown in Figures 5.8, 5.9, and 5.10. A table of gradients of the security costs for each converged subproblem is presented in Table 5.4.

The solution of converged ESCOPF is not much different from the main OPF with contingencies in Figure 5.6 because there is no big change in the subproblem gradients after the first main problem with contingencies are solved.

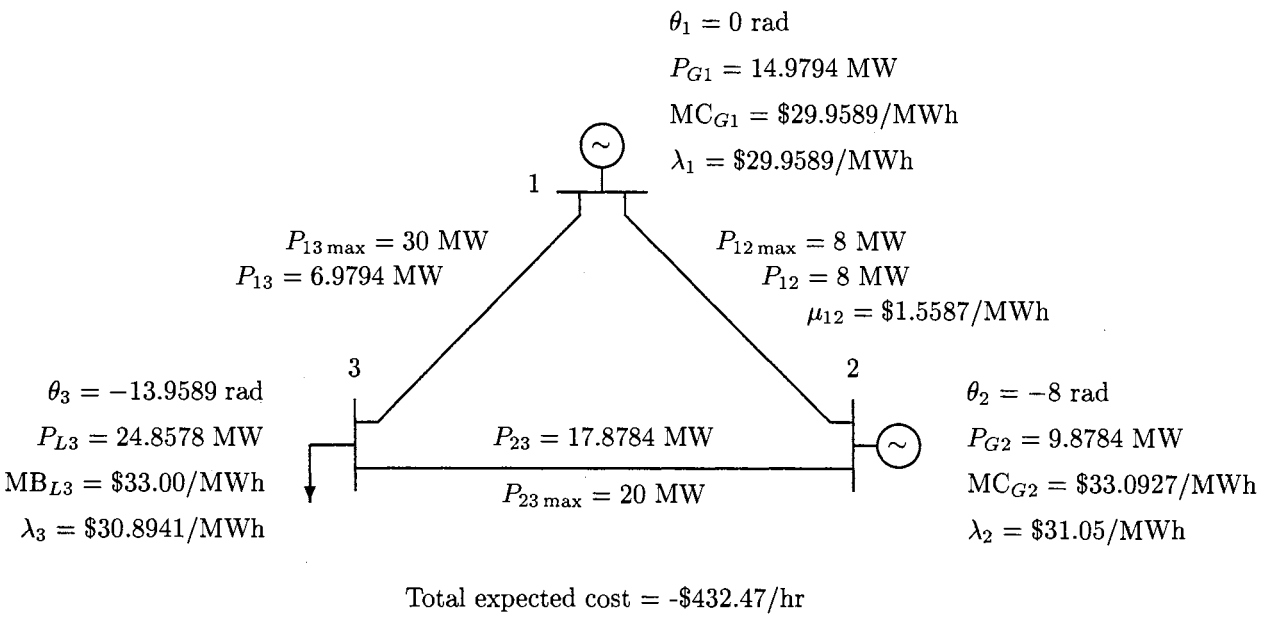

Figure 5.7: Final (converged) solution of (DC) main problem of ESCOPF.

	Subproblem 1	Subproblem 2	Subproblem 3
Security cost	−$433.39/hr	$189.34/hr	−$428.92/hr
dS/dP_{G1}	0	0	0
dS/dP_{G2}	0	−$94.88/MWh	0
dS/dP_{L3}	−$1.87/MWh	$100.00/MWh	$0.72/MWh

Table 5.4: Security costs and gradients of subproblems at final (optimum) solution of ESCOPF.

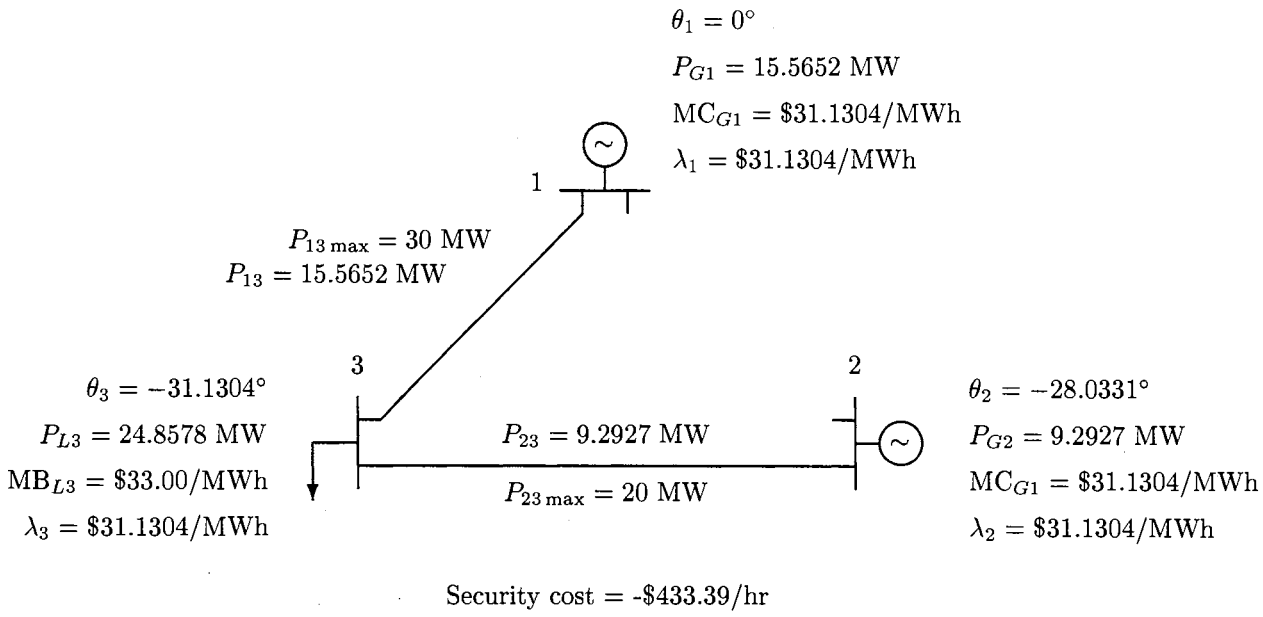

Figure 5.8: Final (converged) solution of (DC) subproblem 1 of ESCOPF.

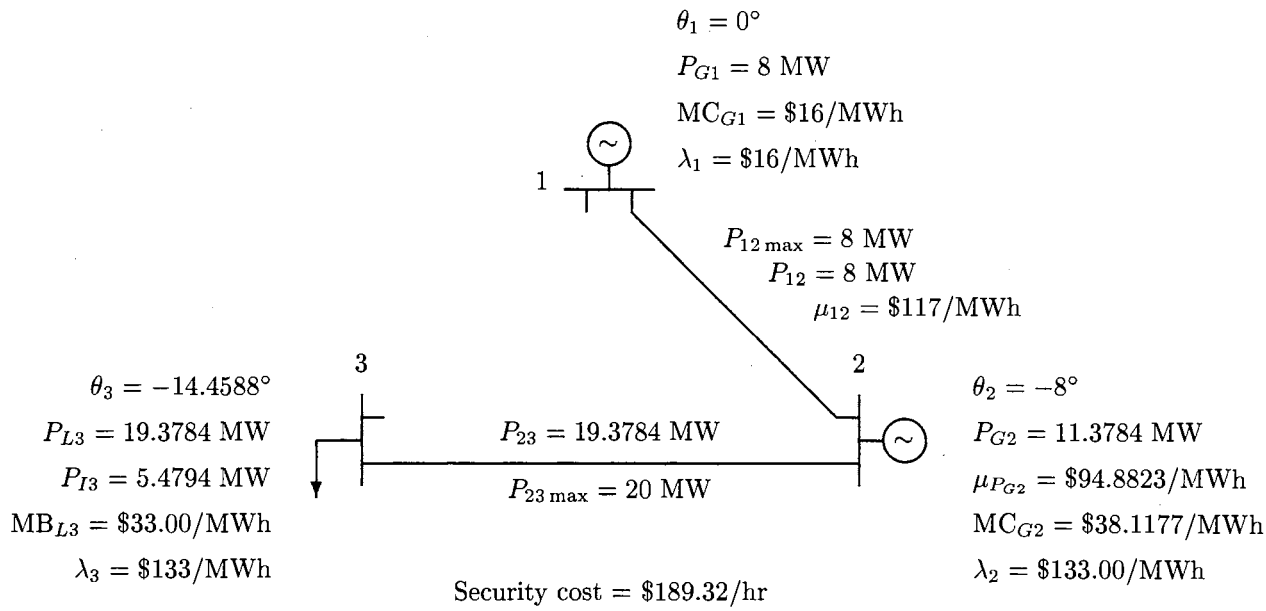

Figure 5.9: Final (converged) solution of (DC) subproblem 2 of ESCOPF.

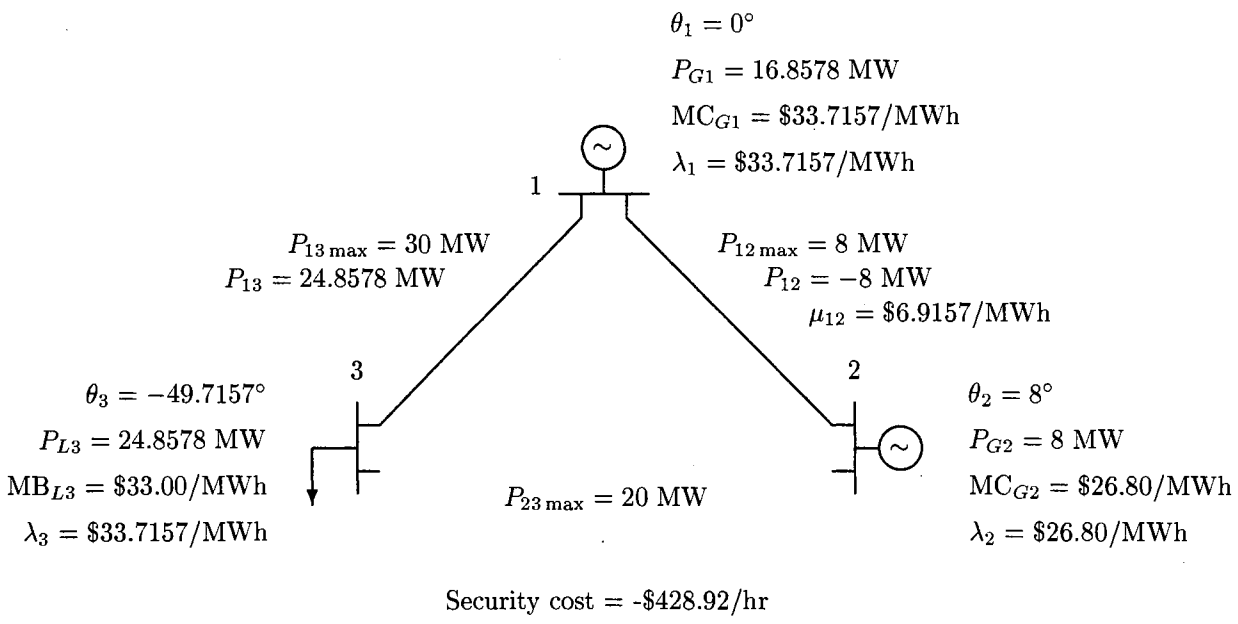

Figure 5.10: Final (converged) solution of (DC) subproblem 3 of ESCOPF.

5.2 5-bus Case

The 5-bus system is modified from the IEEE 14-bus system by removing the low voltage buses (buses 6-14) and the connected transmission lines. The characteristics of the 5-bus system are summarized in Figure 5.11 and Table 5.6. The raw data of the 5-bus case is illustrated in an IEEE-Common Data Format (IEEE-CDF) in Appendix B.

The system consists of two generators connected at buses 1 and 2, a synchronous condenser connected at bus 3, and buses 2, 3, 4, and 5 are load buses. The synchronous condenser supplies only reactive power. The notations of Figure 5.11 are the same as those of the 3-bus case in Figure 5.1. However, for the 5-bus case, the flow in each transmission line is allowed to slightly exceed its capacity limit for a short time period (an emergency time limit) when contingencies occur. The new line flow limit is called an *emergency* limit, and is shown in parentheses next to $P_{ij_{\max}}$ in Figure 5.11.

There are seven credible contingencies for the 5-bus system as shown in Table 5.5, where π^k is the probability of contingency k.

Contingency	Lines	π^k
1	1 - 2	0.01
2	2 - 3	0.01
3	2 - 4	0.01
4	3 - 4	0.01
5	1 - 5	0.01
6	2 - 5	0.01
7	4 - 5	0.01

Table 5.5: Contingencies of 5-bus system and their probabilities.

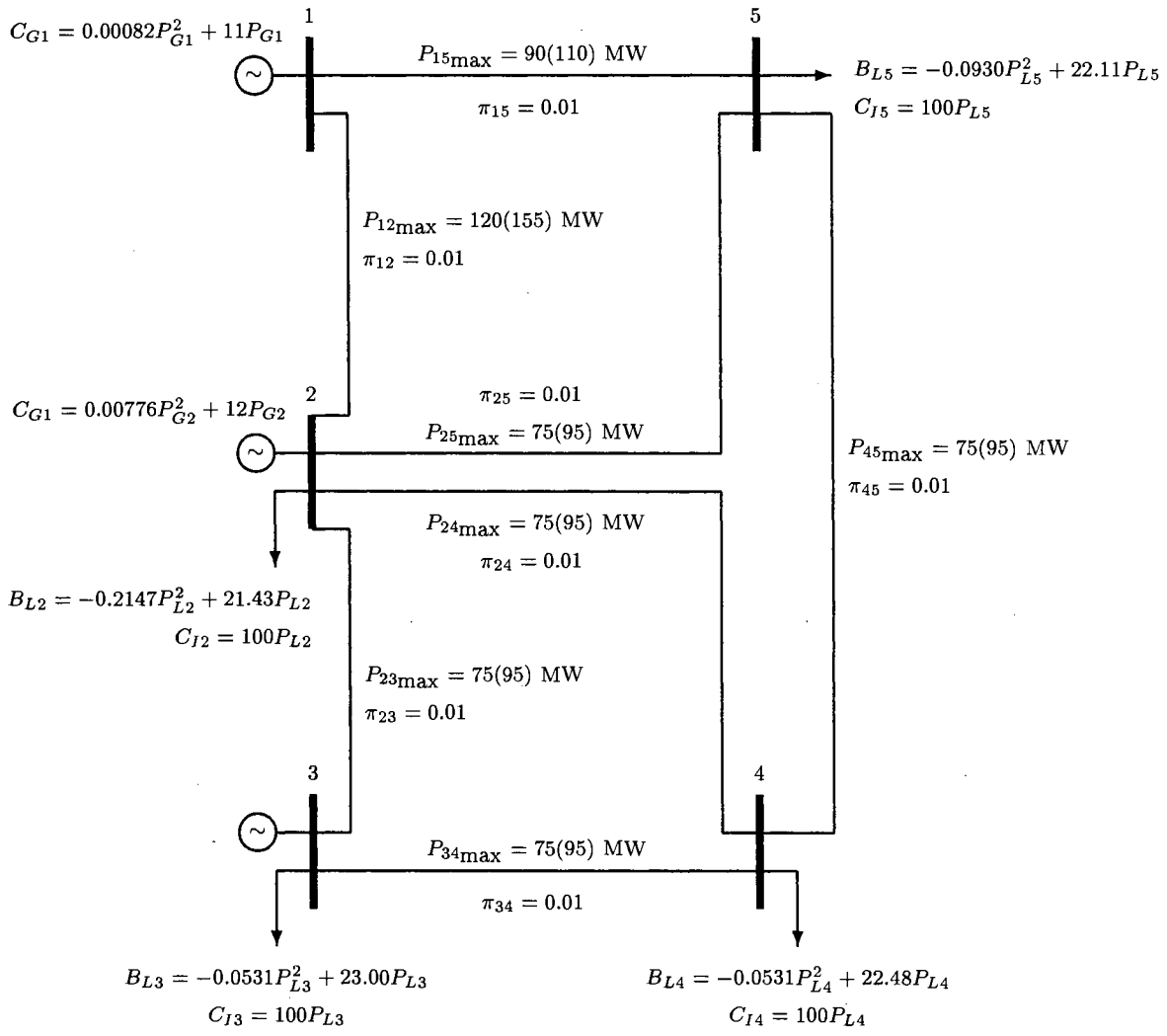

Figure 5.11: Characteristics of 5-bus system.

	$P_{Gi_{\min}}$	$P_{Gi_{\max}}$	$Q_{Gi_{\min}}$	$Q_{Gi_{\max}}$	$\Delta^{+}_{\max}P_{Gi}$	$\Delta^{-}_{\max}P_{Gi}$
Gen 1	45 MW	250 MW	-100 MVar	150 MVar	50 MW	50 MW
Gen 2	15 MW	150 MW	-40 MVar	50 MVar	35 MW	35 MW
SynCon 3	N/A	N/A	-40 MVar	40 MVar	N/A	N/A

Table 5.6: Characteristics of generators in 5-bus system at pre-contingency state.

5.2.1 5-bus Case: DC Approximation

The 5-bus case was solved using the "DC" approximation as expressed in Section 3.3. The algorithm used to solve this case is the *integrated solution method*, discussed in Section 4.2. Since the system is a DC approximation, the power flow equations are linear, and there are no reactive flows and real power losses in the system.

In the integrated solution method, the main problem and all subproblems are solved simultaneously. The final solutions of the 5-bus case are illustrated as the main problem and seven contingency subproblems as shown in Figures 5.12-5.19. The solution of the main problem is the *optimal* pre-contingency state of the ESCOPF problem.

Given the pre-contingency generations in Figure 5.12 and the ramping rates in Table 5.6, the post-contingency limits on generators are shown in Table 5.7. Three of the four post-contingency generation limits differ from their pre-contingency generation limits (the first two columns of Table 5.6) due to ramping constraints. Generator 1 can not ramp down more than 50 MW from its pre-contingency operating point, which is 146.40 MW, and can not ramp up more than 50 MW, either. As a result, the post-contingency minimum and maximum limits on generator 1 are 96.40 MW and 196.40 MW, respectively, regardless of their capacity limits. Meanwhile generator 2

	$P_{Gi_{\min}}$	$P_{Gi_{\max}}$
Gen 1	96.40 MW	196.40 MW
Gen 2	80 MW	150 MW

Table 5.7: Post-contingency generation limits given the initial estimate of the optimal pre-contingency state Y^0.

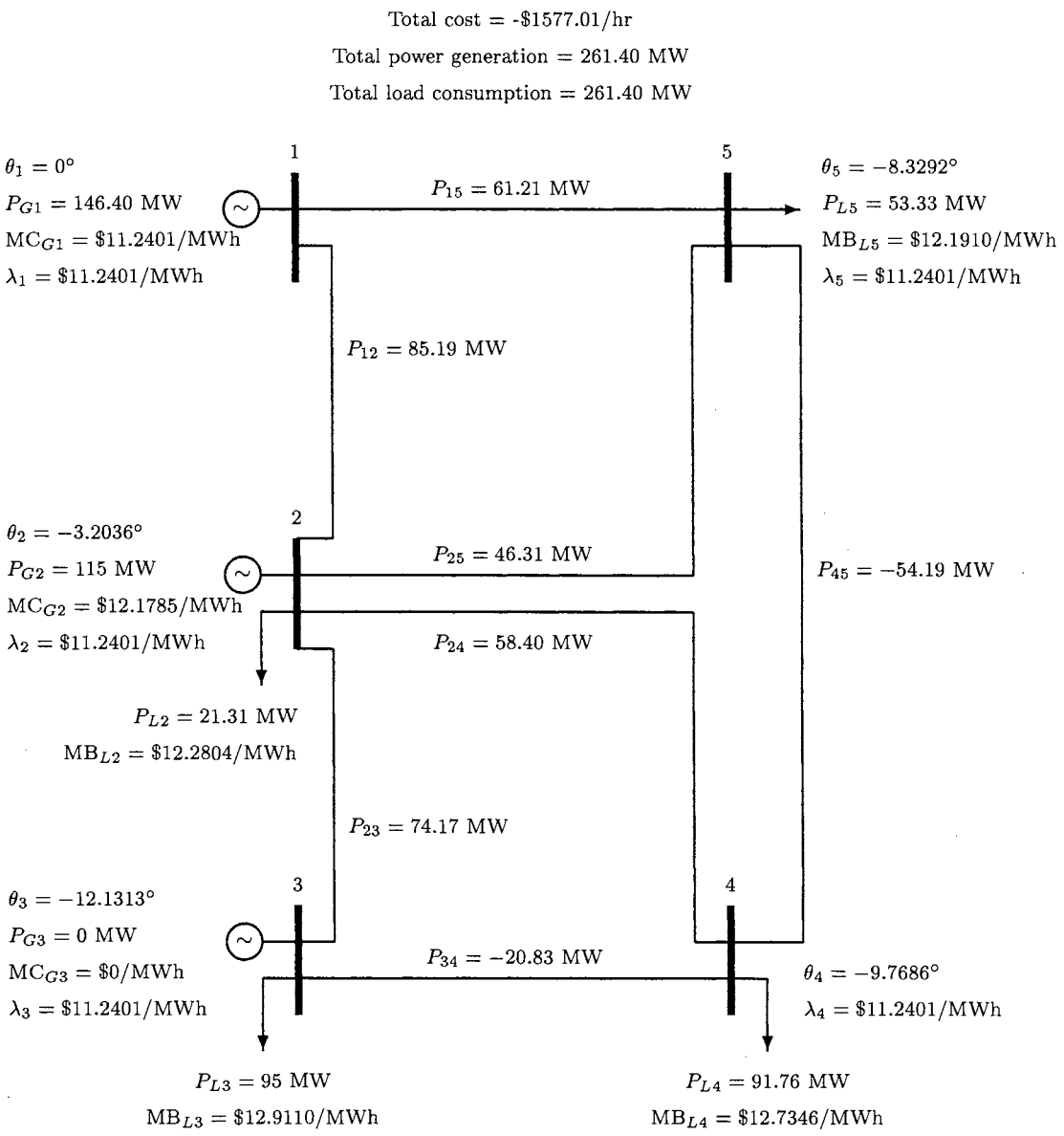

Figure 5.12: Optimal pre-contingency state of (DC) ESCOPF (final solution of main problem).

can ramp up and down for 35 MW from its pre-contingency operating point, which is 115 MW. Therefore, the post-contingency minimum limit for generator 2 is obviously 80 MW, resulting from its ramping down limit, since it is higher than its minimum capacity, which is 15 MW. However, ramping up 35 MW from its pre-contingency operating point, which is 115 MW, generator 2 also hits its maximum capacity limit. As a result, the post-contingency maximum limit on generator 2 is the same as its pre-contingency maximum limit.

Subproblem 1

Subproblem 1 is the most critical contingency because it is the outage of the biggest line, line 1-2, which is connected to the biggest generator in the system. The optimal solution of subproblem 1 is as shown in Figure 5.13. Generator 1 has to reduce the output from the pre-contingency state to $P_{G1}^{1} = 110$ MW because the only line that is connected to bus 1, line 1-5, has the emergency maximum flow limit of 110 MW. Thus, generator 2 needs to cover the rest of the load in the system. The post-contingency maximum limit of generator 2 is 150 MW resulting from its maximum ramping limit as well as its actual capacity limit. Although generator 2 generates at its post-contingency maximum limit, $P_{G2}^{1} = 150$ MW, the total post-contingency generation of the system is only 260 MW, which is a little less than the total pre-contingency load of 261.40 MW. Therefore, we can not avoid the load interruption when the contingency on line 1-2 occurs, given the pre-contingency state in Figure 5.12. The load at buses 2 and 5 is interrupted at the amount of 0.28 MW and 1.12 MW, respectively. The load at the other load buses stay the same as in the pre-contingency state.

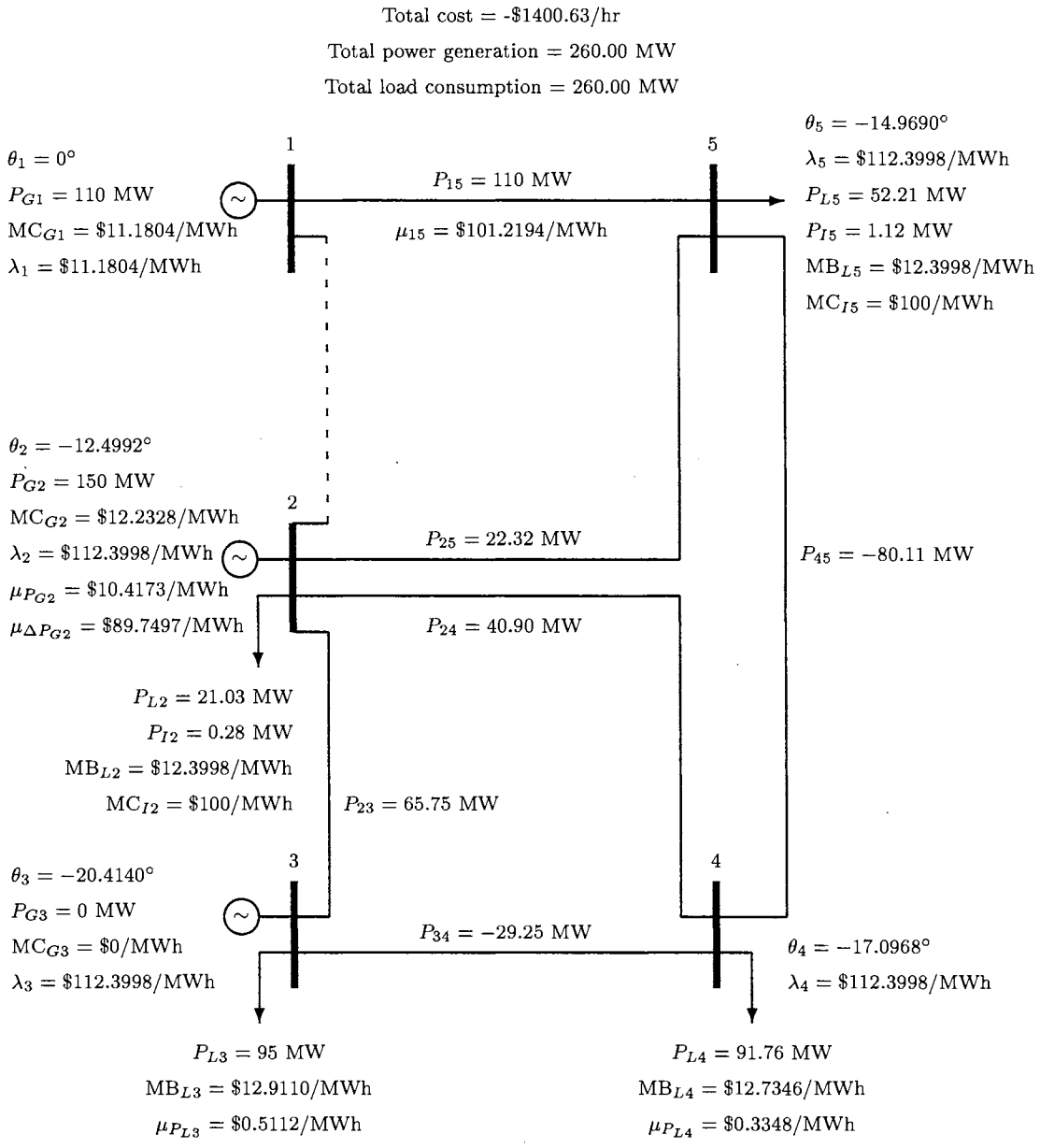

Figure 5.13: Final (converged) solution of subproblem 1 of (DC) ESCOPF.

At bus 2, the Lagrange multiplier $\mu_{P_{G2}} = \$10.4173/\text{MWh}$ is the multiplier of the binding constraint on generator 2 resulting from its maximum capacity limit. If generator 2 had been allowed to generate one more MW when the contingency on line 1-2 occurred, the optimal objective function of subproblem 1 would have been improved by \$10.4173/MWh.

Since the actual maximum capacity limit of generator 2 is equal to its post-contingency limit resulting from the maximum ramping up rate, when generator 2 hits the actual maximum capacity limit, it hits the ramping up limit as well. That is, at post-contingency 1,

$$P^1_{G2} = P^1_{G2_{\max}} = P^0_{G2} + \Delta^+_{\max} P_{G2}.$$

The Lagrange multiplier $\mu_{\Delta P_{G2}} = \$89.7497/\text{MWh}$ is the multiplier of the binding constraint on the maximum ramping up limit of generator 2. It can be interpreted as how much the optimal post-contingency objective function would have been improved by increasing the ramping up limit by 1 MW. An additional MW of ramping-up capacity is worth more than an additional MW of generator capacity ($\mu_{\Delta P_{G2}} > \mu_{P_{G2}}$). This is because relaxing the maximum generator capacity on generator 2 by 1 MW might increase the pre-contingent real power generation P^0_{G2} by 1 MW from its optimal value and might increase the total cost of power generation. On the other hand, if generator 2 must generate at the maximum limit after the contingency on line 1-2 occurs, having one more MW of ramping up capacity at generator 2 would reduce the pre-contingent real power generation P^0_{G2} by 1 MW.

It is obvious that the optimal pre-contingency real power generation of gener-

ator 2, $P_{G2}^0 = 115$ MW, illustrated in Figure 5.12, results from subproblem 1 because P_{G2}^1 must generate at its maximum limit after contingency 1, and its maximum ramping rate is 35 MW.

The multipliers $\mu_{\Delta P_{G2}}$ and $\mu_{P_{G2}}$, corresponding to the ramping-up constraint and the maximum capacity limit, respectively, can be used to calculate the marginal value of spinning reserve on generator 2 as discussed in Section 4.3.1. As a result, the marginal value of spinning reserve on generator 2 for contingency 1 is:

$$\begin{aligned} \pi^1 \cdot (\mu_{\Delta P_{G2}} + \mu_{P_{G2}}) &= 0.01 \cdot (\$89.7497/\text{MW} + \$10.4173/\text{MW}) \\ &= \$1.0017/\text{MW} \end{aligned}$$

Subproblem 2

Losing line 1-2 does not do much damage to the system because there is no load interruption at the post-contingency state. The optimal solution of subproblem 2 given the pre-contingency in Figure 5.12 is illustrated in Figure 5.14. The only binding constraint in this subproblem is the constraint on the flow limit on line 3-4, with $\mu_{P_{34}} = \$17.8985/\text{MWh}$. Line 3-4 is the only line that is connected to the load at bus 3 after the contingency on line 2-3 and has the emergency limit of 95 MW. The minus sign of the real power flow in the line determines the direction of the flow: $P_{ij} = p$ MW means that the real power p MW is flowing from bus i to bus j; $P_{ij} = -p$ MW means that the real power p MW is flowing from bus j to bus i.

It is not a coincidence that the pre-contingent load at bus 3 is exactly 95 MW. The post-contingency limit on line 3-4 must have determined the optimal pre-contingent

load at bus 3 in order to avoid the load interruption when contingency 2 occurs.

Subproblem 3

Subproblem 3 is the contingency on line 2-4. The pre-contingency state in Figure 5.12 can survive this contingency without interrupting any customer load by redispatching the system as shown in Figure 5.15. The emergency limit on line 4-5 is binding with the multiplier $\mu_{P_{45}} = \$22.3529/\text{MWh}$.

Subproblem 4

Subproblem 4 is the contingency on line 3-4. The optimal solution of subproblem 4 given the pre-contingency in Figure 5.12 is illustrated in Figure 5.16. The emergency maximum limit on line 2-3 is binding with the multiplier $\mu_{P_{23}} = \$18.2955/\text{MWh}$ because it is the only line that is connected to the 95 MW load at bus 3.

Generator 2 is binding at its minimum post-contingency limit, $P^4_{G2} = 80$ MW. The minimum post-contingency limit on generator 2 comes from the pre-contingent real power generation of generator 2 and its maximum ramping down limit. The Lagrange multiplier $\mu_{\Delta P_{G2}} = \$0.8267/\text{MWh}$ can be interpreted as how much the objective function of subproblem 4 would have been improved if generator 2 had been allowed one more MW of ramping down capacity. The value of this multiplier is not significant because the marginal cost of generator 2, MC_{G2}, is close to the Lagrange multiplier λ_2 which is the optimal bus price. In fact, $\mu_{\Delta P_{G2}}$ is the difference between the marginal cost of generator 2 and the optimal price at bus 2.

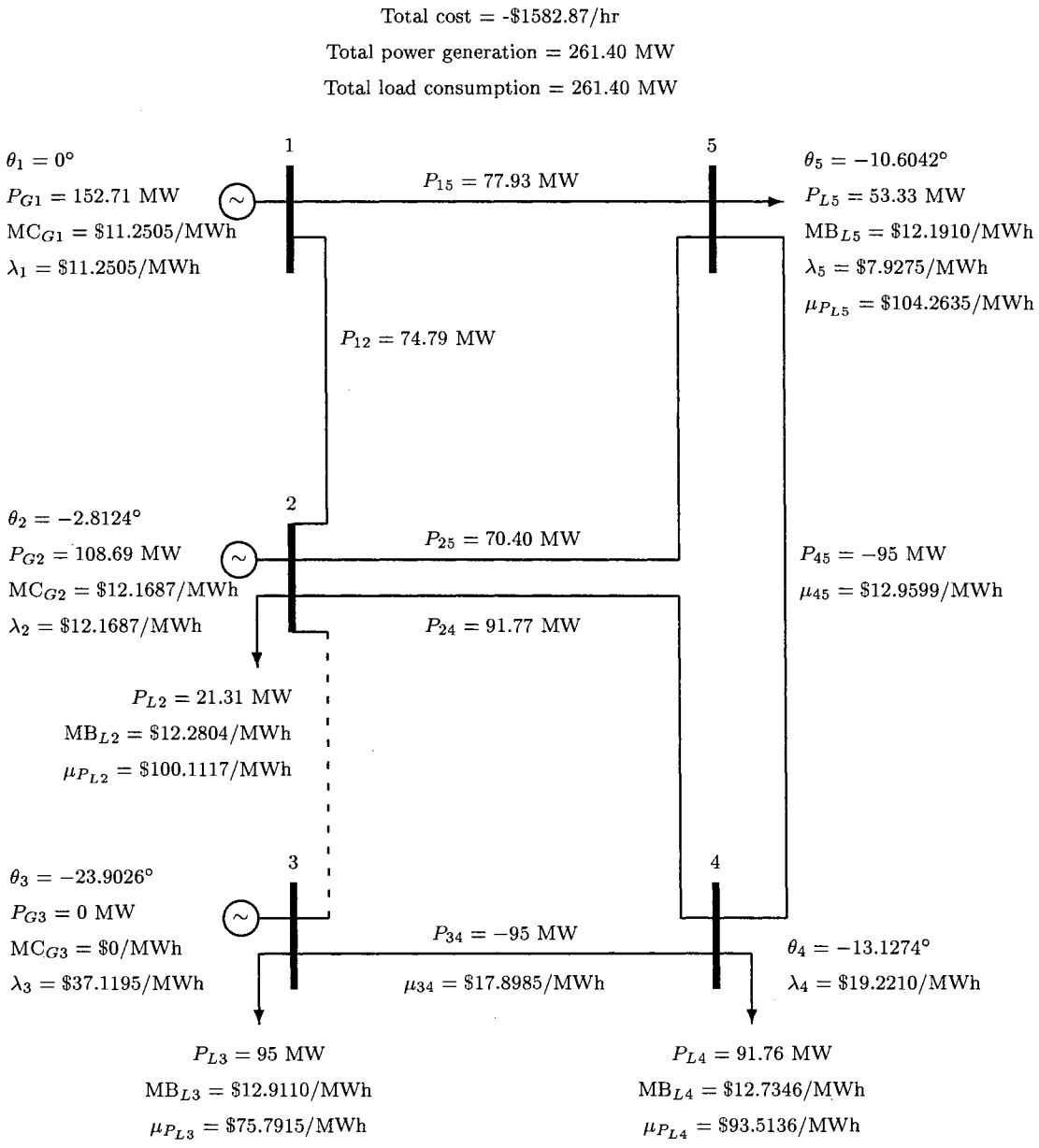

Figure 5.14: Final (converged) solution of subproblem 2 of (DC) ESCOPF.

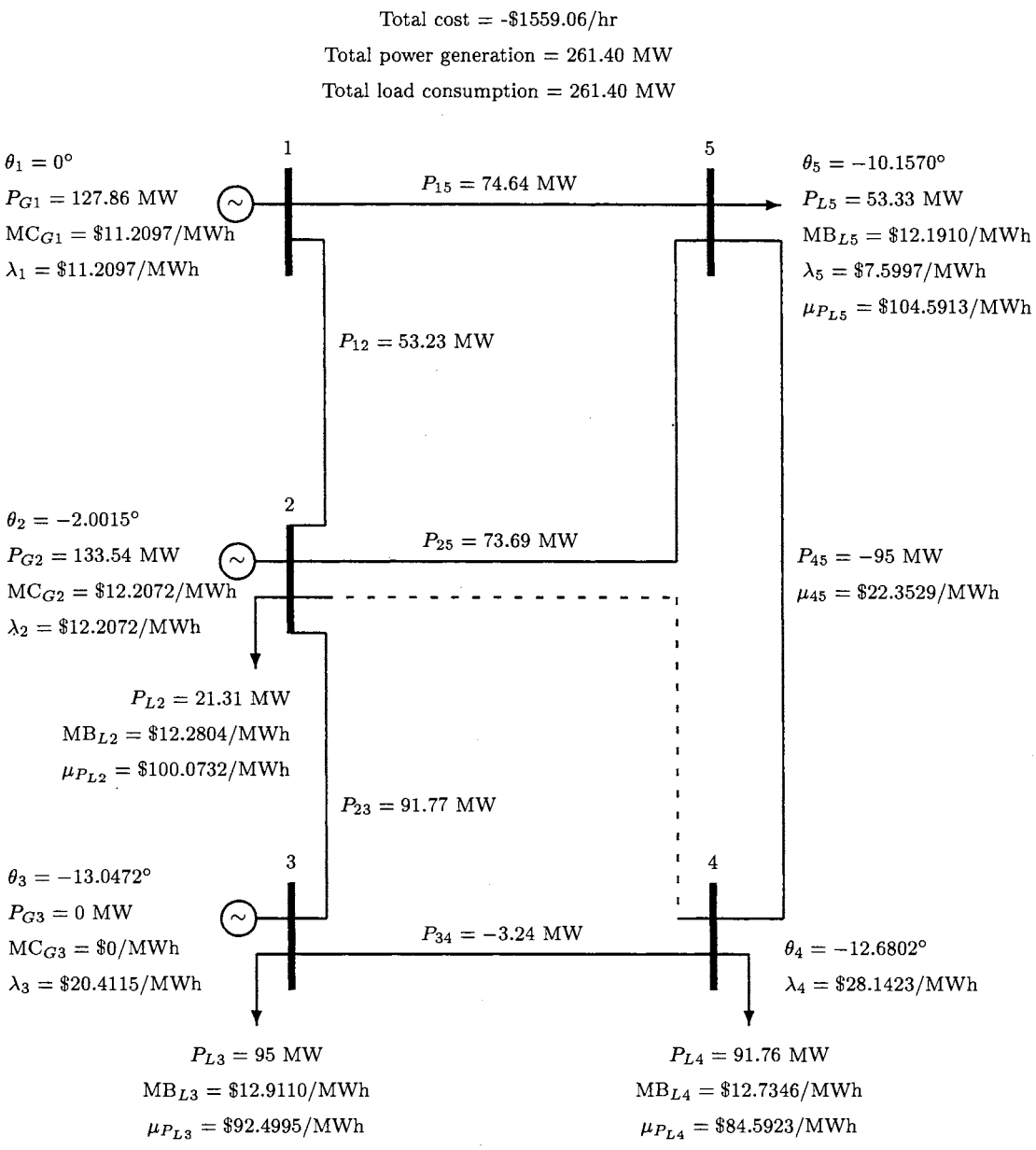

Figure 5.15: Final (converged) solution of subproblem 3 of (DC) ESCOPF.

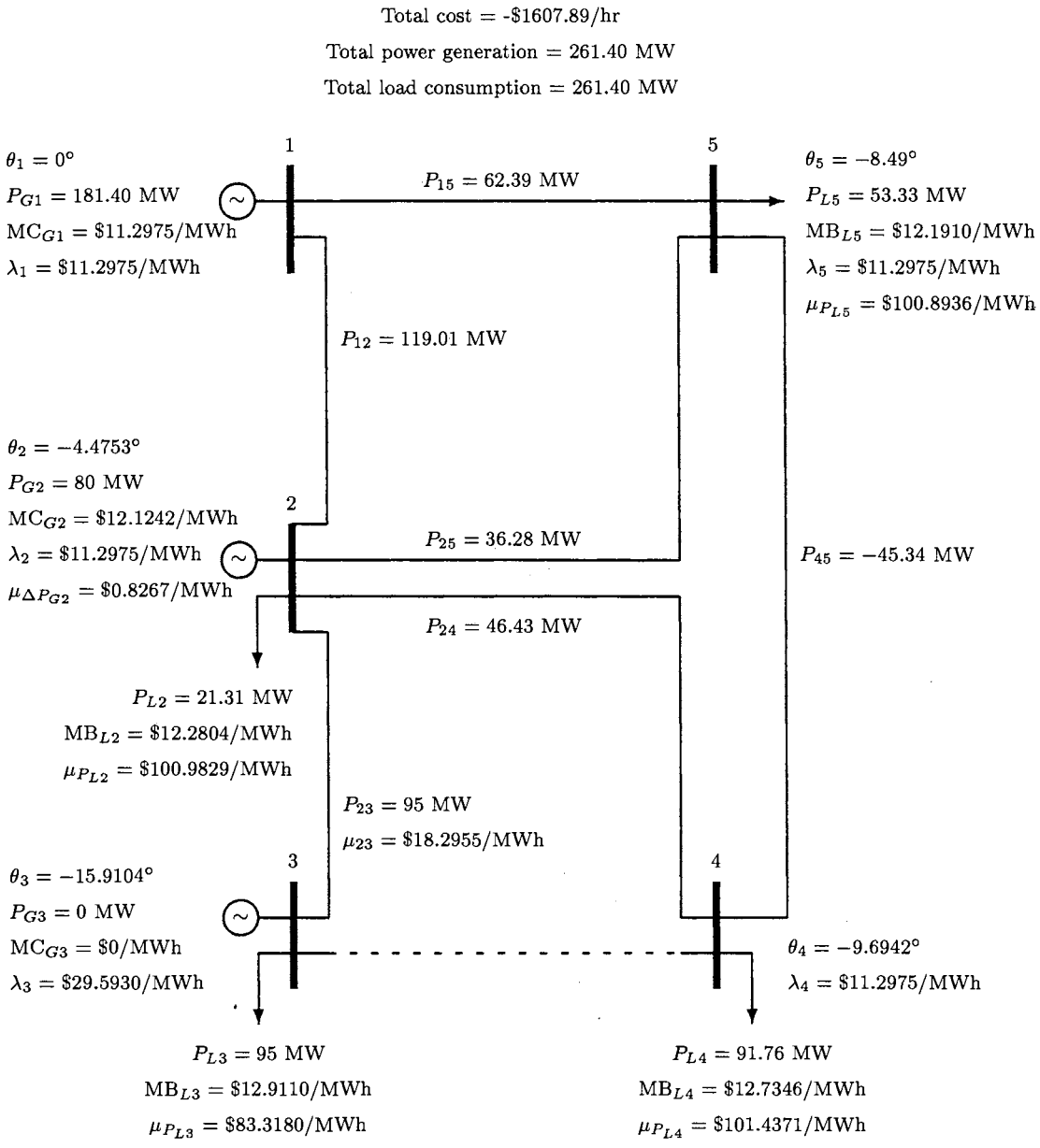

Figure 5.16: Final (converged) solution of subproblem 4 of (DC) ESCOPF.

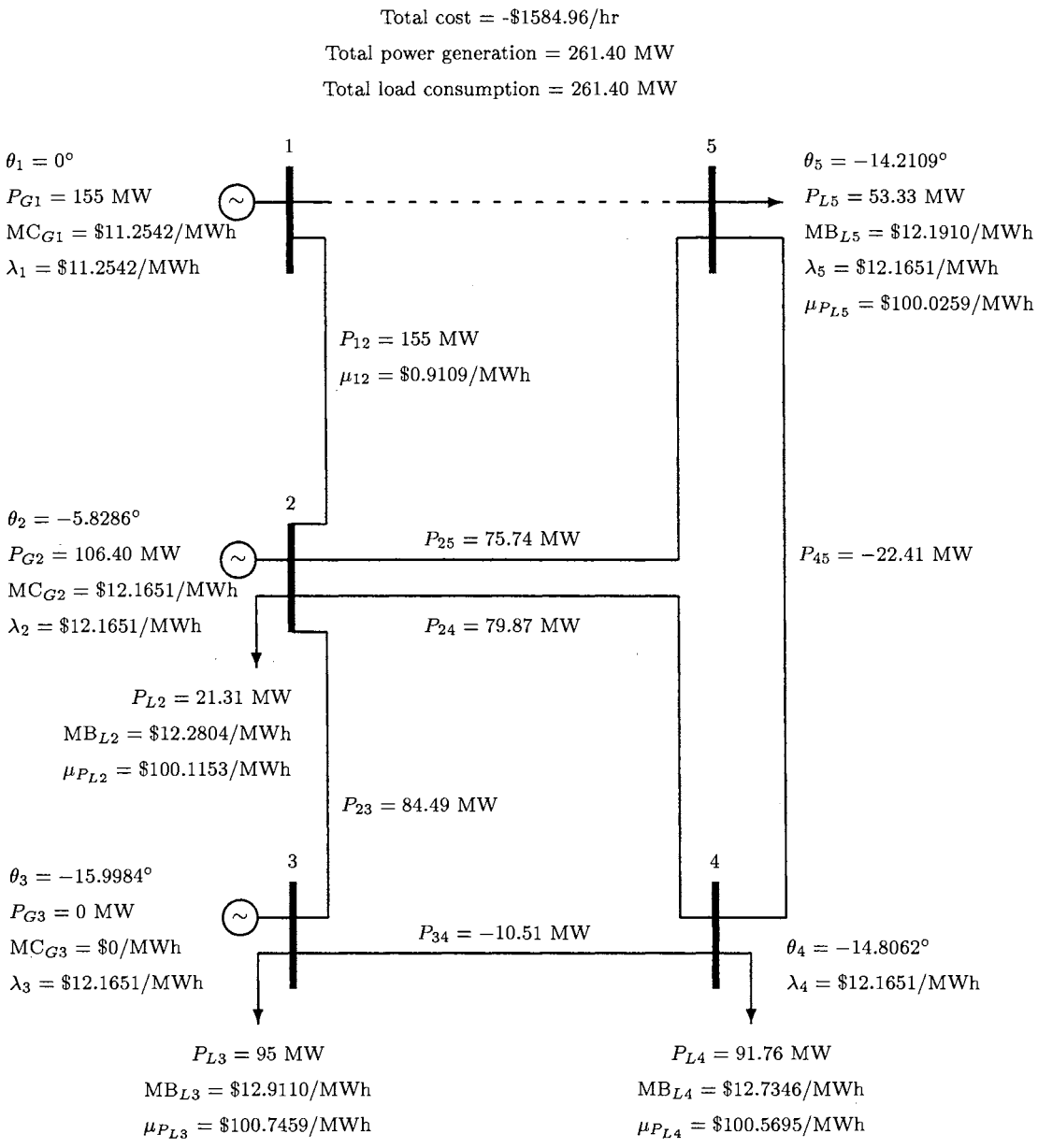

Figure 5.17: Final (converged) solution of subproblem 5 of (DC) ESCOPF.

Subproblem 5

By losing line 1-5, the only line connected to generator 1 is line 1-2 which has an emergency limit at post-contingency of 155 MW. Therefore, generator 1 can generate up to 155 MW without hitting its maximum post-contingency limit. Generator 2 can easily pick up the rest of the load without hitting the post-contingency limit. Since there is no constraint binding on the generation at bus 2, the marginal cost of generator 2 is equal to the bus price λ_2.

Subproblem 6

The optimal solution of subproblem 6 shown in Figure 5.18 is similar to the one of subproblem 4. The flows in the lines of subproblem 6 may be different than the ones of subproblem 4, but the dispatch of the generators is the same. There is no binding constraint at this post-contingency state. Therefore, all the Lagrange multipliers λ_i are the same at all buses. The minimum ramping down of the real power generation at bus 2 is binding with the multiplier $\mu_{\Delta P_{G2}} = \$0.8267/\text{MWh}$.

Subproblem 7

The optimal solution of subproblem 7 shown in Figure 5.19 is similar to the solutions of subproblems 4 and 6. Subproblem 7 has the same generator dispatch as subproblems 4 and 6 even though the flows in the lines are different. The flow limit on line 2-4 is binding with $\mu_{P_{24}} = \$33.0385/\text{MWh}$.

The total costs or the objective functions of subproblems 2, 4, 5, 6, and 7 are obviously higher than the objective function of the pre-contingency state (main

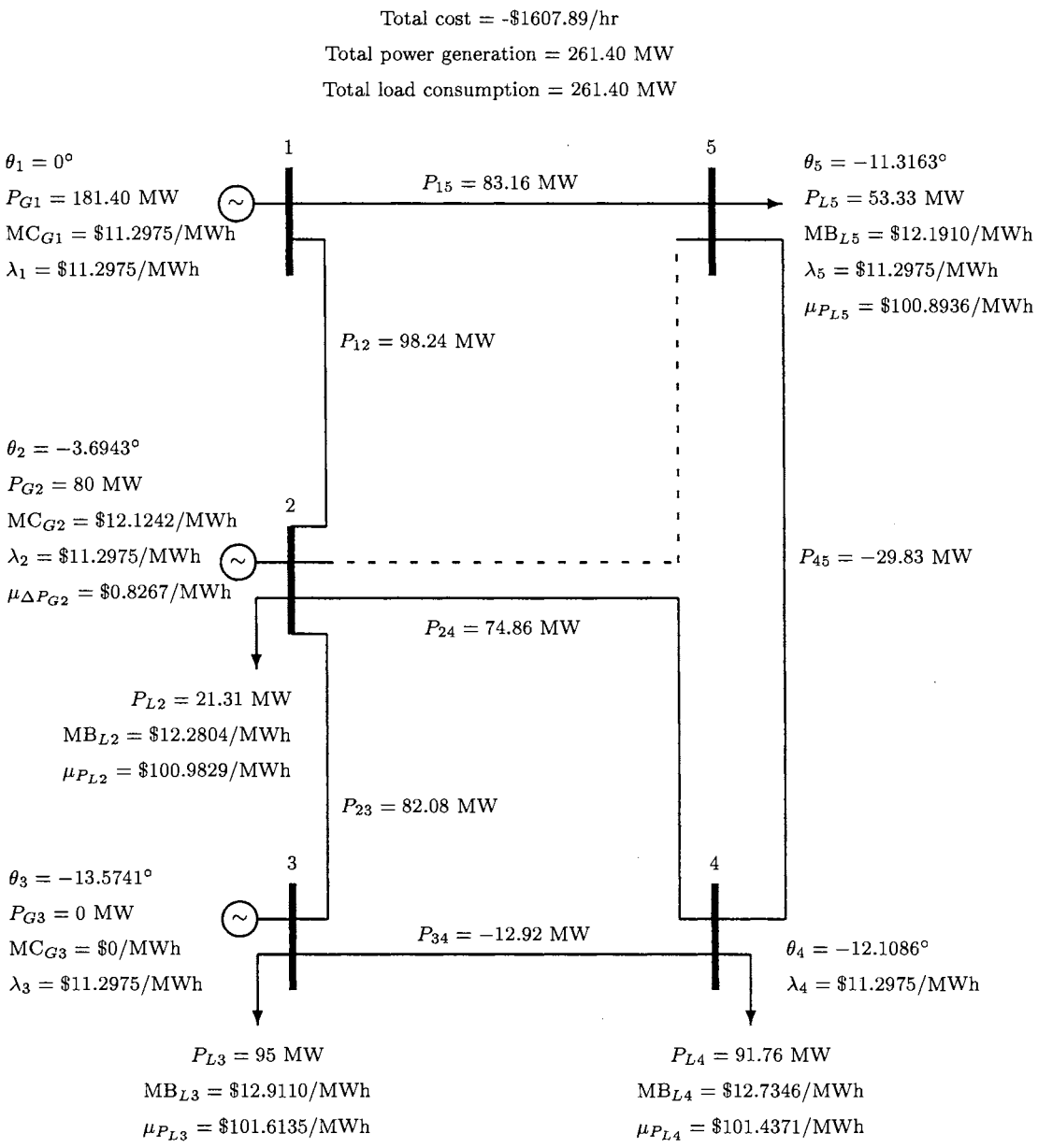

Figure 5.18: Final (converged) solution of subproblem 6 of (DC) ESCOPF.

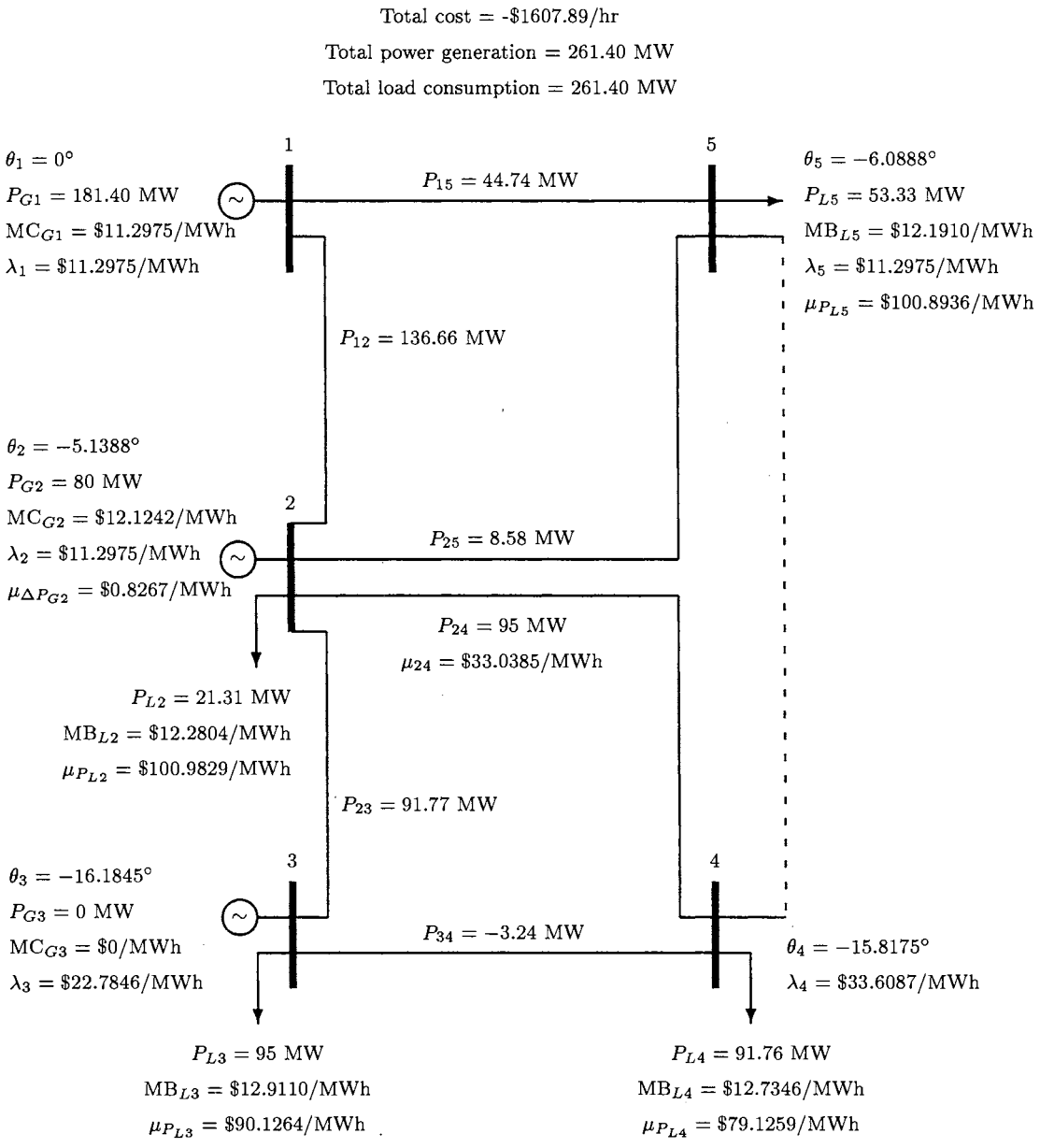

Figure 5.19: Final (converged) solution of subproblem 7 of (DC) ESCOPF.

problem). It may seem like the 5-bus system is getting better or making more money when losing one of these lines. However, these subproblems are optimal but not secure while the optimal pre-contingency state in Figure 5.12 is secure. If we would like to compare the total cost of any subproblem to the total cost of the ESCOPF pre-contingency, we would have to solve the ESCOPF problem on a particular subproblem, treating the subproblem as a pre-contingent state, to find out its secure optimal operating cost. On the other hand, we can compare these subproblems to the standard OPF problem shown in Figure 5.20 because the standard OPF problem minimizes the operating costs without taking into account the security. As a result, losing one of these lines does not make the system better because their total operating costs are higher than the total cost of the standard OPF system.

Since the maximum constraints (both ramping-up constraint and capacity constraint) on generator 2 are not binding in subproblems 2-7: the Lagrange multipliers corresponding to the maximum limits on generator 2 of the subproblems are all zero, the marginal value of spinning reserve of generator 2 results from the spinning reserve of only contingency 1, which is \$1.0017/MW as computed in the discussion of subproblem 1.

In the 5-bus case, we do not run out of interruptible load. Thus, the marginal values of interruptible load are zero in all subproblems.

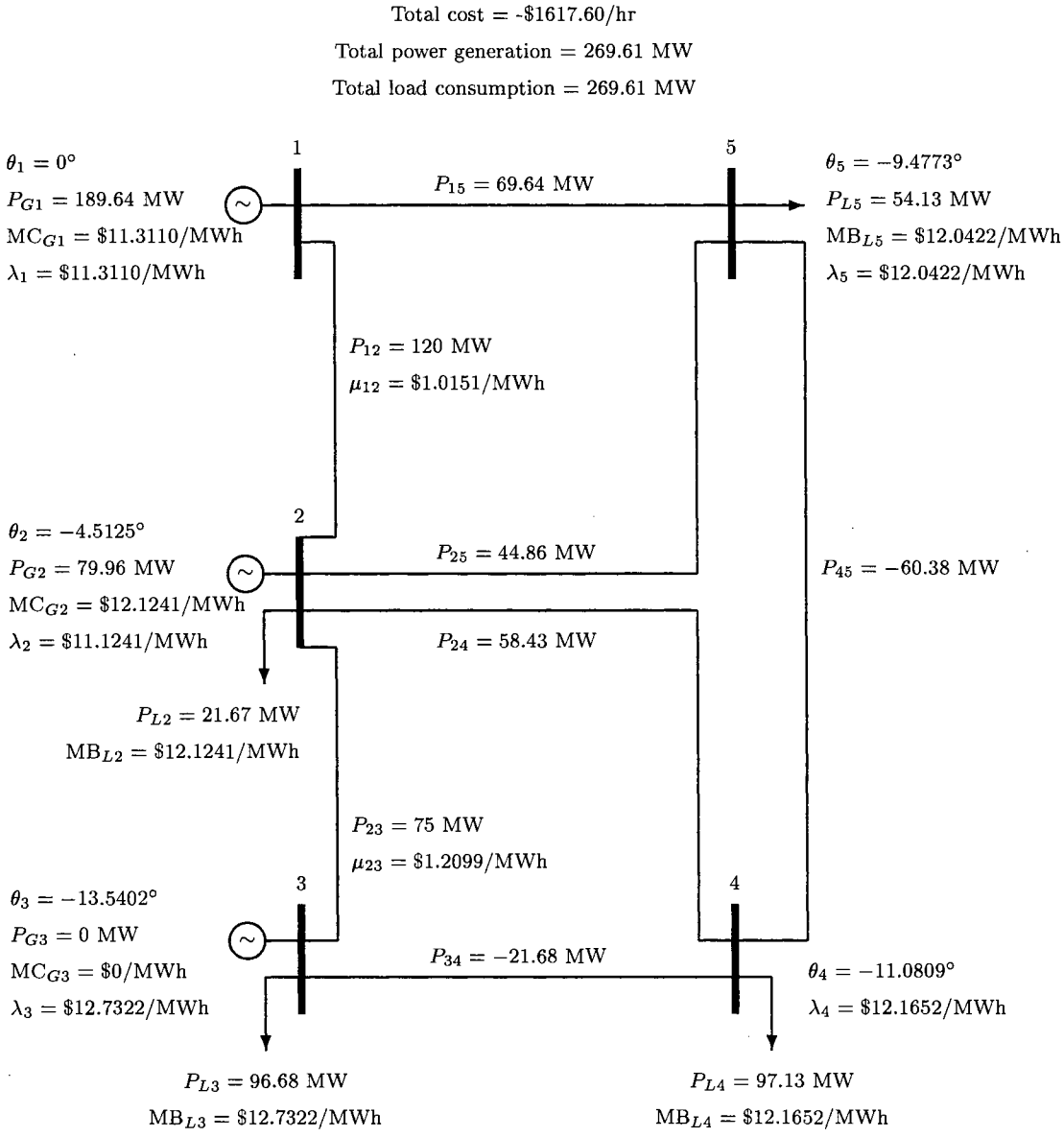

Figure 5.20: Standard (DC) OPF of 5-bus system, ignoring contingencies.

Standard OPF vs. ESCOPF

Figure 5.20 illustrates the solution of the standard OPF for the 5-bus case without taking into consideration the contingencies. However, it is not used as an initial pre-contingency state to solve each subproblem as was done in the decomposition method. We provide the base case OPF solution of the 5-bus case in order to compare it with the optimal "secure" pre-contingency solution resulting from the ESCOPF model.

The results of the base-case OPF and the ESCOPF are summarized in Table 5.8. The total power generation and consumption of the OPF is 8.21 MW more

	OPF (DC)	ESCOPF (DC)
$P_{G_{total}}$ (MW)	269.61	261.40
$P_{L_{total}}$ (MW)	269.61	261.40
Cost (\$/MWh)	-1617.60	-1577.01
Expected Security Cost (\$/MWh)	N/A	-1576.13
P_{G1} (MW)	189.64	146.40
P_{G2}	79.69	115†
MC_{G1} (\$/MWh)	11.31	11.24
MC_{G2} (\$/MWh)	12.12	12.18
P_{L2} (MW)	21.67	21.31
P_{L3}	96.68	95†
P_{L4}	97.13	91.76
P_{L5}	54.13	53.33
P_{12} (MW)	120	85.19
P_{15}	69.64	61.21
P_{23}	75	74.17
P_{24}	58.43	58.40
P_{25}	44.86	46.31
P_{34}	-21.68	-20.83
P_{45}	-60.38	-54.19

Table 5.8: Solutions of standard OPF and ESCOPF of (DC) 5-bus system.

than the total power generation of the ESCOPF. Since we solve this 5-bus case with DC approximation, the real power losses are zero. The total cost of the ESCOPF is higher than that of the OPF because the system operating under the ESCOPF is optimal and secure, while the OPF system is optimal but not secure.

The power generation P_{G2} at bus 2 of the ESCOPF is much higher than that of the OPF because of the contingency 1 as discussed earlier. The OPF system is operating at its optimal solution without considering the contingencies. If contingency 1 occurs to the OPF system, generator 2 will only be able to ramp up to 114.69 MW due to 35 MW of its maximum ramping up capacity, and generator 1 has to reduce its output to 110 MW due to the constraint on the remaining transmission line. As a result, 44.92 MW of load must be interrupted for the system to survive that contingency. If the system fails to interrupt that amount of load, the whole system will collapse due to the cascading failure of the system equipment. On the other hand, if the system is generating at the ESCOPF solution when contingency 1 occurs, the system only drops 1.40 MW of load to survive the contingency. This amount of load interruption is insignificant when it is compared to the total load consumption.

Since generator 2 has higher marginal cost than generator 1 at the given outputs ($\mathrm{MC}_{G2} > \mathrm{MC}_{G1}$), generating more on generator 2 increases the total cost of the ESCOPF system.

The total consumption of load in ESCOPF is lower than that of the OPF because the ESCOPF system takes into consideration the load interruption when a contingency occurs; more load consumed at the pre-contingency state results in more load being interrupted when it is necessary. The load consumption at bus 3

in the ESCOPF system, $P_{L3} = 95$ MW, results from contingencies 2 and 4. After contingency 2, the remaining line connected to the load at bus 3 has the emergency maximum capacity of 95 MW. Similarly, the remaining line connected to bus 3 when contingency 4 occurs has the emergency maximum capacity of 95 MW. If P_{L3} is 96.68 MW as in the OPF system, when one of these two contingencies occurs, the load interruption at bus 3 can not be avoided. Although these contingencies do not cause a blackout in the OPF system because of the advantage of interruptible load, the system still suffers from a fairly high interruption cost.

In the OPF system (see Figure 5.20), generator 1 tries to generate as much as possible to lower the total cost of generation due to its lower marginal cost ($\mathrm{MC}_{G1} < \mathrm{MC}_{G2}$). However, since the flow in line 1-2 has reached its maximum limit, it prevents generator 1 from pushing another MW into the network.

5.2.2 5-bus Case: AC System

Since the DC approximation of the 5-bus case yielded satisfied results, the ESCOPF model was then tested on its exact system, which is called the "AC" power system. The power flow equations of the AC system are now nonlinear and involved voltage magnitudes as expressed in Chapter 2. The reactive power flows of the system are no longer zero as in the DC approximation. Therefore, there are real losses in transmission lines which cause different optimal dispatches from the DC version. The solutions involving reactive power and voltages of the 5-bus AC system are illustrated in Figures 5.21b, 5.22b, 5.23b, 5.24b, 5.25b 5.26b, 5.27b, and 5.28b. The results of the ESCOPF of the 5-bus AC system, illustrated in Figures 5.21a, 5.22a, 5.23a, 5.24a, 5.25a 5.26a, 5.27a, and 5.28a, have the same type of data as the results of the DC case because they are used to compared with the DC approximation. The OPF solution of the AC 5-bus system is shown in Figures 5.29a and 5.29b.

The solutions of the ESCOPF for the 5-bus AC system are compared to those of the DC system in order to see if the DC approximation is reasonable for the ESCOPF problem. The optimal pre-contingency solution and the solution of the standard OPF of the 5-bus AC system are summarized and compared to the solutions of the DC case in Table 5.9. The results of the AC case are not quite different from the results of the DC case. Since the behavior between the standard OPF and the ESCOPF in the AC case is similar to that of the DC case, the explanation of this behavior is therefore the same as in the DC case discussed earlier. However, there is a piece of important information regarding load interruption in subproblem 1 of the DC and

	DC		AC	
	OPF	ESCOPF	OPF	ESCOPF
$P_{G_{total}}$ (MW)	269.61	261.40	269.78	259.93
$P_{L_{total}}$ (MW)	269.61	261.40	258.84	251.16
Cost (\$/MWh)	-1617.60	-1577.01	-1487.60	-1466.91
Expected Security Cost (\$/MWh)	N/A	-1576.13	N/A	-1464.58
P_{G1} (MW)	189.64	146.40	198.18	151.74
P_{G2}	79.69	115	71.60	108.18
MC_{G1} (\$/MWh)	11.31	11.24	11.33	11.25
MC_{G2} (\$/MWh)	12.12	12.18	12.11	12.17
P_{L2} (MW)	21.67	21.31	21.70	21.56
P_{L3}	96.68	95	94.06	88.92
P_{L4}	97.13	91.76	91.60	90.11
P_{L5}	54.13	53.33	51.48	50.56
I_{12}/I_{21} (MW)	120	85.19	120/119.35	86.42/85.23
I_{15}/I_{51}	69.64	61.21	64.57/65.03	56.34/56.72
I_{23}/I_{32}	75	74.17	70.13/70.09	67.35/67.34
I_{24}/I_{42}	58.43	58.40	54.25/54.59	54.42/54.79
I_{25}/I_{52}	44.86	46.31	41.43/41.88	42.94/43.40
I_{34}/I_{43}	-21.68	-20.83	24.31/26.01	21.09/23.13
I_{45}/I_{54}	-60.38	-54.19	56.53/56.54	51.01/51.02

Table 5.9: Solutions of standard OPF and ESCOPF of DC and AC 5-bus systems.

AC 5-bus cases, which is analyzed as follows.

In contingency subproblem 1 shown in Figure 5.22a, there is no load interruption in the AC case, but there is a small amount of load interruption in subproblem 1 of the DC case. Therefore, the solutions of subproblem 1 of DC and AC systems are summarized and compared in Table 5.10.

Since the amount of load supplied at the pre-contingency state in the AC case is smaller than the total consumption in the DC case due to the real power losses in the network, when contingency 1 occurs to the AC case, the load interruption can be avoided. Moreover, the amount of load interruption in the DC case is fairly small and can be ignored. Therefore, it is reasonable that the AC case does not

	Subproblem 1	
	DC	AC
$P_{G_{total}}$ (MW)	261.40	261.98
$P_{L_{total}}$ (MW)	261.40	251.16
Cost ($/MWh)	-1400.63	-1409.80
P_{G1} (MW)	110	118.80
P_{G2}	150	143.18
P_{L2} (MW)	21.03	21.56
P_{L3}	95	88.92
P_{L4}	91.76	90.11
P_{L5}	52.21	50.56
P_{I2} (MW)	0.28	0
P_{I5}	1.12	0
I_{15}/I_{51} (MW)	110	110/ 109.54
I_{23}/I_{32}	65.75	58.60/58.75
I_{24}/I_{42}	40.90	37.75/38.98
I_{25}/I_{52}	22.32	22.73/25.08
I_{34}/I_{43}	29.25	30.44/32.37
I_{45}/I_{54}	80.11	72.21/78.43

Table 5.10: Solutions of subproblem 1 for DC and AC 5-bus systems.

have to interrupt any load at the post-contingency state. Because of the losses in the transmission lines, the total power generation of the AC case has to be higher than its total load consumption in order to cover the losses. Therefore, the total generation of the AC case is almost the same as that of the DC case even though the total load consumptions of the two cases are different.

In the AC system, the flow from bus i to bus j is different from the flow from bus j to bus i due to the losses. However, the line flows in the AC and DC cases are not much different. We can see that the DC and AC systems of subproblem 1 do not yield much different answers.

The solutions of other subproblems in the AC system are also similar to their

DC approximations. Therefore, we can conclude that the DC approximation can be used as a reasonable approximation for solving the ESCOPF of a power system.

The solution of subproblem 1 illustrated in Figure 5.22a shows that the multiplier of ramping-up constraint on generator 2 is nonzero because generator 2 hits the ramping-up limit in contingency 1. However, the maximum capacity constraint on generator 2 is not binding in this subproblem due to less generation at the pre-contingency state. The multiplier of ramping-up limit $\mu_{\Delta P_{G2}} = \$55.1798/\text{MW}$ can therefore be used to compute the marginal value of spinning reserve of generator 2 for AC 5-bus system. Since there are no binding constraints on maximum limits of generator 2 in other subproblems, the marginal value of spinning reserve on generator 2 is:

$$\pi^1 \cdot \mu_{\Delta P_{G2}} = 0.01 \cdot \$55.1798/\text{MW} = \$0.5518/\text{MW}$$

Similar to the DC 5-bus case, the interruptible load is not exhausted, and therefore the marginal value of interruptible load of the AC 5-bus system is zero.

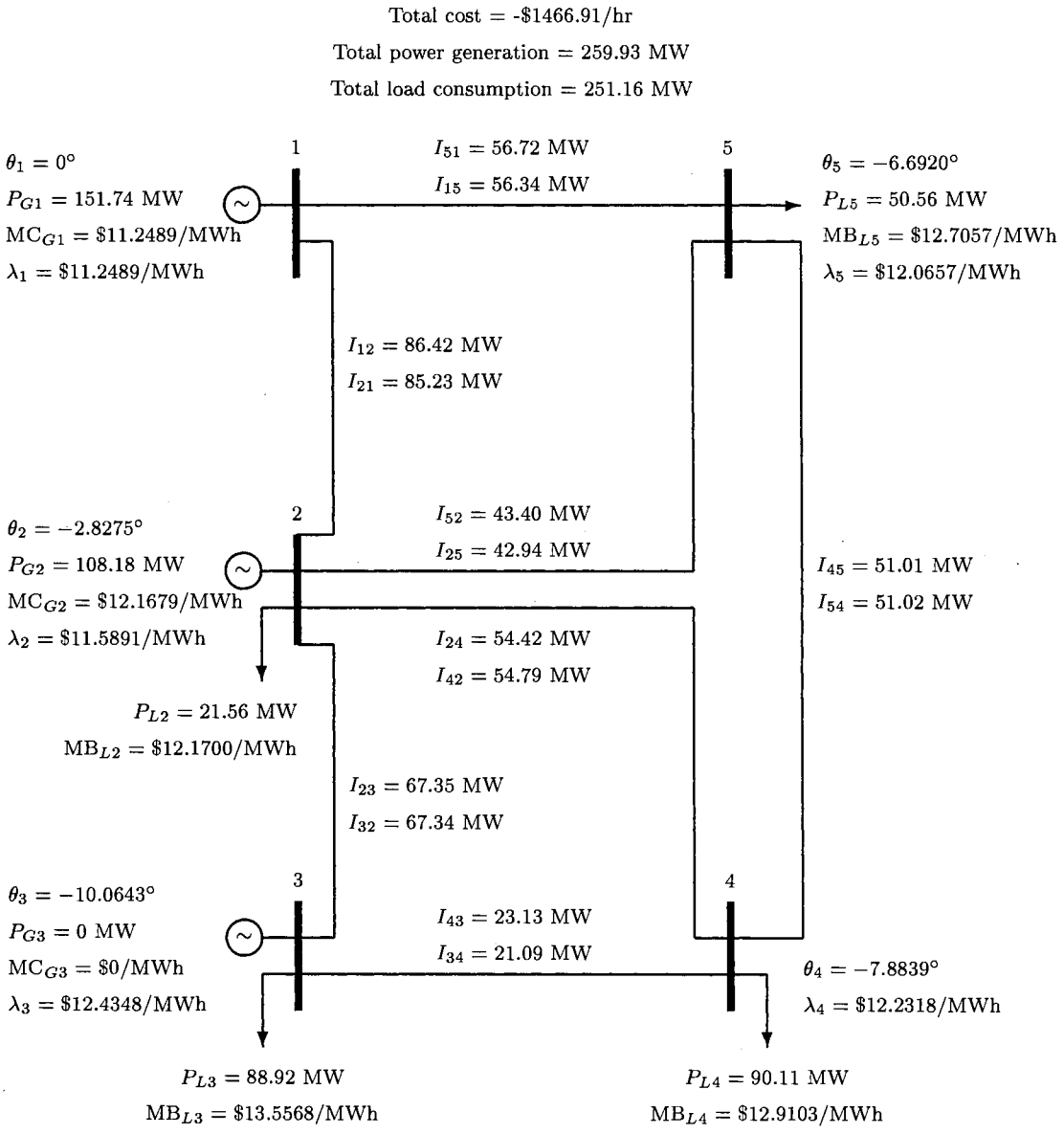

Figure 5.21a: Optimal pre-contingency state of (AC) ESCOPF (final solution of main problem), showing real power, angles, line flows, and prices.

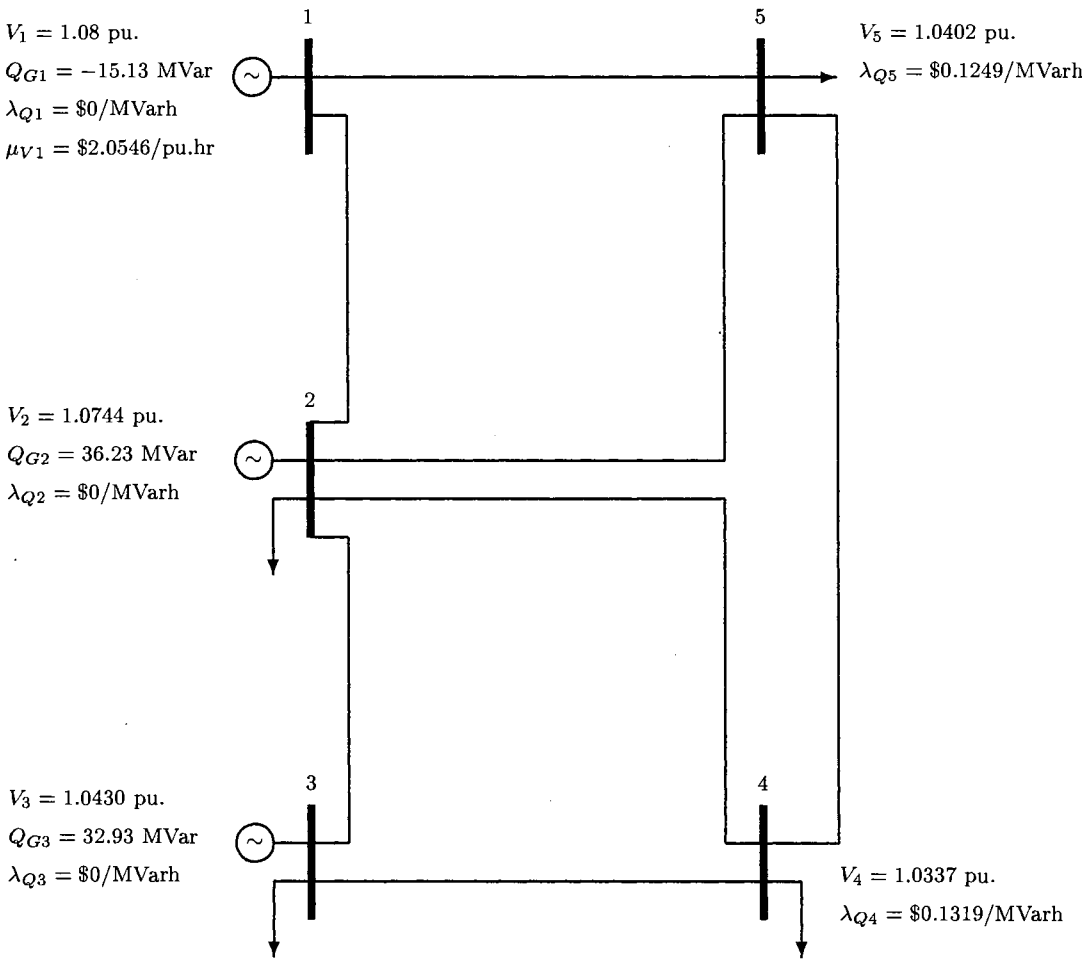

Figure 5.21b: Optimal pre-contingency state of (AC) ESCOPF (final solution of main problem), showing reactive power and voltages.

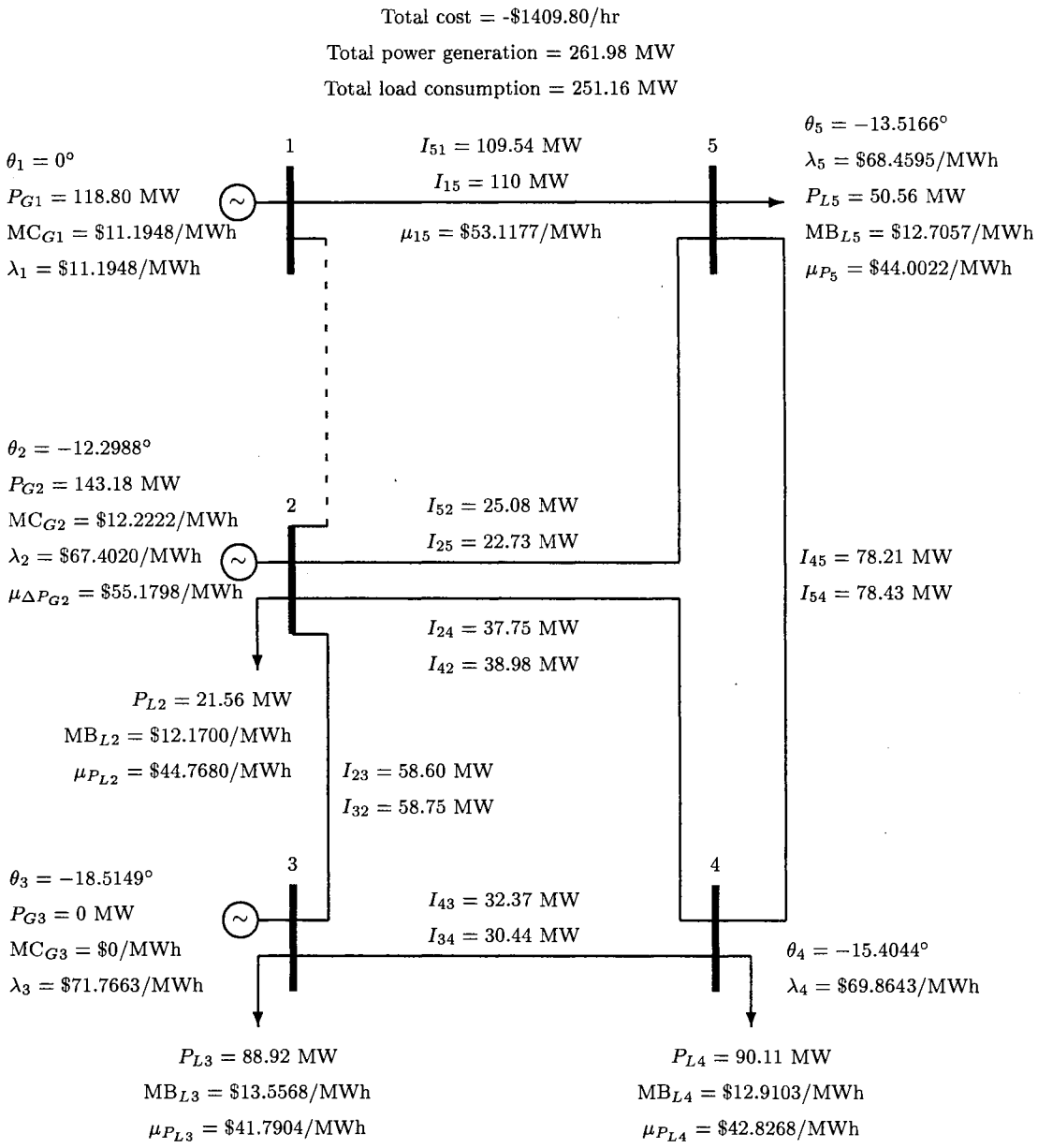

Figure 5.22a: Final (converged) solution of subproblem 1 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

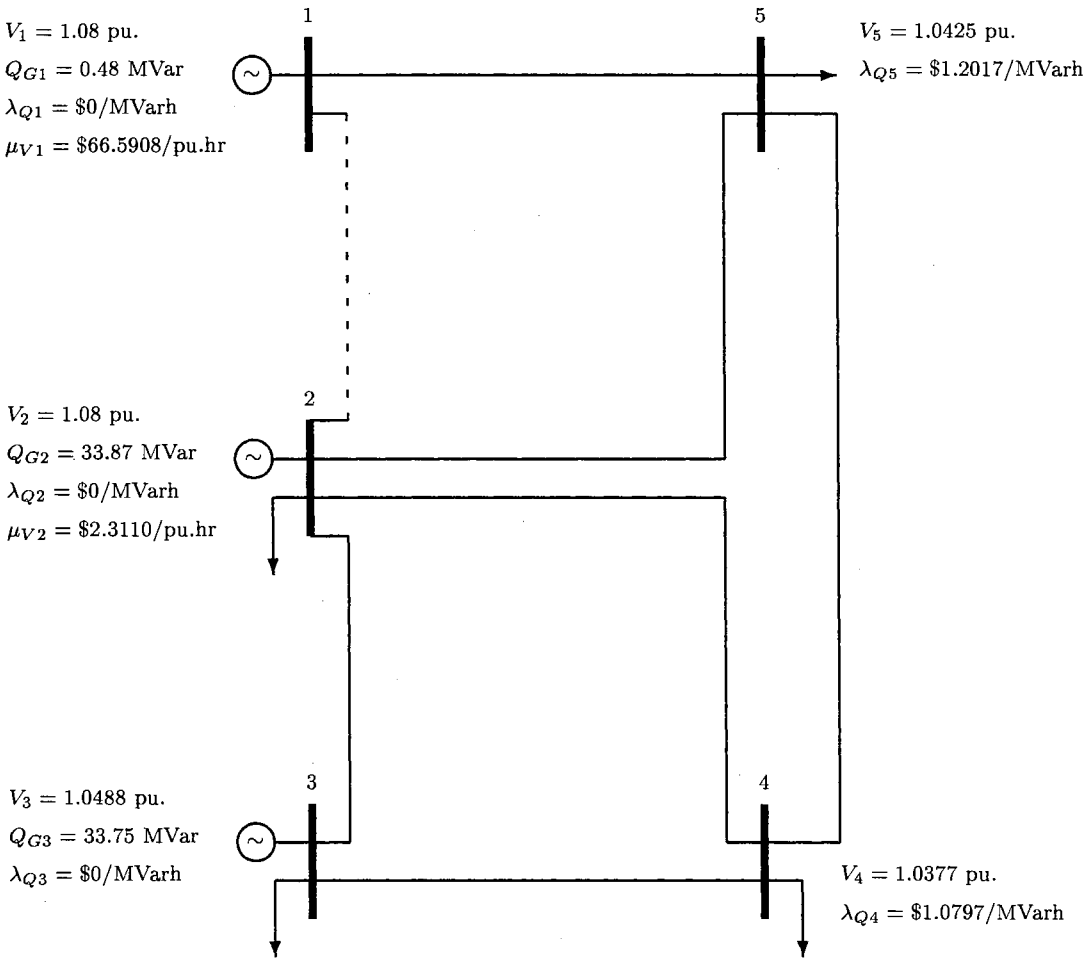

Figure 5.22b: Final (converged) solution of subproblem 1 of (AC) ESCOPF, showing reactive power and voltages.

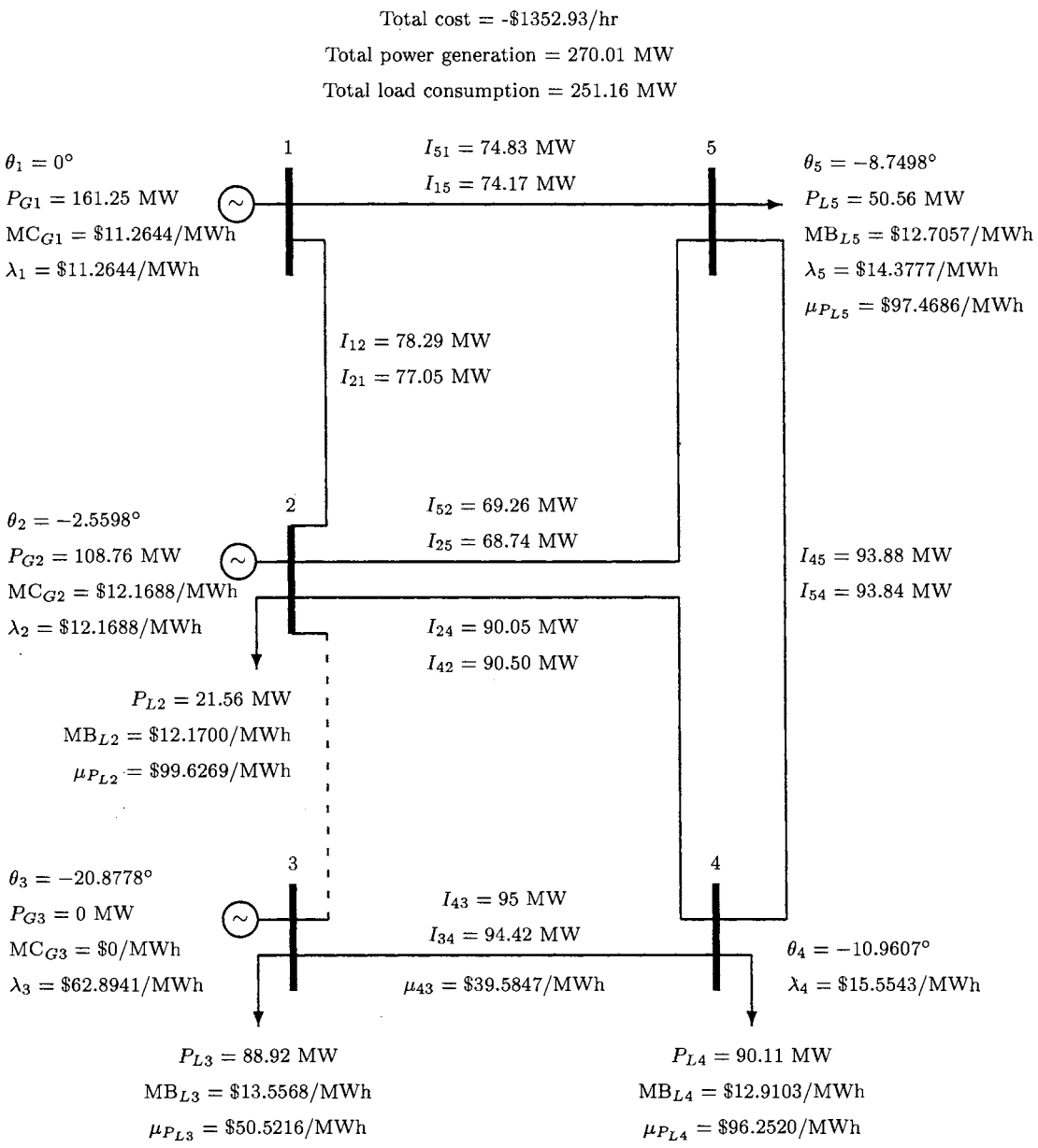

Figure 5.23a: Final (converged) solution of subproblem 2 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

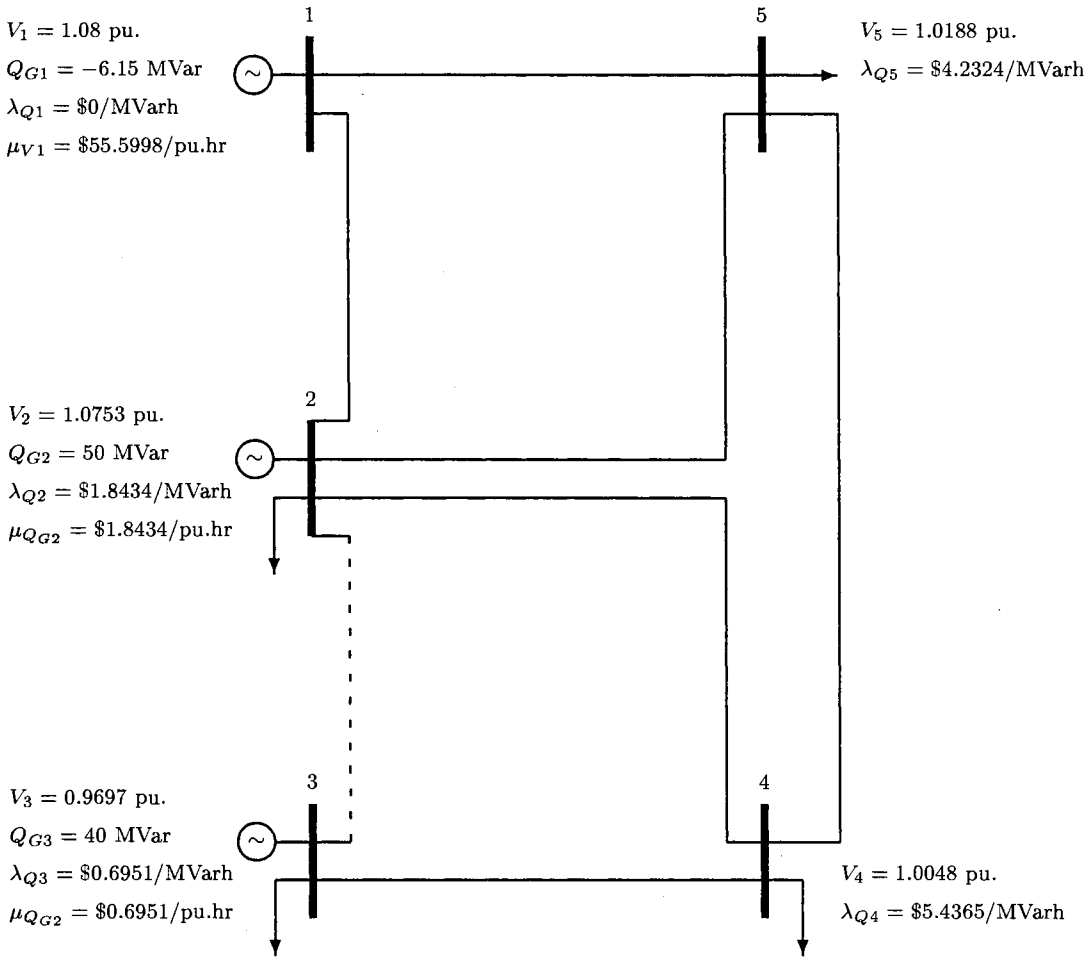

Figure 5.23b: Final (converged) solution of subproblem 2 of (AC) ESCOPF, showing reactive power and voltages.

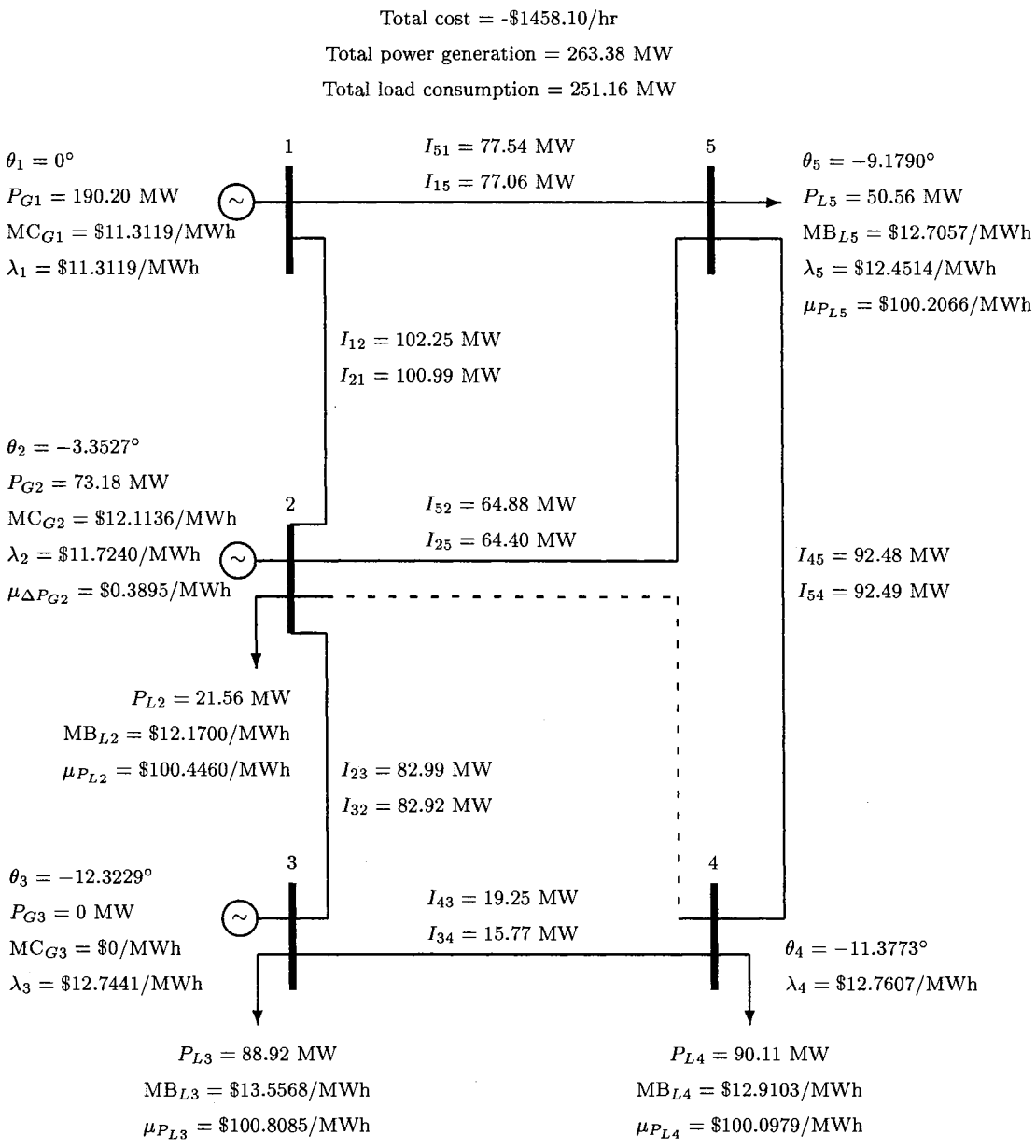

Figure 5.24a: Final (converged) solution of subproblem 3 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

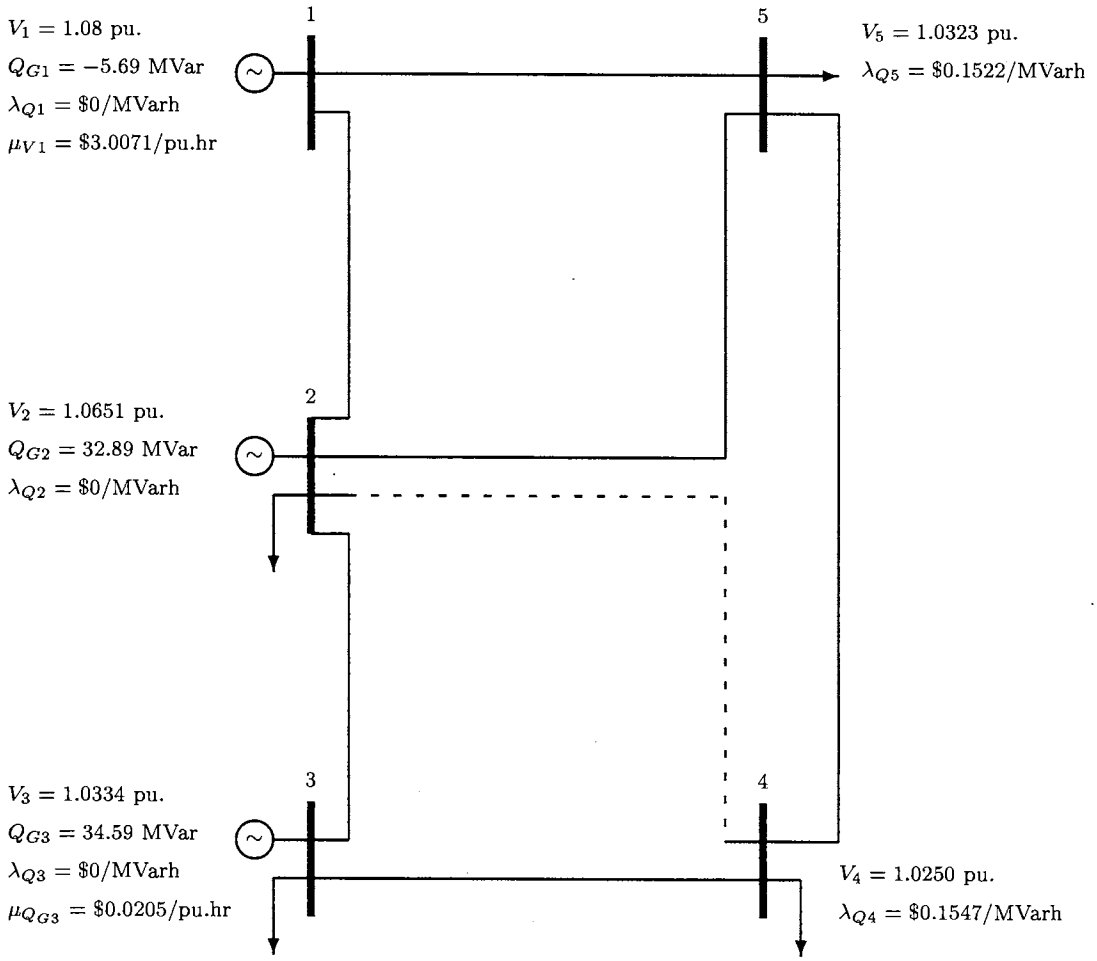

Figure 5.24b: Final (converged) solution of subproblem 3 of (AC) ESCOPF, showing reactive power and voltages.

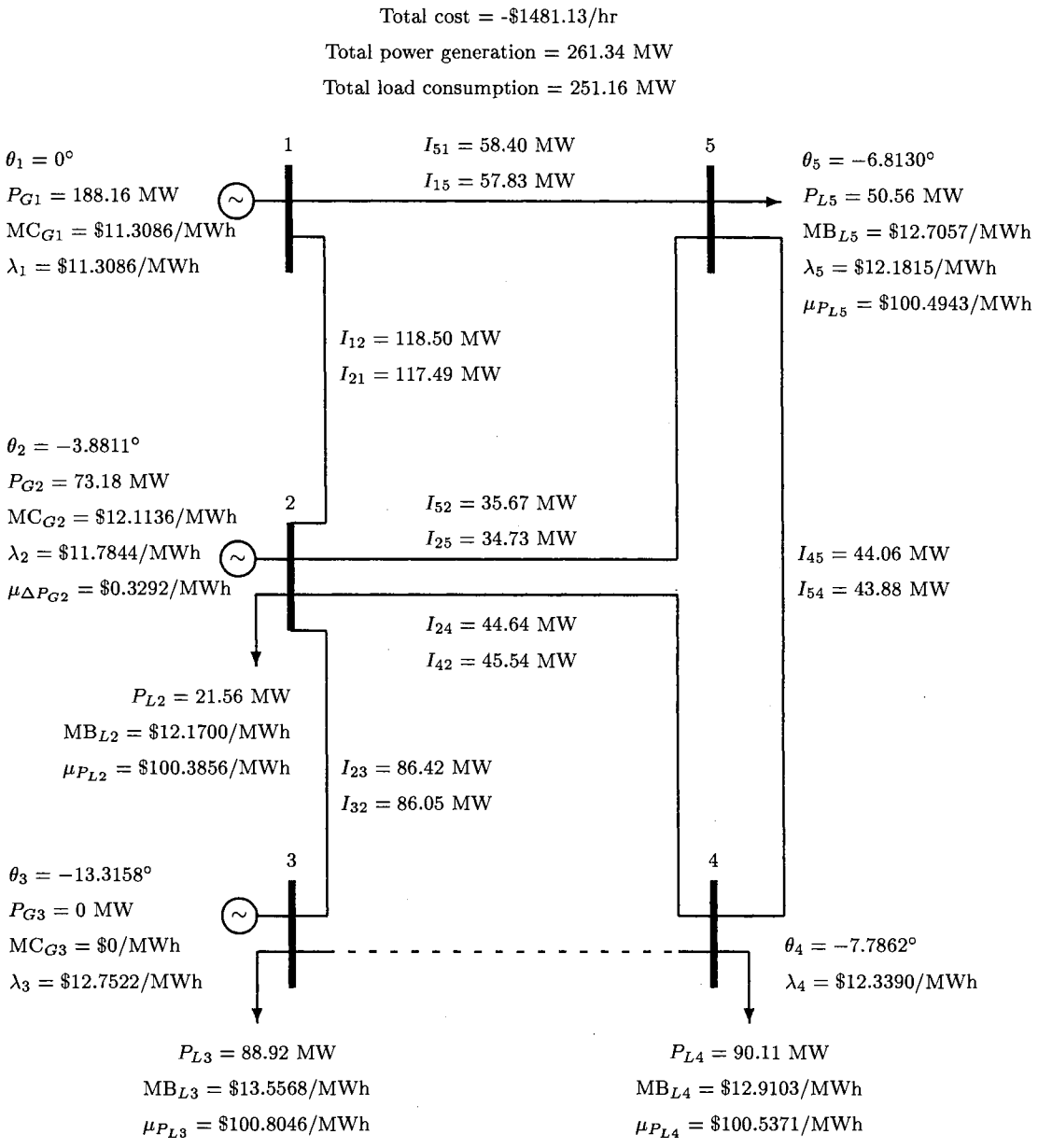

Figure 5.25a: Final (converged) solution of subproblem 4 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

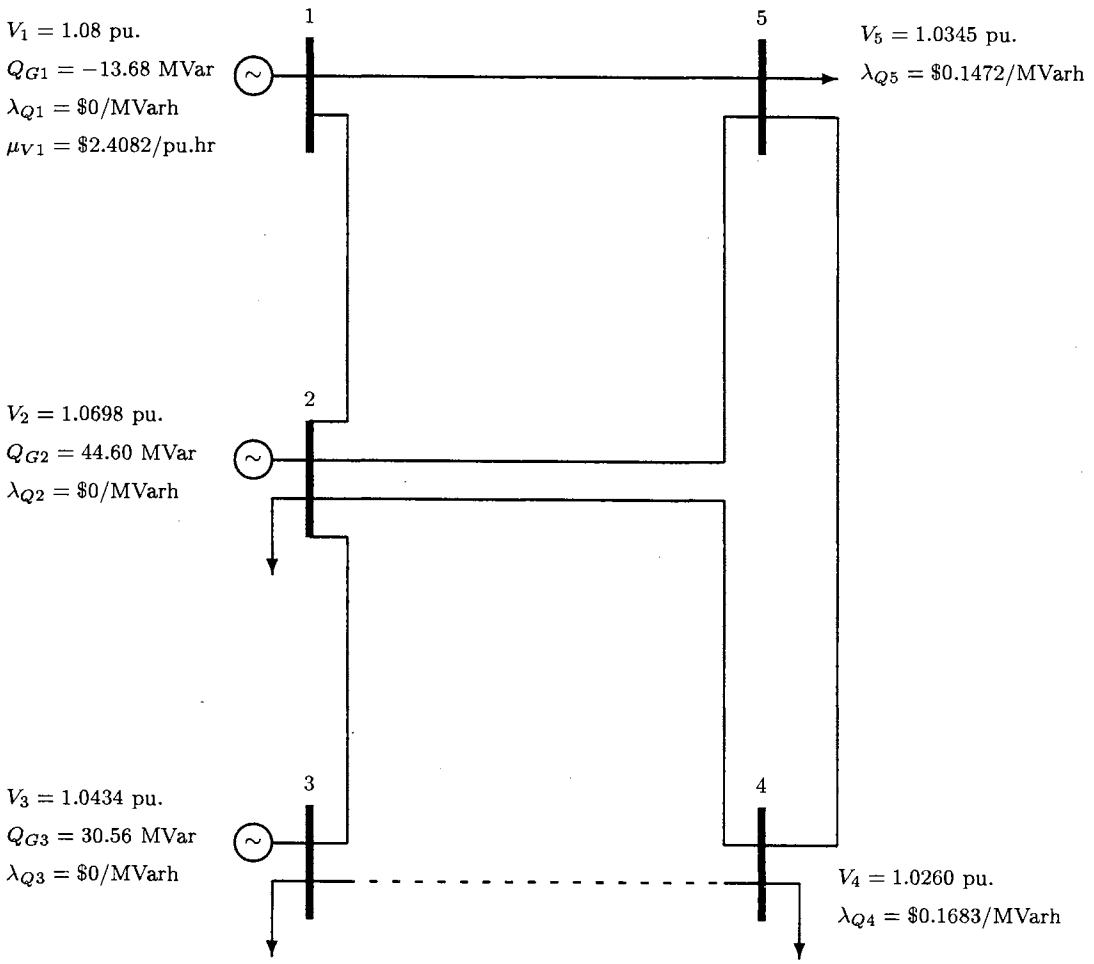

Figure 5.25b: Final (converged) solution of subproblem 4 of (AC) ESCOPF, showing reactive power and voltages.

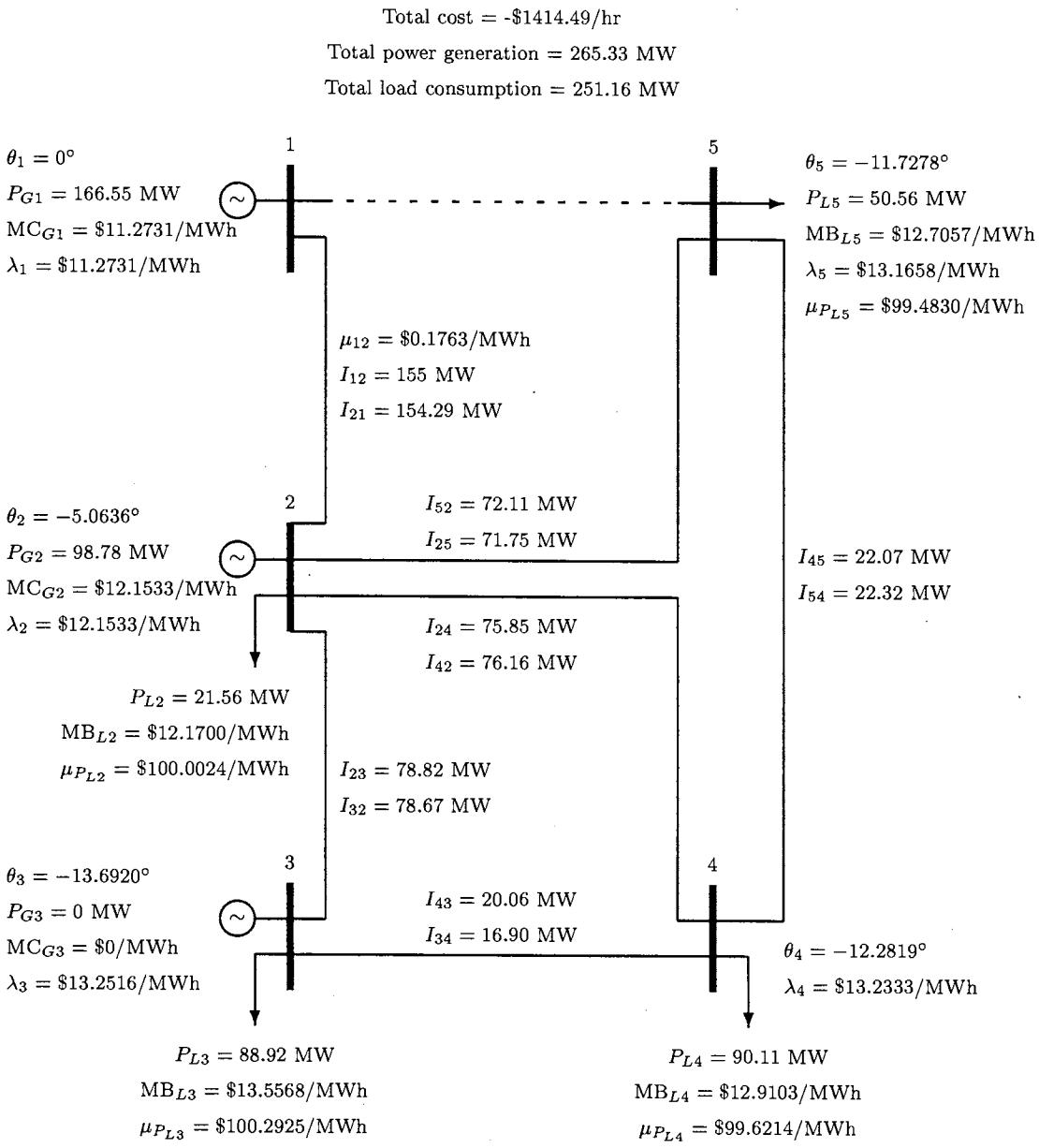

Figure 5.26a: Final (converged) solution of subproblem 5 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

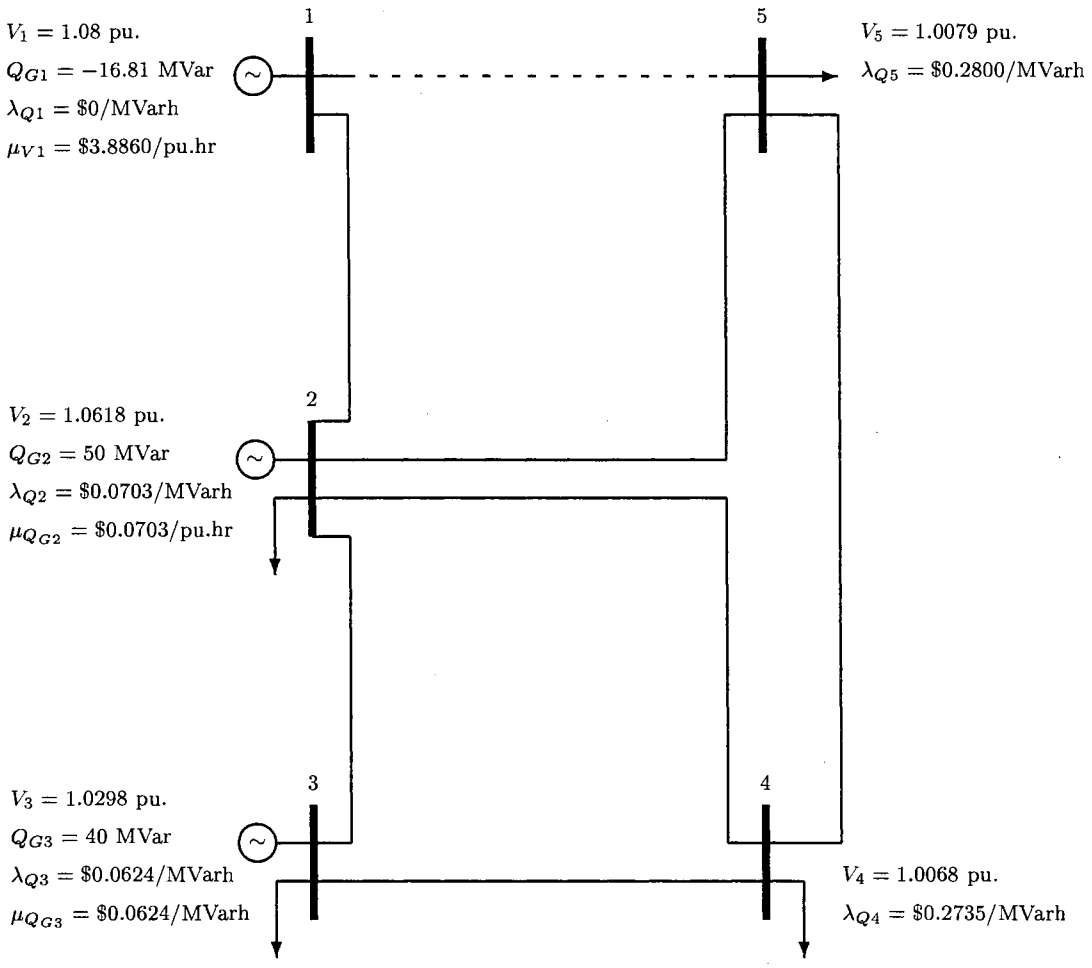

Figure 5.26b: Final (converged) solution of subproblem 5 of (AC) ESCOPF, showing reactive power and voltages.

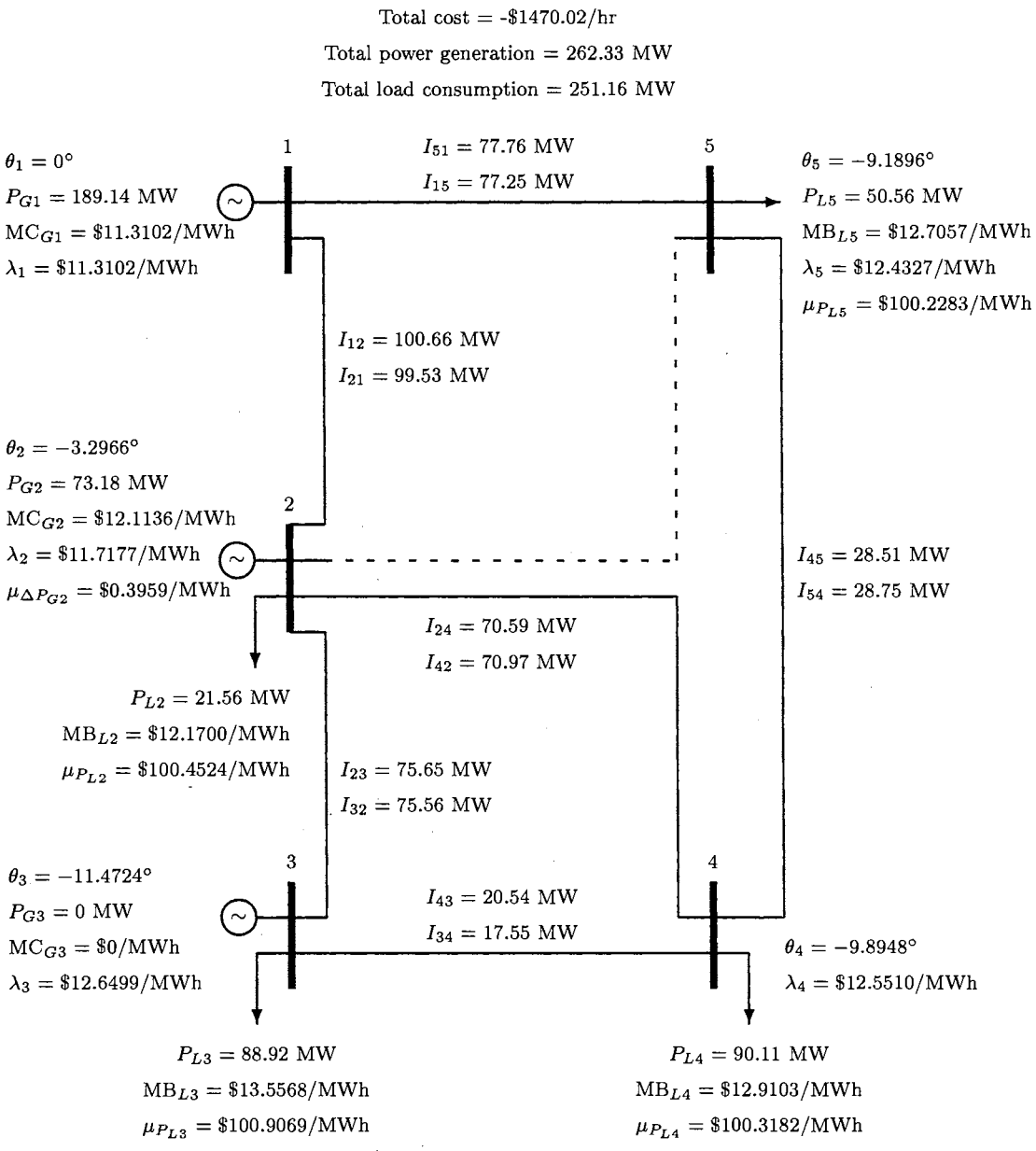

Figure 5.27a: Final (converged) solution of subproblem 6 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

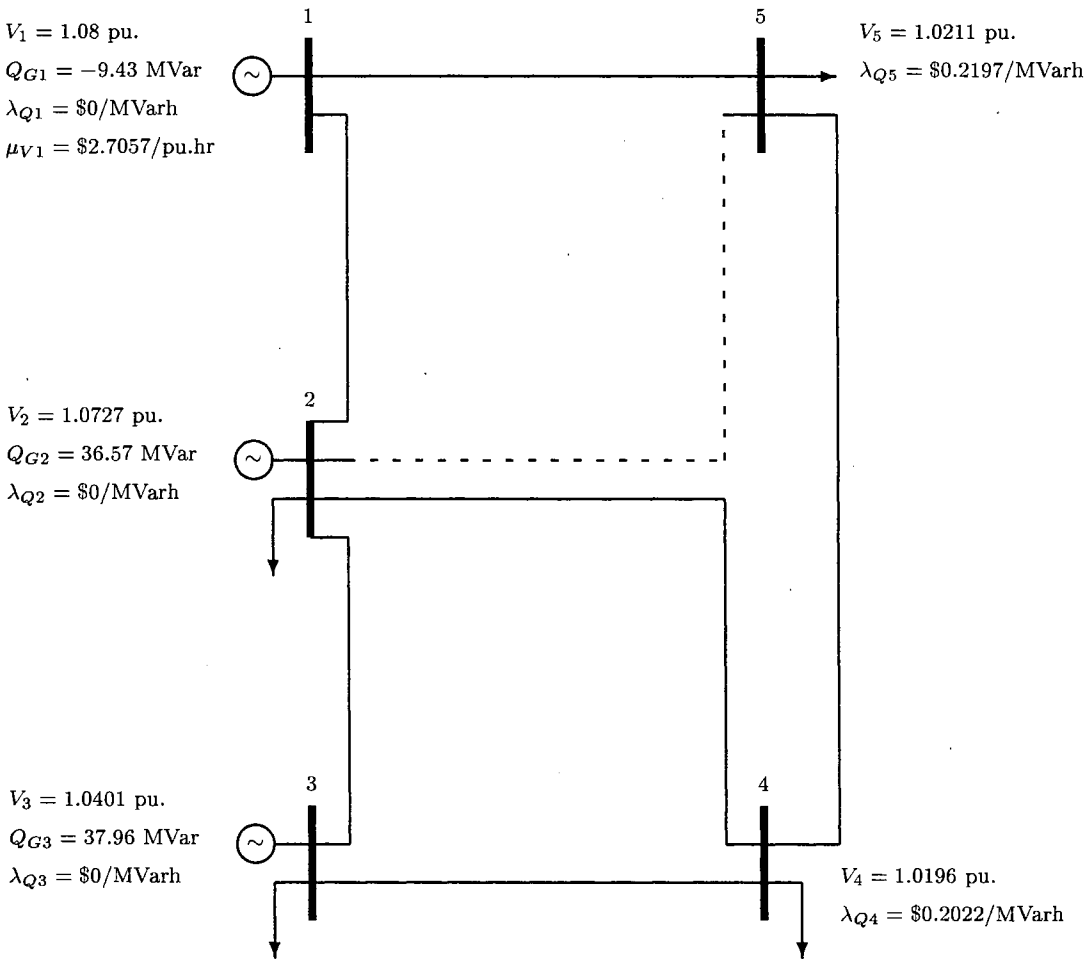

Figure 5.27b: Final (converged) solution of subproblem 6 of (AC) ESCOPF, showing reactive power and voltages.

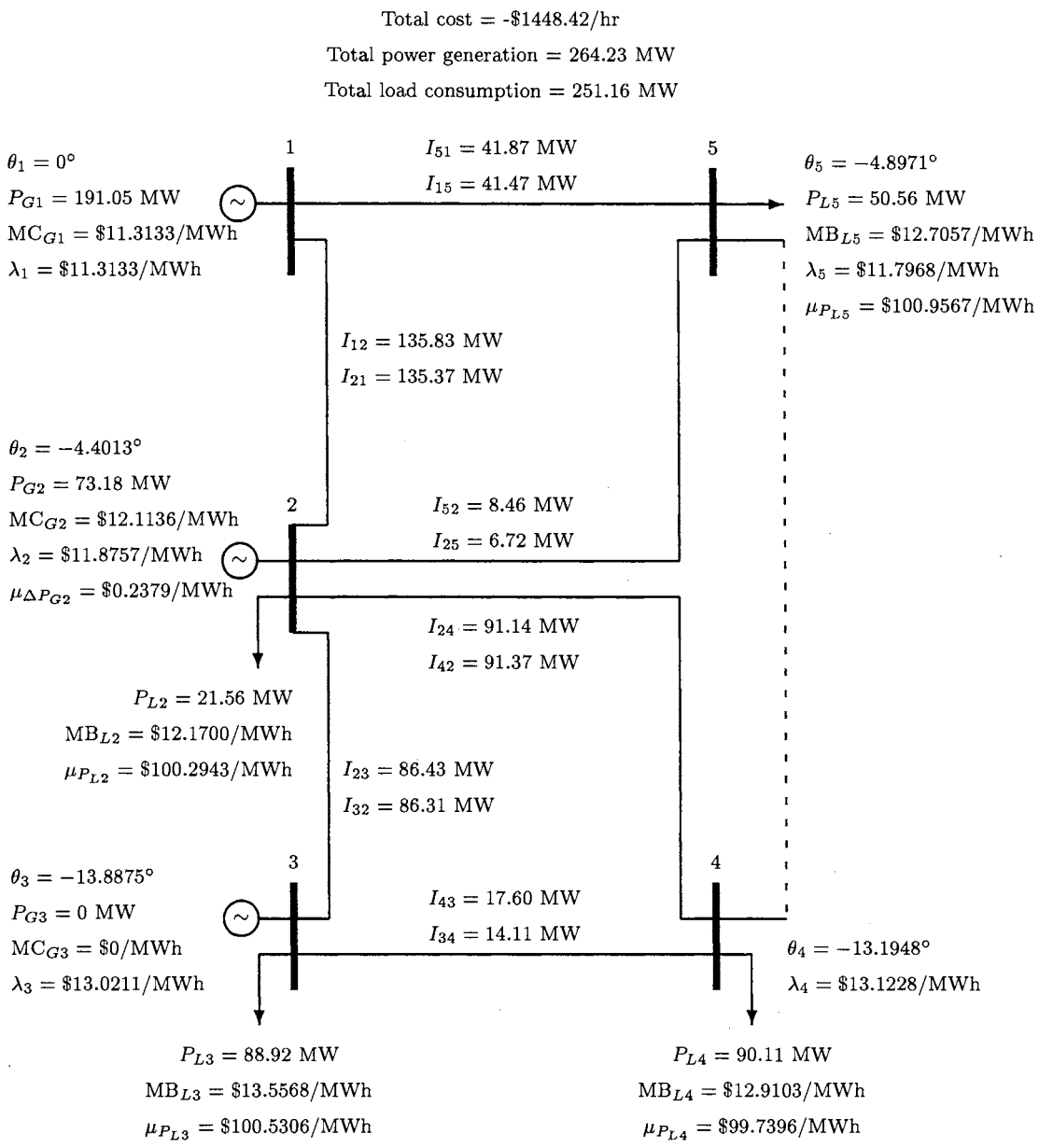

Figure 5.28a: Final (converged) solution of subproblem 7 of (AC) ESCOPF, showing real power, angles, line flows, and prices.

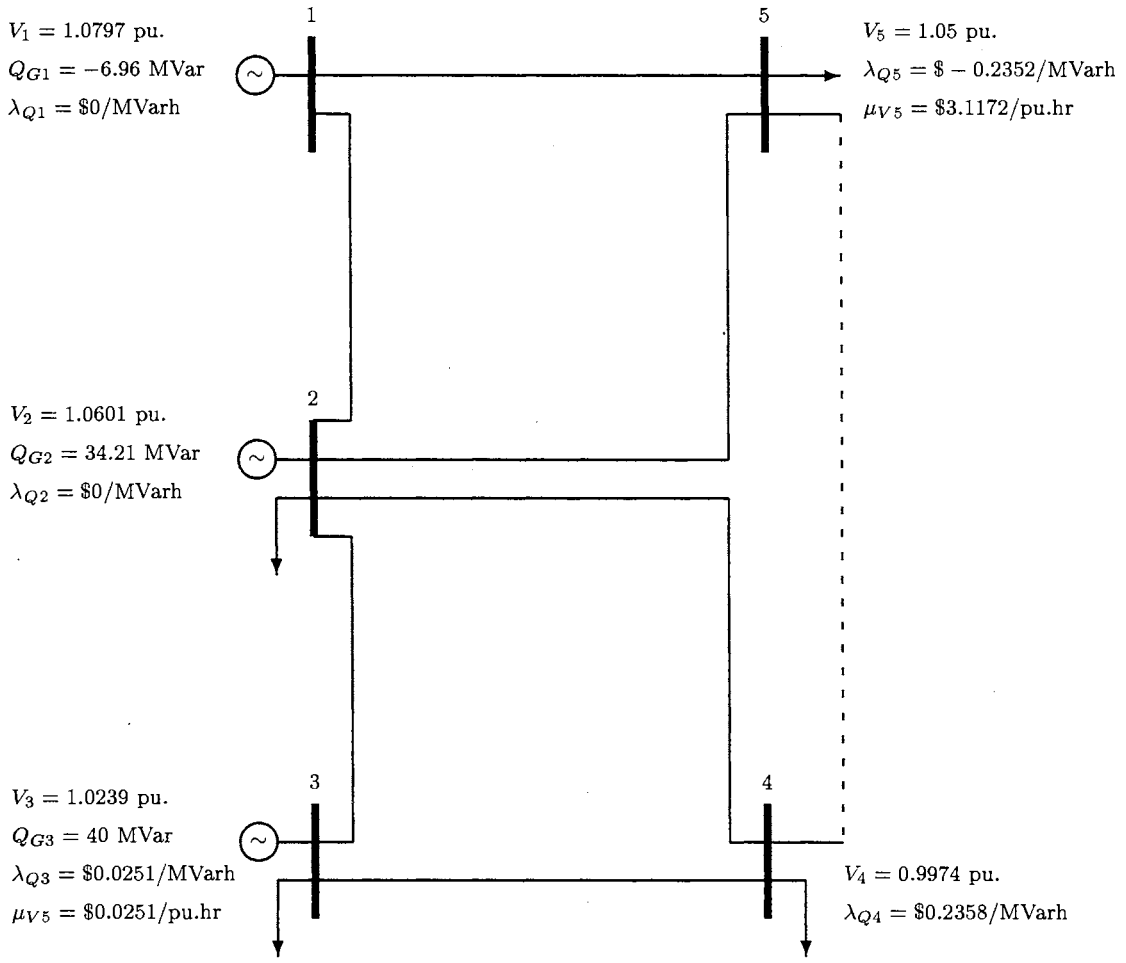

Figure 5.28b: Final (converged) solution of subproblem 7 of (AC) ESCOPF, showing reactive power and voltages.

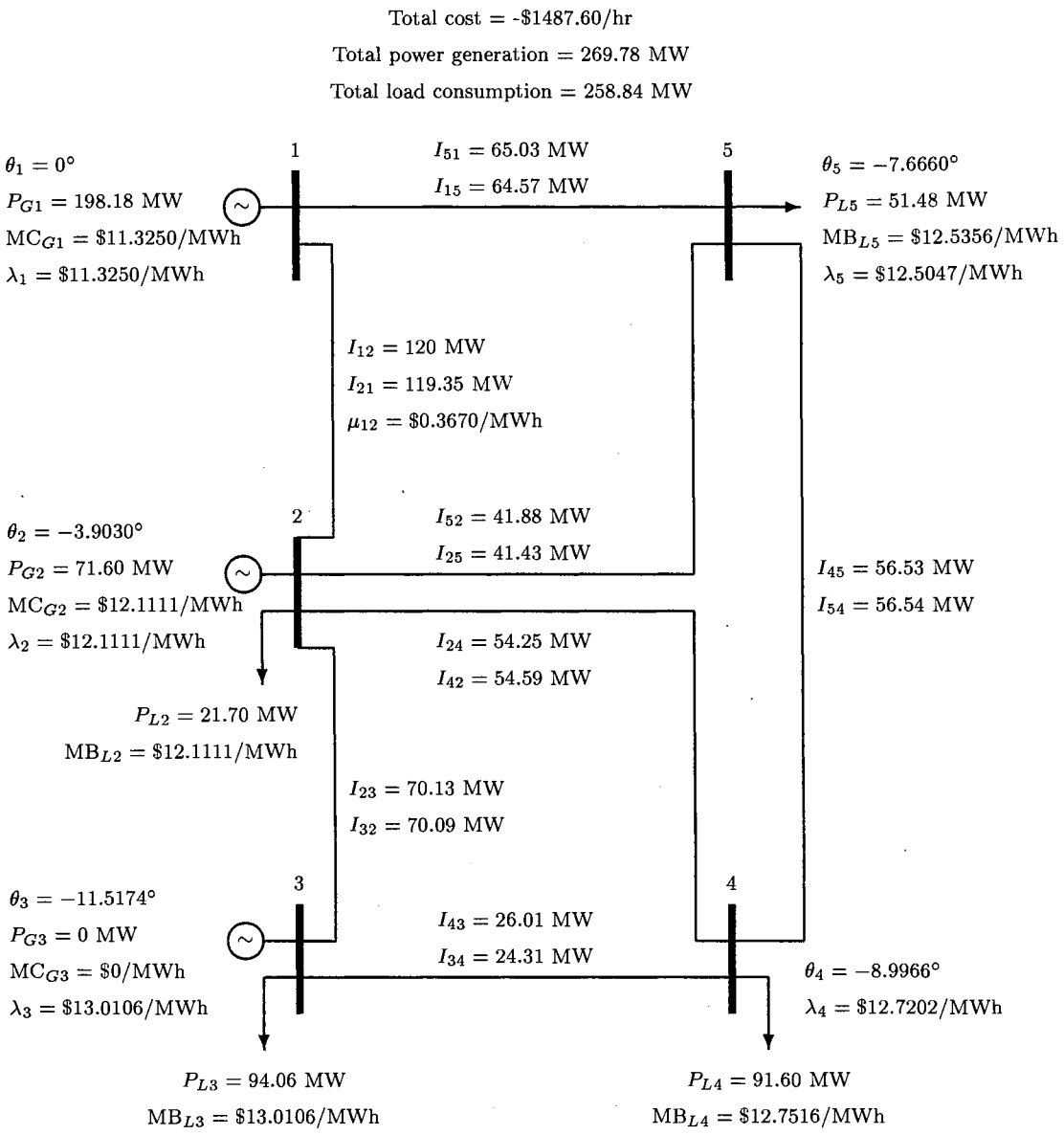

Figure 5.29a: Standard (AC) OPF of 5-bus system, showing real power, angles, line flows, and prices.

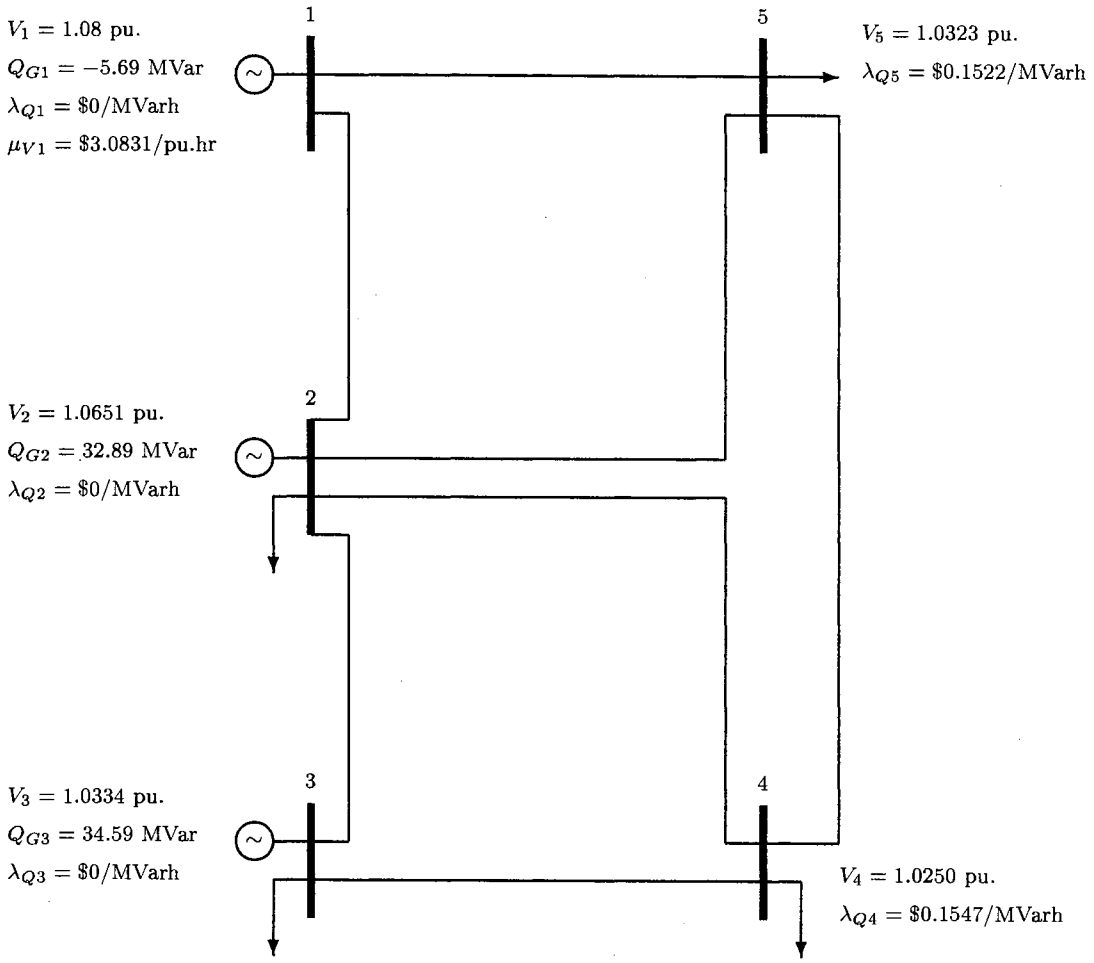

Figure 5.29b: Standard (AC) OPF of 5-bus system, showing reactive power and voltages.

5.3 IEEE 14-bus Case

The network of the IEEE 14-bus case is as shown in Figure 5.30, where its characteristics are summarized in Table 5.11. The system has two generators connected at buses 1 and 2, and three synchronous condensers connected at buses 3, 6 and 8. Loads are connected at buses 3, 4, 5, 6, 9, 10, 11, 12, 13, and 14. The line flow limits are shown in Figure 5.30, where the emergency limits are in the parentheses. Additional data of the IEEE 14-bus system is in Appendix B.

	Bus	$P_{Gi_{\min}}$	$P_{Gi_{\max}}$	$Q_{Gi_{\min}}$	$Q_{Gi_{\max}}$	$\Delta^{+}_{\max}P_{Gi}$	$\Delta^{-}_{\max}P_{Gi}$
Gen	1	45 MW	250 MW	-100 MVar	150 MVar	50 MW	50 MW
Gen	2	15 MW	150 MW	-40 MVar	50 MVar	35 MW	35 MW
SynCon	3	N/A	N/A	-40 MVar	40 MVar	N/A	N/A
SynCon	6	N/A	N/A	-100 MVar	104 MVar	N/A	N/A
SynCon	8	N/A	N/A	-100 MVar	104 MVar	N/A	N/A

Table 5.11: Characteristics of generators in IEEE14-bus system at pre-contingency state.

Although there are twenty lines in the 14-bus network, there are considered seven contingencies only on the high-voltage transmission lines. The credible contingencies and their probabilities π^k are shown in Table 5.12.

Contingency	Lines	π^k
1	1 - 2	0.01
2	2 - 3	0.01
3	2 - 4	0.01
4	3 - 4	0.01
5	1 - 5	0.01
6	2 - 5	0.01
7	4 - 5	0.01

Table 5.12: Credible contingencies of IEEE 14-bus system and their probabilities.

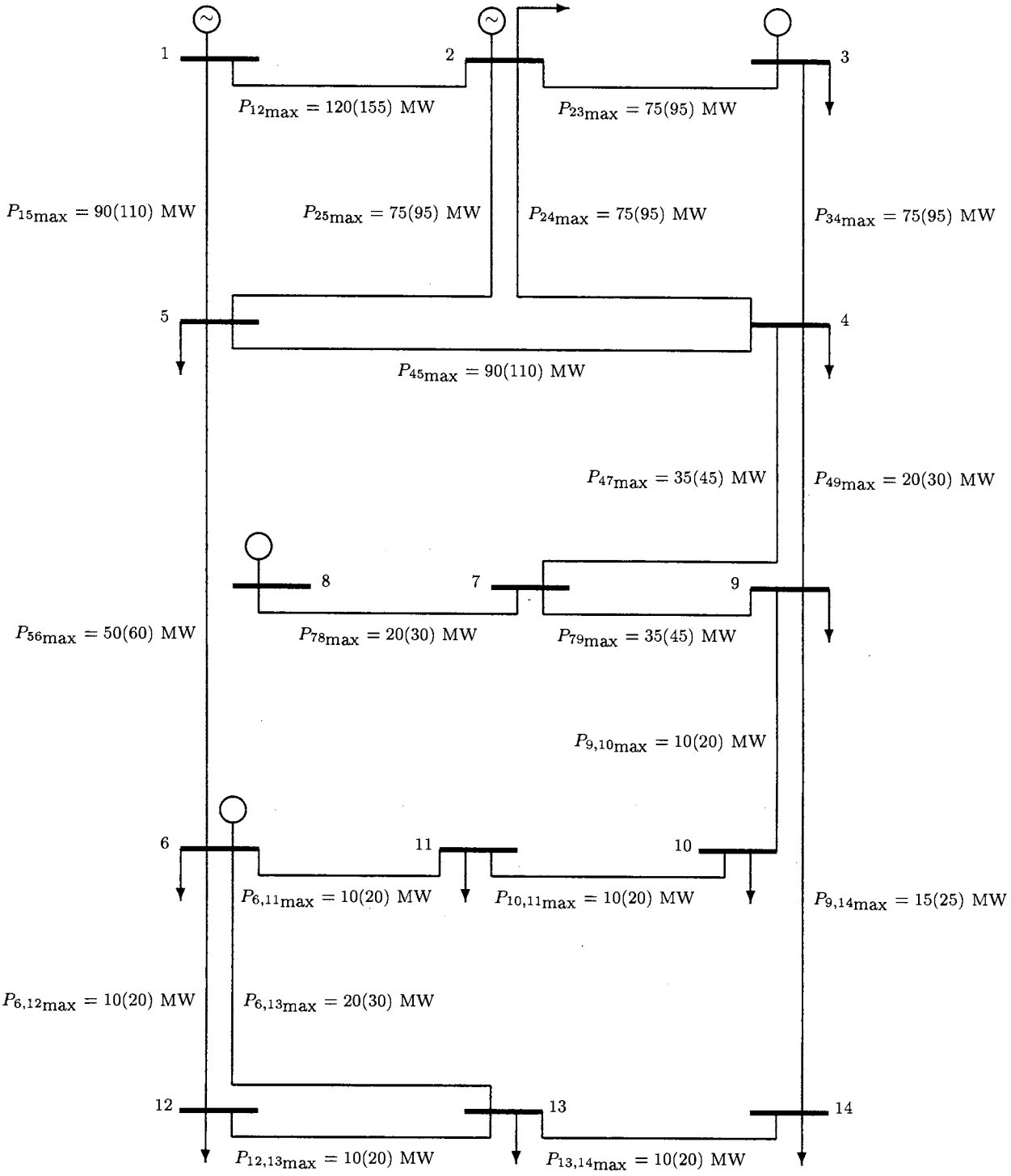

Figure 5.30: Characteristics of 14-bus system.

5.3.1 IEEE 14-bus Case: DC Approximation

The solutions of the IEEE 14-bus system can not be illustrated in the figures because there is not enough space for all the information. Thus, the solution of each subproblem is summarized in pair with the solution of the main problem in a table for easy understanding. Table 5.13 shows the results of contingency subproblem 1 compared to the results of the main problem. All the results of the IEEE 14-bus system are in Appendix C.

Since the 5-bus case is the truncated system of the IEEE 14-bus case, where the low-voltage lines are ignored, the solutions of the IEEE 14-bus are similar to the solutions of the 5-bus case as discussed in detail earlier. Therefore, We can skip the discussion of the results for the IEEE 14-bus case. Subproblem 1 of the IEEE 14-bus case behaves the same way as subproblem 1 of the 5-bus case does. The numbers in parentheses following the real load at contingency 1 in Table 5.13 are the interrupted load at each bus. The total amount of load interruption is 5.77 MW, which is not significant compared to the total load consumption. There is load interruption in other subproblems. Tables 5.14-5.19 show the results of subproblems 2 to 7 compared to the solution of the main problem (ESCOPF).

Table 5.20 compares the solution of the standard OPF and the ESCOPF for the DC 14-bus case. The results are similar to those of the 5-bus case shown in Table 5.8

	ESCOPF (DC)	Contingency 1 (DC)
$P_{G_{total}}$ (MW)	265.77	260.00
$P_{L_{total}}$ (MW)	265.77	260.00
Cost (\$/MWh)	-208.51	410.00
P_{G1} (MW)	150.77	110
P_{G2}	115	150
MC_{G1} (\$/MWh)	11.25	11.18
MC_{G2} (\$/MWh)	12.18	12.23
P_{L2} (MW)	21.34	20.55 (0.79)
P_{L3}	95	95
P_{L4}	49.86	48.19 (1.66)
P_{L5}	7.76	7.49 (0.27)
P_{L6}	11.46	11.07 (0.39)
P_{L9}	30.69	29.66 (1.03)
P_{L10}	9.41	9.10 (0.31)
P_{L11}	3.64	3.52 (0.12)
P_{L12}	6.38	6.17 (0.21)
P_{L13}	14.22	13.75 (0.47)
P_{L14}	16.00	15.50 (0.50)
P_{12} (MW)	88.21	-
P_{23}	74.93	65.90
P_{24}	59.99	41.21
P_{34}	-20.77	-29.10
P_{15}	62.56	110
P_{25}	46.95	22.34
P_{45}	-58.25	-81.34
P_{56}	43.50	43.50
P_{47}	30.70	28.76
P_{78}	0	0
P_{49}	17.61	16.50
P_{79}	30.70	28.76
$P_{9,10}$	6.68	5.57
$P_{6,11}$	6.37	7.05
$P_{10,11}$	-2.73	-3.53
$P_{6,12}$	7.63	7.48
$P_{6,13}$	18.04	17.91
$P_{12,13}$	1.25	1.31
$P_{9,14}$	10.94	10.03
$P_{13,14}$	5.07	5.47

Table 5.13: Solution of subproblem 1 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 2 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost (\$/MWh)	-208.51	-239.15
P_{G1} (MW)	150.77	185.77
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.25	11.30
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	101.63
P_{23}	74.93	-
P_{24}	59.99	91.36
P_{34}	-20.77	-95
P_{15}	62.56	84.14
P_{25}	46.95	68.93
P_{45}	-58.25	-99.43
P_{56}	43.50	45.88
P_{47}	30.70	29.18
P_{78}	0	0
P_{49}	17.61	16.75
P_{79}	30.70	29.18
$P_{9,10}$	6.68	5.22
$P_{6,11}$	6.37	7.83
$P_{10,11}$	-2.73	-4.19
$P_{6,12}$	7.63	7.79
$P_{6,13}$	18.04	18.79
$P_{12,13}$	1.25	1.41
$P_{9,14}$	10.94	10.01
$P_{13,14}$	5.07	5.99

Table 5.14: Solution of subproblem 2 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 3 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost (\$/MWh)	-208.51	-239.15
P_{G1} (MW)	150.77	185.77
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.25	11.30
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	101.25
P_{23}	74.93	90.39
P_{24}	59.99	-
P_{34}	-20.77	-4.61
P_{15}	62.56	84.52
P_{25}	46.95	68.51
P_{45}	-58.25	-100.34
P_{56}	43.50	45.93
P_{47}	30.70	29.15
P_{78}	0	0
P_{49}	17.61	16.73
P_{79}	30.70	29.15
$P_{9,10}$	6.68	5.19
$P_{6,11}$	6.37	7.86
$P_{10,11}$	-2.73	-4.22
$P_{6,12}$	7.63	7.80
$P_{6,13}$	18.04	18.81
$P_{12,13}$	1.25	1.41
$P_{9,14}$	10.94	9.99
$P_{13,14}$	5.07	6.01

Table 5.15: Solution of subproblem 3 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 4 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost (\$/MWh)	-208.51	-239.15
P_{G1} (MW)	150.77	185.77
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.25	11.30
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	121.89
P_{23}	74.93	95
P_{24}	59.99	48.42
P_{34}	-20.77	-
P_{15}	62.56	63.88
P_{25}	46.95	37.13
P_{45}	-58.25	-50.21
P_{56}	43.50	43.04
P_{47}	30.70	30.99
P_{78}	0	0
P_{49}	17.61	17.78
P_{79}	30.70	30.99
$P_{9,10}$	6.68	6.96
$P_{6,11}$	6.37	6.09
$P_{10,11}$	-2.73	-2.45
$P_{6,12}$	7.63	7.60
$P_{6,13}$	18.04	17.89
$P_{12,13}$	1.25	1.21
$P_{9,14}$	10.94	11.12
$P_{13,14}$	5.07	4.89

Table 5.16: Solution of subproblem 4 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 5 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost ($/MWh)	-208.51	-212.43
P_{G1} (MW)	150.77	155
P_{G2}	115	110.77
MC_{G1} ($/MWh)	11.25	11.25
MC_{G2} ($/MWh)	12.18	12.17
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	155
P_{23}	74.93	85.55
P_{24}	59.99	82.07
P_{34}	-20.77	-9.45
P_{15}	62.56	-
P_{25}	46.95	76.80
P_{45}	-58.25	-27.33
P_{56}	43.50	41.71
P_{47}	30.70	31.83
P_{78}	0	0
P_{49}	17.61	18.26
P_{79}	30.70	31.83
$P_{9,10}$	6.68	7.77
$P_{6,11}$	6.37	5.28
$P_{10,11}$	-2.73	-1.64
$P_{6,12}$	7.63	7.51
$P_{6,13}$	18.04	17.47
$P_{12,13}$	1.25	1.12
$P_{9,14}$	10.94	11.63
$P_{13,14}$	5.07	4.37

Table 5.17: Solution of subproblem 5 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 6 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost ($/MWh)	-208.51	-239.15
P_{G1} (MW)	150.77	155
P_{G2}	115	110.77
MC_{G1} ($/MWh)	11.25	11.30
MC_{G2} ($/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	101.15
P_{23}	74.93	83.01
P_{24}	59.99	76.80
P_{34}	-20.77	-11.99
P_{15}	62.56	84.61
P_{25}	46.95	-
P_{45}	-58.25	-34.71
P_{56}	43.50	42.14
P_{47}	30.70	31.56
P_{78}	0	0
P_{49}	17.61	18.11
P_{79}	30.70	31.56
$P_{9,10}$	6.68	7.51
$P_{6,11}$	6.37	5.54
$P_{10,11}$	-2.73	-1.90
$P_{6,12}$	7.63	7.54
$P_{6,13}$	18.04	17.60
$P_{12,13}$	1.25	1.15
$P_{9,14}$	10.94	11.46
$P_{13,14}$	5.07	4.54

Table 5.18: Solution of subproblem 6 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESCOPF (DC)	Contingency 7 (DC)
$P_{G_{total}}$ (MW)	265.77	265.77
$P_{L_{total}}$ (MW)	265.77	265.77
Cost (\$/MWh)	-208.51	-239.15
P_{G1} (MW)	150.77	185.77
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.25	11.30
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.34
P_{L3}	95	95
P_{L4}	49.86	49.86
P_{L5}	7.76	7.76
P_{L6}	11.46	11.46
P_{L9}	30.69	30.69
P_{L10}	9.41	9.41
P_{L11}	3.64	3.64
P_{L12}	6.38	6.38
P_{L13}	14.22	14.22
P_{L14}	16.00	16.00
P_{12} (MW)	88.21	135.53
P_{23}	74.93	89.07
P_{24}	59.99	89.39
P_{34}	-20.77	-5.93
P_{15}	62.56	50.24
P_{25}	46.95	15.73
P_{45}	-58.25	-
P_{56}	43.50	58.20
P_{47}	30.70	21.35
P_{78}	0	0
P_{49}	17.61	12.25
P_{79}	30.70	21.35
$P_{9,10}$	6.68	-2.32
$P_{6,11}$	6.37	15.38
$P_{10,11}$	-2.73	-11.74
$P_{6,12}$	7.63	8.64
$P_{6,13}$	18.04	22.72
$P_{12,13}$	1.25	2.26
$P_{9,14}$	10.94	5.24
$P_{13,14}$	5.07	10.77

Table 5.19: Solution of subproblem 7 compared to the optimal pre-contingency state of (DC) IEEE 14-bus system.

	ESOPF (DC)	OPF (DC)
$P_{G_{total}}$ (MW)	265.77	269.59
$P_{L_{total}}$ (MW)	265.77	269.59
Cost (\$/MWh)	-208.51	-243.42
Expected Security Cost (\$/MWh)	-191.25	N/A
P_{G1} (MW)	150.77	189.53
P_{G2}	115	80.06
MC_{G1} (\$/MWh)	11.25	11.31
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	21.34	21.68
P_{L3}	95	96.45
P_{L4}	49.86	50.34
P_{L5}	7.76	7.93
P_{L6}	11.46	11.68
P_{L9}	30.69	31.08
P_{L10}	9.41	9.54
P_{L11}	3.64	3.70
P_{L12}	6.38	6.49
P_{L13}	14.22	14.45
P_{L14}	16.00	16.22
P_{12} (MW)	88.21	120
P_{23}	74.93	75
P_{24}	59.99	58.66
P_{34}	-20.77	-21.45
P_{15}	62.56	69.53
P_{25}	46.95	44.71
P_{45}	-58.25	-61.96
P_{56}	43.50	44.34
P_{47}	30.70	31.03
P_{78}	0	0
P_{49}	17.61	17.80
P_{79}	30.70	31.03
$P_{9,10}$	6.68	6.70
$P_{6,11}$	6.37	6.54
$P_{10,11}$	-2.73	-2.85
$P_{6,12}$	7.63	7.76
$P_{6,13}$	18.04	18.36
$P_{12,13}$	1.25	1.27
$P_{9,14}$	10.94	11.05
$P_{13,14}$	5.07	5.18

Table 5.20: Solutions of standard OPF and ESCOPF of (DC) IEEE 14-bus system.

5.3.2 IEEE 14-bus Case: AC System

Tables 5.21-5.27 show the comparison of the optimal pre-contingency state solution of the AC IEEE 14-bus case and the solution of each subproblem. The overall results of the actual IEEE 14-bus case are in Appendix C.

	ESCOPF (AC)	Contingency 1 (AC)
$P_{G_{total}}$ (MW)	270.89	268.76
$P_{L_{total}}$ (MW)	241.83	237.77
Cost (\$/MWh)	159.31	631.31
P_{G1} (MW)	155.89	118.76
P_{G2}	115	150
MC_{G1} (\$/MWh)	11.26	11.19
MC_{G2} (\$/MWh)	12.18	12.23
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	85.45 (4.06)
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	-
I_{23}/I_{32}	69.07/69.11	58.83/58.97
I_{24}/I_{42}	57.97/57.87	40.67/40.63
I_{34}/I_{43}	17.75/18.77	23.96/24.63
I_{15}/I_{51}	59.45/59.36	110/109.39
I_{25}/I_{52}	45.84/45.53	22.06/22.23
I_{45}/I_{54}	56.56/56.31	79.48/79.56
I_{56}/I_{65}	49.60/49.60	49.73/49.73
I_{47}/I_{74}	34.51/34.51	35.40/35.40
I_{78}/I_{87}	4.31 / 4.31	24.37/24.37
I_{49}/I_{94}	19.66/19.66	19.56/19.56
I_{79}/I_{97}	35.00/35.00	37.00/37.00
$I_{9,10}/I_{10,9}$	3.01/ 3.01	2.71/ 2.71
$I_{6,11}/I_{11,6}$	10.00/10.00	9.08/ 9.08
$I_{10,11}/I_{11,10}$	6.94/ 6.94	5.94/ 5.94
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.28/ 7.28
$I_{6,13}/I_{13,6}$	17.98/17.98	17.47/17.47
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.66/ 1.66
$I_{9,14}/I_{14,9}$	6.87/ 6.87	7.28/ 7.28
$I_{13,14}/I_{14,13}$	7.32/ 7.32	6.63/ 6.63

Table 5.21: Solution of subproblem 1 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 2 (AC)
$P_{G_{total}}$ (MW)	270.89	282.61
$P_{L_{total}}$ (MW)	241.83	241.83
Cost (\$/MWh)	159.31	261.68
P_{G1} (MW)	155.89	202.61
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.26	11.33
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	109.35/107.69
I_{23}/I_{32}	69.07/69.11	-
I_{24}/I_{42}	57.97/57.87	89.90/89.66
I_{34}/I_{43}	17.75/18.77	91.61/92.04
I_{15}/I_{51}	59.45/59.36	82.81/ 82.40
I_{25}/I_{52}	45.84/45.53	68.03/67.73
I_{45}/I_{54}	56.56/56.31	97.49/97.46
I_{56}/I_{65}	49.60/49.60	54.42/54.42
I_{47}/I_{74}	34.51/34.51	35.12/35.12
I_{78}/I_{87}	4.31 / 4.31	24.43/24.43
I_{49}/I_{94}	19.66/19.66	19.15/19.15
I_{79}/I_{97}	35.00/35.00	34.95/34.95
$I_{9,10}/I_{10,9}$	3.01/ 3.01	1.00/ 1.00
$I_{6,11}/I_{11,6}$	10.00/10.00	10.51/10.51
$I_{10,11}/I_{11,10}$	6.94/ 6.94	7.37/ 7.37
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.43/ 7.43
$I_{6,13}/I_{13,6}$	17.98/17.98	18.07/18.07
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.87/ 1.87
$I_{9,14}/I_{14,9}$	6.87/ 6.87	6.01/ 6.01
$I_{13,14}/I_{14,13}$	7.32/ 7.32	7.61/ 7.61

Table 5.22: Solution of subproblem 2 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 3 (AC)
$P_{G_{total}}$ (MW)	270.89	275.10
$P_{L_{total}}$ (MW)	241.83	241.83
Cost (\$/MWh)	159.31	176.63
P_{G1} (MW)	155.89	195.10
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.26	11.32
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	101.48/100.23
I_{23}/I_{32}	69.07/69.11	84.67/84.45
I_{24}/I_{42}	57.97/57.87	-
I_{34}/I_{43}	17.75/18.77	2.09/ 4.90
I_{15}/I_{51}	59.45/59.36	81.64/81.24
I_{25}/I_{52}	45.84/45.53	68.58/68.14
I_{45}/I_{54}	56.56/56.31	97.04/97.10
I_{56}/I_{65}	49.60/49.60	51.68/51.68
I_{47}/I_{74}	34.51/34.51	34.96/34.96
I_{78}/I_{87}	4.31 / 4.31	24.80/24.80
I_{49}/I_{94}	19.66/19.66	19.19/19.19
I_{79}/I_{97}	35.00/35.00	36.05/36.05
$I_{9,10}/I_{10,9}$	3.01/ 3.01	1.74/ 1.74
$I_{6,11}/I_{11,6}$	10.00/10.00	9.85/ 9.85
$I_{10,11}/I_{11,10}$	6.94/ 6.94	6.70/ 6.70
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.38/ 7.38
$I_{6,13}/I_{13,6}$	17.98/17.98	17.85/17.85
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.77/ 1.77
$I_{9,14}/I_{14,9}$	6.87/ 6.87	6.65/ 6.65
$I_{13,14}/I_{14,13}$	7.32/ 7.32	7.15/ 7.15

Table 5.23: Solution of subproblem 3 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 4 (AC)
$P_{G_{total}}$ (MW)	270.89	272.44
$P_{L_{total}}$ (MW)	241.83	241.83
Cost (\$/MWh)	159.31	146.52
P_{G1} (MW)	155.89	192.44
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.26	11.32
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	118.30/117.48
I_{23}/I_{32}	69.07/69.11	87.23/86.85
I_{24}/I_{42}	57.97/57.87	47.84/47.81
I_{34}/I_{43}	17.75/18.77	-
I_{15}/I_{51}	59.45/59.36	60.96/ 60.84
I_{25}/I_{52}	45.84/45.53	36.71/36.51
I_{45}/I_{54}	56.56/56.31	50.55/50.36
I_{56}/I_{65}	49.60/49.60	45.62/45.62
I_{47}/I_{74}	34.51/34.51	35.79/35.79
I_{78}/I_{87}	4.31 / 4.31	18.76/18.76
I_{49}/I_{94}	19.66/19.66	20.23/20.23
I_{79}/I_{97}	35.00/35.00	38.05/38.05
$I_{9,10}/I_{10,9}$	3.01/ 3.01	3.99/ 3.99
$I_{6,11}/I_{11,6}$	10.00/10.00	7.88/ 7.88
$I_{10,11}/I_{11,10}$	6.94/ 6.94	4.75/ 4.75
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.21/ 7.21
$I_{6,13}/I_{13,6}$	17.98/17.98	17.02/17.02
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.50/ 1.50
$I_{9,14}/I_{14,9}$	6.87/ 6.87	8.24/ 8.24
$I_{13,14}/I_{14,13}$	7.32/ 7.32	5.85/ 5.85

Table 5.24: Solution of subproblem 4 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 5 (AC)
$P_{G_{total}}$ (MW)	270.89	276.66
$P_{L_{total}}$ (MW)	241.83	241.83
Cost (\$/MWh)	159.31	221.51
P_{G1} (MW)	155.89	164.71
P_{G2}	115	111.95
MC_{G1} (\$/MWh)	11.26	11.27
MC_{G2} (\$/MWh)	12.18	12.17
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	155 /153.84
I_{23}/I_{32}	69.07/69.11	79.92/79.74
I_{24}/I_{42}	57.97/57.87	80.27/79.90
I_{34}/I_{43}	17.75/18.77	7.11/ 8.86
I_{15}/I_{51}	59.45/59.36	-
I_{25}/I_{52}	45.84/45.53	75.16/74.75
I_{45}/I_{54}	56.56/56.31	27.80/27.57
I_{56}/I_{65}	49.60/49.60	46.48/46.48
I_{47}/I_{74}	34.51/34.51	37.74/37.74
I_{78}/I_{87}	4.31 / 4.31	26.52/26.52
I_{49}/I_{94}	19.66/19.66	20.78/20.78
I_{79}/I_{97}	35.00/35.00	39.38/39.38
$I_{9,10}/I_{10,9}$	3.01/ 3.01	4.57/ 4.57
$I_{6,11}/I_{11,6}$	10.00/10.00	7.11/ 7.11
$I_{10,11}/I_{11,10}$	6.94/ 6.94	4.00/ 4.00
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.08/ 7.08
$I_{6,13}/I_{13,6}$	17.98/17.98	16.56/16.56
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.40/ 1.40
$I_{9,14}/I_{14,9}$	6.87/ 6.87	8.62/ 8.62
$I_{13,14}/I_{14,13}$	7.32/ 7.32	5.35/ 5.35

Table 5.25: Solution of subproblem 5 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 6 (AC)
$P_{G_{total}}$ (MW)	270.89	273.96
$P_{L_{total}}$ (MW)	241.83	237.77
Cost (\$/MWh)	159.31	163.78
P_{G1} (MW)	155.89	193.96
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.26	11.32
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	100.39/99.20
I_{23}/I_{32}	69.07/69.11	77.18/77.01
I_{24}/I_{42}	57.97/57.87	75.21/74.82
I_{34}/I_{43}	17.75/18.77	9.22/10.43
I_{15}/I_{51}	59.45/59.36	81.43/81.03
I_{25}/I_{52}	45.84/45.53	-
I_{45}/I_{54}	56.56/56.31	34.15/33.92
I_{56}/I_{65}	49.60/49.60	46.89/46.89
I_{47}/I_{74}	34.51/34.51	36.99/36.99
I_{78}/I_{87}	4.31 / 4.31	24.13/24.13
I_{49}/I_{94}	19.66/19.66	20.52/20.52
I_{79}/I_{97}	35.00/35.00	38.55/38.55
$I_{9,10}/I_{10,9}$	3.01/ 3.01	4.01/ 4.01
$I_{6,11}/I_{11,6}$	10.00/10.00	7.44/ 7.44
$I_{10,11}/I_{11,10}$	6.94/ 6.94	4.30/ 4.30
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.09/ 7.09
$I_{6,13}/I_{13,6}$	17.98/17.98	16.66/16.66
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.45/ 1.45
$I_{9,14}/I_{14,9}$	6.87/ 6.87	8.24/ 8.24
$I_{13,14}/I_{14,13}$	7.32/ 7.32	5.58/ 5.58

Table 5.26: Solution of subproblem 6 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	Contingency 7 (AC)
$P_{G_{total}}$ (MW)	270.89	273.97
$P_{L_{total}}$ (MW)	241.83	241.83
Cost (\$/MWh)	159.31	163.83
P_{G1} (MW)	155.89	193.97
P_{G2}	115	80
MC_{G1} (\$/MWh)	11.26	11.32
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	20.68
P_{L3}	89.51	89.51
P_{L4}	45	45
P_{L5}	7.14	7.14
P_{L6}	10.38	10.38
P_{L9}	26.23	26.23
P_{L10}	8.07	8.07
P_{L11}	3.19	3.19
P_{L12}	5.65	5.65
P_{L13}	12.47	12.47
P_{L14}	13.50	13.50
I_{12}/I_{21} (MW)	85.51/84.88	133.25/132.91
I_{23}/I_{32}	69.07/69.11	85.58/85.49
I_{24}/I_{42}	57.97/57.87	89.46/89.42
I_{34}/I_{43}	17.75/18.77	7.75/11.10
I_{15}/I_{51}	59.45/59.36	47.14/47.17
I_{25}/I_{52}	45.84/45.53	14.20/13.79
I_{45}/I_{54}	56.56/56.31	-
I_{56}/I_{65}	49.60/49.60	58.11/58.11
I_{47}/I_{74}	34.51/34.51	27.74/27.74
I_{78}/I_{87}	4.31 / 4.31	23.53/23.53
I_{49}/I_{94}	19.66/19.66	15.41/15.41
I_{79}/I_{97}	35.00/35.00	32.00/32.00
$I_{9,10}/I_{10,9}$	3.01/ 3.01	6.97/ 6.97
$I_{6,11}/I_{11,6}$	10.00/10.00	16.85/16.85
$I_{10,11}/I_{11,10}$	6.94/ 6.94	13.73/13.73
$I_{6,12}/I_{12,6}$	7.49/ 7.49	8.44/ 8.44
$I_{6,13}/I_{13,6}$	17.98/17.98	21.81/21.81
$I_{12,13}/I_{13,12}$	1.82/ 1.82	2.67/ 2.67
$I_{9,14}/I_{14,9}$	6.87/ 6.87	5.33/ 5.33
$I_{13,14}/I_{14,13}$	7.32/ 7.32	11.62/11.62

Table 5.27: Solution of subproblem 7 compared to the optimal pre-contingency state of (AC) IEEE 14-bus system.

	ESCOPF (AC)	OPF (AC)
$P_{G_{total}}$ (MW)	270.89	281.17
$P_{L_{total}}$ (MW)	241.83	249.94
Cost (\$/MWh)	159.31	139.65
Expected Security Cost (\$/MWh)	-203.90	N/A
P_{G1} (MW)	155.89	201.23
P_{G2}	115	79.94
MC_{G1} (\$/MWh)	11.26	11.33
MC_{G2} (\$/MWh)	12.18	12.12
P_{L2} (MW)	20.68	21.68
P_{L3}	89.51	93.93
P_{L4}	45	47.39
P_{L5}	7.14	7.53
P_{L6}	10.38	10.41
P_{L9}	26.23	26.12
P_{L10}	8.07	8.04
P_{L11}	3.19	3.19
P_{L12}	5.65	5.67
P_{L13}	12.47	12.49
P_{L14}	13.50	13.48
I_{12}/I_{21} (MW)	85.51/84.88	-
I_{23}/I_{32}	69.07/69.11	58.83/58.97
I_{24}/I_{42}	57.97/57.87	40.67/40.63
I_{34}/I_{43}	17.75/18.77	23.96/24.63
I_{15}/I_{51}	59.45/59.36	110/109.39
I_{25}/I_{52}	45.84/45.53	22.06/22.23
I_{45}/I_{54}	56.56/56.31	-
I_{56}/I_{65}	49.60/49.60	49.73/49.73
I_{47}/I_{74}	34.51/34.51	35.40/35.40
I_{78}/I_{87}	4.31 / 4.31	24.37/24.37
I_{49}/I_{94}	19.66/19.66	19.56/19.56
I_{79}/I_{97}	35.00/35.00	37.00/37.00
$I_{9,10}/I_{10,9}$	3.01/ 3.01	2.71/ 2.71
$I_{6,11}/I_{11,6}$	10.00/10.00	9.08/ 9.08
$I_{10,11}/I_{11,10}$	6.94/ 6.94	5.94/ 5.94
$I_{6,12}/I_{12,6}$	7.49/ 7.49	7.28/ 7.28
$I_{6,13}/I_{13,6}$	17.98/17.98	17.47/17.47
$I_{12,13}/I_{13,12}$	1.82/ 1.82	1.66/ 1.66
$I_{9,14}/I_{14,9}$	6.87/ 6.87	7.28/ 7.28
$I_{13,14}/I_{14,13}$	7.32/ 7.32	6.63/ 6.63

Table 5.28: Solutions of standard OPF and ESCOPF of (AC) IEEE 14-bus system.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In order to maintain the security of a power system, the proper amount of spinning reserves and transmission reserves should be maintained during the normal operating state. At a post-contingency state, the system with enough spinning and transmission reserves can be rescheduled so that it is operating within post-contingency operational limits. Recent studies have introduced the *security-constrained optimal power flow* (SCOPF) to include the *security constraints* of the post-contingency states into the *standard optimal power flow* (OPF) problem, with and without the capabilities of *post-contingency rescheduling.* Power system security and methods for optimizing power systems operation were reviewed in Chapter 2.

In this research, a new optimal power flow model which takes into account system security, called the *expected security-cost optimal power flow* (ESCOPF), is presented. The ESCOPF model handles security as an economic cost instead of as a constraint as has been done in the SCOPF. The *security cost* of a contingency is the minimum cost of the system operating within operational limits at the post-contingency state, given the system has been operating at a particular pre-contingency

state. We choose to incorporate the post-contingency state security costs in our objective function as expected costs because we do not know which contingency will occur, only the probability of its occurrence. The ESCOPF model also takes into account post-contingency rescheduling: generator rescheduling and load interruption. There are costs associated with post-contingency rescheduling, including generator maintenance costs and customer compensation for interrupted load.

In the beginning, we applied the "DC" approximation to the ESCOPF model because it simplifies the problem by linearizing the constraints. Furthermore, it eliminates reactive power flow and real power losses in transmission lines. After the model was successfully implemented for the DC cases, we applied it to actual power systems or "AC".

Two methods were used to solve the ESCOPF problem: the *decoupled* method and the *integrated* method. The decoupled method was introduced because the ESCOPF problem could be decoupled into a main problem and several subproblems with a connection between the two types of problems.

Initially, the Newton method was applied to the decoupled problem. But there was a "cycling" behavior in which a small number of constraints repeatedly entered and left the active set of binding constraints. We then explored the application of the primal-dual interior-point (PDIP) method to the decoupled problem. In our initial implementation, in which $\nu^k \to 0$ for each subproblem, the PDIP actually identified the set of active constraints in each subproblem. Therefore, the cycling behavior of the binding constraints still occurs when the PDIP is applied to the decoupled problem. We believe that the decoupled approach could be successfully used if the iteration

process were adjusted so that the PDIP does not involve identifying the active set before the entire problem converges. However, we were more concerned that the PDIP might not work in general with our ESCOPF model. Therefore, we applied the PDIP to the integrated problem, which combines the main and subproblems together, just to see if it worked. While the PDIP was failed to solve the 5-bus and 14-bus systems with the decoupled method, the corresponding integrated problems of those systems were successfully solved using PDIP.

The ESCOPF model has been tested on the DC 3-bus system, DC and AC 5-bus systems, and DC and AC IEEE 14-bus systems. The results of the DC and AC systems of the 5-bus and IEEE 14-bus cases verify that the DC approximation is an acceptable approximation for the study of the ESCOPF, at least for the cases studied. Moreover, the marginal values of spinning reserve and interruptible load can be studied from the solutions of ESCOPF case studies.

Future Work

So far, the ESCOPF model has been applied only to a few small hypothetical power systems. The largest system we analyzed had only fourteen buses and twenty transmission lines. Some of the data used in these cases are not very realistic. For example, some of the emergency line flow limits in the IEEE 14-bus case are assumed to be almost twice as much as their normal operating state limits. Moreover, the cost curves and customer benefit curves are assumed, and therefore may not be realistic. We would like to see our model applied to much larger and more realistic systems.

For much larger systems, the decoupled method has an advantage over the integrated method. It would allow us to incorporate *parallel processing* into the solution process. Several subproblem calculations can be done simultaneously by assigning each subproblem to a different computer. The information from these subproblems can be sent back to the main computer for use in solving the main problem of the ESCOPF. This will speed up the computation process. Instead of having one machine do all the work, the tasks will be shared among several machines. However, the decoupled method must be successfully implemented before parallel processing can be applied.

Since our case studies are very small and have only two generators, we ignore the contingencies involving the loss of a generator. The only type of contingencies that is considered in our test cases is the loss of transmission lines. The testing should be extended to take into account generator outages in systems that have more generators and are bigger than our case studies.

The ESCOPF model does not include the dynamic behavior of the power system. It is a *static* nonlinear problem. The pre-contingency operating point is assumed to exist up to the instant just before the contingency occurs. We do not consider the dynamic behavior that occurs during the transition from the pre-contingent operating point to the post-contingency state. If the contingency causes the system to become unstable, the post-contingency state cannot be reached even though there is a post-contingency solution. Future work on power system security should be extended to include dynamic stability.

BIBLIOGRAPHY

[1] U.S. Department of Energy. "An Analysis of the Electric Power Outages in the Western United States." *Report to the President of the United States*, August 1996.

[2] McCaw, R. "The Great Blackout." *Power Engineering*, pp. 36A–36C, December 1965.

[3] Kaye, J. R., Wu, F. F., and Varaiya, P. "Pricing for System Security." *IEEE Transactions on Power Systems*, 10(2):575–583, May 1995.

[4] Monticelli, A., Pereira, M. V. F., and Granville, S. "Security-Constrained Optimal Power Flow with Post-Contingency Corrective Rescheduling." *IEEE Transactions on Power Systems*, 2(1):175–181, February 1987.

[5] Stott, B., Alsac, O., and Monticelli, A. J. "Security Analysis and Optimization." *Proceedings of the IEEE*, 75(12):1623–1644, December 1987.

[6] Gedra, T. W. "A Framework of Security Pricing in a Re-regulated Environment." In *29th Annual Frontiers of Power Conferences*. October 1996.

[7] Wood, A. J. and Wollenberg, B. F. *Power Generation Operation and Control*. 2nd edition. John Wiley & Sons, Inc., New York, 1996.

[8] Grainger, J. J. and Stevenson Jr., W. D. *Power System Analysis*. McGraw-Hill, Inc., New York, 1994.

[9] Huneault, M. and Galiana, F. D. "A Survey of the Optimal Power Flow Literature." *IEEE Transactions on Power Systems*, 6(2):762–770, May 1991.

[10] Dommel, H. W. and Tinney, W. F. "Optimal Power Flow Solutions." *IEEE Transactions on Power Apparatus and Systems*, PAS-87(10):1866–1876, October 1968.

[11] Sun, D. I., Ashley, B., Brewer, B., Hughes, A., and Tinney, W. F. "Optimal Power Flow by Newton Approach." *IEEE Transactions on Power Apparatus and Systems*, PAS-103(10):2864–2875, October 1984.

[12] Sun, D. I., Demaree, K. D., and Brewer, B. "Application and Adaptation of Newton for Optimal Power Flow." In *IEEE Tutorial Course: Application of Optimization Methods for Economy/Security Functions in Power System Operations*, pp. 14–19. The Institute of Electrical and Electronics Engineers, Inc., New York, 1990.

[13] Monticelli, A. and Liu, Wen-Hsiung E. "Adaptive Movement Penalty Method for The Newton Optimal Power Flow." *IEEE Transactions on Power Systems*, 7(1):334–342, February 1992.

[14] Chen, Jiann-Fuh and Chen, Shin-Der. "Multiobjective Power Dispatch with Line Flow Constraints Using the Fast Newton-Raphson Method." *IEEE Transactions on Power Systems*, 5(4):1447–1454, November 1990.

[15] El-Hawary, M. E. and Ravindranath, K. M. "Combine Loss and Cost Objectives in Daily Hydro-Thermal Economic Scheduling." *IEEE Transactions on Power Systems*, 6(3):1106–1112, August 1991.

[16] Gedra, T. W. *Optimal Power Flow Using Newton's Method*, 1995. School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, Oklahoma.

[17] Hsiao, Ying-Tung, Liu, Chun-Chang, Chiang, Hsiao-Dong, and Chen, Yuan-Lin. "A New Approach for Optimal VAR Sources Planning in Large Scale Electric Power Systems." *IEEE Transactions on Power Systems*, 8(3):988–996, August 1993.

[18] Hong, Ying-Yi. "Impacts of Load Models on Solutions of Newton Optimal Power Flow." In *Proceedings of the 10th IEEE Region Conference on Computer, Communication, Control and Power Engineering*, pp. 398–401. Beijing, China, October 1993.

[19] Bjelogrlic, M., Calovic, M., and Babic, B. S. "Application of Newton's Optimal Power Flow in Voltage/Reactive Power Control." *IEEE Transactions on Power Systems*, 5(4):1447–1454, November 1990.

[20] Noroozian, M., Ängquist, L., Ghandhari, M., and Anderson, G. "Use of UPFC for Optimal Power Flow Control." *IEEE Transactions on Power Delivery*, 12(4):1629–1634, October 1997.

[21] Damrongkulkamjorn, P., Arcot, P. K., and Dcouto, P. "A Screening Technique for Optimally Locating Phase Shifter in Power System." In *IEEE/PES Transmission & Distribution Conference*, pp. 233–238. April 1994.

[22] Pilotto, L. A. S., Ping, W. W., Carvalho, A. R., Wey, A., Long, W. F., Alvarado, F. L., and Edris, A. "Determination of Needed FACTS Controllers that Increase Asset Utilization of Power Systems." *IEEE Transactions on Power Delivery*, 12(1):364–371, January 1997.

[23] Cova, B., Losignore, N., Marannino, P., and Montagna, M. "Contingency Constrained Optimal Reactive Power Flow Procedures for Voltage Control in Planning and Operation." Accepted for presentation at the IEEE/NTUA Atherns Power Tech Conference: Planning, Operation and Control of Today's Electric Power Systems, Athens, Greece.

[24] Gedra, T. W. "On Transmission Congestion and Pricing." IEEE Transactions on Power Systems, to appear.

[25] Crisan, O. and Mohtadi, M. A. "Efficient Identification of Binding Inequality Constraints in The Optimal Power Flow Newton Approach." *IEE Proceedings, Part C: Generation, Transmission and Distribution*, 139(5):365–370, September 1992.

[26] Tinney, W. F., Bright, J. M., Demaree, K. D., and Hughes, B. A. "Some Deficiencies in Optimal Power Flow." *IEEE Transactions on Power Systems*, 3(2):676–681, May 1988.

[27] Hong, Ying-Yi. "Enhanced Newton Optimal Power Flow Approach: Experiences in Taiwan Power System." *IEE Proceedings, Part C: Generation, Transmission and Distribution*, 139(3):205–210, May 1992.

[28] Maria, G. A. and Findlay, J. A. "A New Optimal Power Flow Program for Ontario Hydro EMS." *IEEE Transactions on Power Systems*, 2(3):576–584, August 1987.

[29] Santos Jr., A. and da Costa, G. R. M. "Optimal-Power-Flow Solution by Newton's Method Applied to an Augmented Lagrangian Function." *IEE Proceedings of Generation, Transmission, and Distribution*, 142(1):33–36, January 1995.

[30] Yan, X. and Quintana, V. H. "An Efficient Predictor-corrector Interio Point Algorithm for Security-Constrained Economic Dispatch." *IEEE Transactions on Power Systems*, 12(2):803–810, May 1997.

[31] Ponnambaiam, K., Quintana, V. H., and Vannelli, A. "A Fast Algorithm for Power System Optimization Problems Using An Interior Point Method." *IEEE Transactions on Power Systems*, 7(2):892–899, May 1992.

[32] Momoh, J. A., Austin, R. F., Adapa, R., and Ogbuobiri, E. C. "Application of Interior Point Method to Economic Dispatch." In *IEEE International Conference on Systems, Man and Cybernatics*, volume 2, pp. 1096–1101. 1992.

[33] Wu, Yu-Chi, Debs, A. S., and Marsten, R. E. "A Direct Nonlinear Predictor-Corrector Primal-Dual Interior Point Algorithm for Optimal Power Flows." *IEEE Transactions on Power Systems*, 9(2):876–883, May 1994.

[34] Wei, H., Sasaki, H., Kubokawa, J., and Yokoyama, R. "An Interior Point Nonlinear Programming for Optimal Power Flow Problems with A Novel Data Structure." In *20th International Conference on Power Industry Computer Applications*, pp. 134–141. 1997.

[35] Granville, S. "Optimal Reactive Dispatch Through Interior Point Methods." *IEEE Transactions on Power Systems*, 9(1):136–146, February 1994.

[36] da Costa, G. R. M. "Optimal Reactive Dispatch Through Primal-Dual Method." *IEEE Transactions on Power Systems*, 12(2):669–674, May 1997.

[37] Karmarkar, N. "A New Polynomial-time Algorithm for Linear Programming." *combinatorica*, 4:373–395, 1984.

[38] Wright, S. J. *Primal-Dual Interior-Point Methods*. SIAM: Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[39] Gedra, T. W. "Power Economics and Regulation." Class notes for ECEN 5193 at Oklahoma State University.

[40] Varian, H. R. *Microeconomic Analysis*. 2nd edition. W. W Norton & Company, Inc., New York, 1984.

[41] Oren, S. S., Spiller, P. T., Varaiya, P., and Wu, F. "Nodal Prices and Transmission Rights: A Critical Appraisal." *The Electricity Journal*, April 1995.

[42] Huang, G. and Song, K. "A Simple Two Stage Optimization Algorithm for Constrained Power Economic Dispatch." *IEEE Transactions on Power Systems*, 9(4):1818–1824, November 1994.

[43] Gjengedal, T. "Emission Constrained Unit-Commitment (ECUC)." *IEEE Transactions on Energy Conversion*, 11(1):1818–1824, March 1996.

[44] Rodrigues, M., Saavedra, O. R., and Monticelli, A. "Asynchronous Programming Model for the Concurrent Solution of the Security Constrained Optimal Power Flow Program." *IEEE Transactions on Power Systems*, 9(4):2021–2027, November 1994.

[45] Ng, W. Y. "Generalized Generation Distribution Factors for Power System Security Evaluations." *IEEE Transactions on Power Apparatus and Systems*, PAS-100(3):1001–1005, March 1981.

[46] Dandachi, N. H., Rawlins, M. J., Alsac, O., Prais, M., and Stott, B. "OPF for Reactive Pricing Studies on the NGC System." *IEEE Transactions on Power Systems*, 11(1):226–232, February 1996.

[47] Ambriz-Perez, H. "Incorporation of a UPFC Model in an Optimal Power Flow Using Newton's Method." *IEE Proceedings of Generation, Transmission, and Distribution*, 145(3):336–344, May 1998.

[48] Baughman, M. L., Siddiqi, S. N., and Zarnijau, J. W. "Advanced Pricing in Electrical Systems Part I: Theory." *IEEE Transactions on Power Systems*, 12(1):489–502, February 1997.

[49] Gedra, T. W. *Introduction to Constrained Optimization*, 1996. School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, Oklahoma.

[50] Harsan, H., Pruvot, P., and Hadjsaid, N. "An Efficient Method for Assessing Security on a Daily Basis." *IEEE Transactions on Power Systems*, 12(4):1542–1548, November 1997.

[51] Balu, N. "On-Line Power System Security Analysis." *Proceedings of the IEEE*, 80(2):262–280, February 1992.

[52] Ishikida, T. and Varaiya, P. P. "Pricing of Electric Power Under Uncertainty: Information and Efficiency." *IEEE Transactions on Power Systems*, 10(2):884–890, May 1995.

[53] Aganagic, M., Awobamise, B., Raina, G., and McCartney, A. I. "Economic Dispatch with Generation Contingency Constraints." *IEEE Transactions on Power Systems*, 12(3):1229–1234, August 1997.

[54] Aoki, K. and Satoh, T. "Economic Dispatch with Network Security Constraints Using Parametric Quadratic Programming." *IEEE Transactions on Power Apparatus and Systems*, PAS-101(12):4548–4556, December 1982.

[55] Alsac, O., Bright, J., Prais, M., and Stott, B. "Further Developments in LP-Based Optimal Power Flow." *IEEE Transactions on Power Systems*, 5(3):697–711, August 1990.

[56] Fahmideh-VojDani, A. R. and Galiana, F. D. "Economic Dispatch with Generation Constraints." *IEEE Transactions on Automatic Control*, 25(2):213–217, April 1980.

[57] Lu, C. N., Chen, S. S., and Ong, C. M. "The Incorporation of HVDC Equations in Optimal Power Flow Methods Using Sequential Quadratic Programming Technique." *IEEE Transactions on Power Systems*, 3(3):1005–1011, August 1988.

[58] Vaahedi, E., Magdy, H., and El-Din, Z. "Considerations in Applying Optimal Power Flow to to Power System Operation." *IEEE Transactions on Power Systems*, 4(2):694–700, May 1989.

[59] Almeida, K. C., Galiana, F. D., and Soares, S. "A General Parametric Optimal Power Flow." *IEEE Transactions on Power Systems*, 9(1):540–546, February 1994.

[60] Happ, H. H. "Optimal Power Dispatch - A Comprehensive Survey." *IEEE Transactions on Power Apparatus and Systems*, PAS-96(3):841–850, May/June 1977.

[61] Burchett, R. C., Happ, H. H., and Wirgau, K. A. "Large-Scale Optimal Power Flow." *IEEE Transactions on Power Apparatus and Systems*, PAS-101(10):3722–3732, October 1982.

[62] Wacker, G. "Customer Cost of Electric Service Interruptions." *Proceedings of the IEEE*, 77(6):919–929, June 1989.

[63] Sanghvi, A. P. "Measurement and Application of Customer Interruption Costs/Value of Service for Cost-Benefit Reliability Evaluation: Some Commonly Raised Issues." *IEEE Transactions on Power Systems*, 5(4):1333–1343, November 1990.

[64] Tollefson, G., Billinton, R., and Wacker, G. "Comprehensive Bibliography on Reliability Worth and Electrical Service Consumer Interruption Costs: 1980-1990." *IEEE Transactions on Power Systems*, 6(4):1508–1511, November 1991.

[65] Billington, R., Chan, E., and Wacker, G. "Probability Distribution Approach to Describe Customer Costs due to Electric Supply Interruptions." *IEE Proceedings of Generation, Transmission, and Distribution*, 141(6):594–598, November 1994.

[66] Ghajar, R., Billinton, R., and Chan, E. "Distributed Nature fo Residential Customer Outage Costs." *IEEE Transactions on Power Systems*, 11(3):1236–1244, August 1996.

[67] Subramanian, R. K., Billinton, R., and Wacker, G. "Understanding Industrail Losses Resulting from Electric Service Interruptions." *IEEE Transactions on Industrail Applications*, 29(1):238–244, January/February 1993.

[68] Sullivan, M. J. and Vardell, T. "Interruption Costs, Customer Satisfaction and Expectations for Service Reliability." *IEEE Transactions on Power Systems*, 11(2):989–995, May 1996.

[69] Sullivan, M. J., Vardell, T., and Johnson, M. "Power Interruption Costs to Industrial and Commercial Customers of Electricity." *IEEE Transactions on Industrail Applications*, 33(6):1448–1997, November/December 1997.

[70] Barcelo, W. R. and Rastgoufard, P. "Dynamic Economic Dispatch Using the Extended Security Constrained Economic Dipatch Algorithm." *IEEE Transactions on Power Systems*, 10(2):575–583, May 1995.

[71] Papalexopoulos, A., Hao, S., Liu, E., Alaywan, Z., and Kato, K. "Cost/Benefits Analysis of an Optimal Power Flow: The PG&E Experience." *IEEE Transactions on Power Systems*, 9(2):796–804, May 1994.

[72] Hong, Ying-Yi and Pan, Ching-Tsai. "An Enhanced Newton OPF." In *IEE International Conference on Advances in Power System Control, Operation and Management*, pp. 627–630. Hong Kong, November 1991.

[73] Hong, Ying-Yi. "Application of Newton Optimal Power Flow to Assessment of VAR Control Sequences on Voltage Security: Case Studies for a Practical Power System." *IEE Proceedings, Part C: Generation, Transmission and Distribution*, 140(6):539–544, November 1993.

[74] Majumdar, S., Chattopadhyay, D., and Parikh, J. "Interruptible Load Management Using Optimal Power Flow Analysis." *IEEE Transactions on Power Systems*, 11(2):715–720, May 1996.

[75] Working Group on a Common Format for Exchange of Solved Load Flow Data. "Common Format for Exchange of Solved Load Flow Data." *IEEE Transactions*

on Power Apparatus and Systems, PAS-92(6):1916–1925, November/December 1973.

APPENDIX A

ESCOPF PROGRAM

A.1 ESCOPF Program for DC Model

```
  function[] = ESCOPF()

% function[] = ESCOPF()
%
% Solve Expected Security-Cost Optimal Power Flow by Integrated Solution
% Method.

%=========================================================================
% Read data from ".cdf", ".gdf", and ".ldf".
% Then pass data to Initial to set up initial values.

  clear
  [o] = ReadDATA;
  [x_main] = Initial(o);    % Set initial values for MAIN.

%*****************************************
  if x_main.n.K==0          % In case of no contingency (base case OPF).
     x_sub = [];
  else
     for k = 1:x_main.n.K,  % Set initial values for SUB.
         x_sub(k) = Modified(o,x_main,k);
     end %for k
  end %if

%************WHILE LOOP*********************
converged = 0;
while (~converged)
%-----------------------------------------
% Compute W matrix of main problem.

  [Z,W_main,g0_main,norm_g_main] = Derivatives(x_main,0);
  x_main.var.Z = Z;

%-----------------------------------------
% Compute W matrices and set up indicies of the whole problem.

  [x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub,W_main,g0_main);

%-----------------------------------------
% Call subroutine to compute deltaZ of the huge problem.

  [x_all,converged] = IPRho(x_all,W_all,g_all,converged);

%-----------------------------------------
```

```
% Check for Neighbourhood of Central path.

  Nu = [x_all.var.MuH; x_all.var.MuL; x_all.var.MurhoH; x_all.var.MurhoL].* ...
       [x_all.var.sH;  x_all.var.sL;  x_all.var.rhoH;   x_all.var.rhoL  ];
  r = 0.25;
  N2_r = norm(Nu- ...
    x_main.var.Nu*ones(x_all.n.NaH+x_all.n.NaL+x_all.n.NaH+x_all.n.NaL,1)) ...
              -r*x_main.var.Nu;
  fprintf('N2_r = %8.4g \n',N2_r);
  fprintf('N_inf = %8.4g \n',norm(Nu,inf));

% Set flag to tell IP to calculate new Nu when we are in neighborhood of Nu.

  x_main.flag.newNu = 0;          % reset flag every time.
  if N2_r<= 0;
     x_main.flag.newNu = 1;
%    done = input(sprintf('Quit this problem? YES=1;NO=0 : '));
  end %if

%-------------------------------------------
% Calculate new Nu (or not).

  g_tol = 1e-6;   % Tolerance for gradient.
  hold_Nu = 0;    % Keep changing Nu to smaller value when flag.newNu is 1.

  if x_main.var.Nu <= x_main.var.NuLast
     hold_Nu = 1;              % Hold Nu to this value and try to get g to 0.
     if (norm(g_all)<=g_tol)   % Quit IP when g is this small.
         converged = 1;
         disp('*****************')
         disp('Program converges')
         disp('*****************')
     end % if (norm(g_all ...
  end %if x_main.var.Nu<= ...

  if (x_main.flag.newNu & ~hold_Nu)
     x_main.var.Nu = x_main.var.NuFac*x_main.var.Nu;
     for k = 1:x_main.n.K,
         x_sub(k).var.Nu = x_sub(k).var.NuFac*x_sub(k).var.Nu;
     end %for k
  end %if x_main.flag.newNu

%-------------------------------------------
% Unpack Z

  [x_main,x_sub] = UnPackZ(x_all,x_main,x_sub);

%-------------------------------------------
% Print out results for each iteration.

% look_g = input(sprintf(' Want to see gradient? YES=1; NO=0: '));
  look_g = 0;   % Don't need to look at gradient anymore.

% Compute MC and MB and Objective function.

  x_main.cost.MC = x_main.cost.BetaPu+(2*x_main.cost.GammaPu.*x_main.var.PG);
  x_main.cost.MB = x_main.cost.BetaLPu+(2*x_main.cost.GammaLPu.*x_main.var.PL);

  CostPG0 = (x_main.cost.Alpha) + ...
            (x_main.cost.Beta).* (x_main.m.SBase*x_main.var.PG) + ...
            (x_main.cost.Gamma).*(x_main.m.SBase*x_main.var.PG).^2;
  Benefit0 = (x_main.cost.AlphaL) + ...
             (x_main.cost.BetaL).*(x_main.m.SBase*x_main.var.PL) + ...
             (x_main.cost.GammaL).*(x_main.m.SBase*x_main.var.PL).^2;
  CostRho0 = x_main.cost.rho*[x_main.var.rhoH; x_main.var.rhoL];

  x_main.cost.Obj = x_main.m.Pi0*( sum(CostPG0)-sum(Benefit0)+sum(CostRho0) );
```

```
% Compute amount of load interrupted.

  x_main.var.PI = zeros(x_main.n.NBus,1);

  disp('Print statement for MAIN');
  if converged
     pause
  end %if converged
  Print(x_main,0,look_g,norm_g_main);

%-------------------------------------------
  for k = 1:x_main.n.K,
%     look_g = input(sprintf(' Want to see gradient? YES=1; NO=0: '));
      look_g = 0;

% Compute MC and MB and Objective function.

      x_sub(k).cost.MC = x_sub(k).cost.BetaPu + ...
                         (2*x_sub(k).cost.GammaPu.*x_sub(k).var.PG);
      x_sub(k).cost.MB = x_sub(k).cost.BetaLPu + ...
                         (2*x_sub(k).cost.GammaLPu.*x_sub(k).var.PL);

      CostPGk = (x_sub(k).cost.Alpha) + ...
                (x_sub(k).cost.Beta).* (x_sub(k).m.SBase*x_sub(k).var.PG) + ...
                (x_sub(k).cost.Gamma).*(x_sub(k).m.SBase*x_sub(k).var.PG).^2;
      Benefitk = (x_sub(k).cost.AlphaL) + ...
                 (x_sub(k).cost.BetaL).*(x_sub(k).m.SBase*x_sub(k).var.PL) + ...
                 (x_sub(k).cost.GammaL).*(x_sub(k).m.SBase*x_sub(k).var.PL).^2;
      CostIntr = (x_sub(k).cost.AlphaI) + ...
                 (x_sub(k).cost.BetaI).* ...
                 (x_sub(k).m.SBase*(x_main.var.PL-x_sub(k).var.PL)) + ...
                 (x_sub(k).cost.GammaI).* ...
                 (x_sub(k).m.SBase*(x_main.var.PL-x_sub(k).var.PL));
      CostRhok = x_sub(k).cost.rho*[x_sub(k).var.rhoH; x_sub(k).var.rhoL];

      x_sub(k).cost.Obj = x_sub(k).m.Pi(k)*( sum(CostPGk)-sum(Benefitk)+ ...
                                             sum(CostIntr)+sum(CostRhok) );

% Compute amount of load interrupted.

      PI = x_main.var.PL - x_sub(k).var.PL;
      x_sub(k).var.PI = zeros(x_main.n.NBus,1);
      x_sub(k).var.PI(x_sub(k).ind.Loadbus) = PI;

      fprintf(1,'Print statement for SUBPROBLEM  %2i which is line
             (%2i,%2i).\n',[k x_main.ind.fLOut(k) x_main.ind.tLOut(k)]');

      if converged
         pause
      end %if converged
      Print(x_sub(k),k,look_g,norm_g_sub(k));

      x_main.cost.Obj = x_main.cost.Obj+x_sub(k).cost.Obj;

  end %for k

%-------------------------------------------
% Print out total objective function, once ESCOPF is done.

  disp([ sprintf('Expected Security Cost  =')  sprintf(' %12.4f $/hour',x_main.cost.Obj)]);
%-------------------------------------------
end %while (~converged)

%*******************************************
```

```
  function[o] = ReadDATA()

% function[o] = ReadDATA()
%
% This function must be run at the beginning of the SCOPF program to read
% data from data files, i.e. CDF and GDF files.
% The data is read only one time through out the whole program regardless
% how many times the program calls main or subproblems.
%
% Only vectors or matrices of useful variables are passed to other .m files.
% This function also sets the neccessary convergence tolerances for SCOPF.

%=========================================================================
% Read CDF and GDF data of the selected ieee file.

  Filename = input('DC Case name: ','s');
  cdffile = [ Filename '.cdf' ];
  gdffile = [ Filename '.gdf' ];
  ldffile = [ Filename '.ldf' ];

% Create a structure containing bus data, branch data, and generator data
% coming out from rcdf & rgdf
  [o.bus,o.line] = ReadCDF(cdffile);
  [o.gen]        = ReadGDF(gdffile);
  [o.load]       = ReadLDF(ldffile);
  disp('DATA HAS BEEN READ IN');

%*************************************************************************
```

```
  function[ob,ol]=ReadCDF(cdffile)

% function[ob,ol]=ReadCDF(cdffile)
%
% Read in the data from .cdf file.

%=========================================================================
  cdffid = fopen(cdffile,'r');      % Read .cdf format file.
  cdfstr = fscanf(cdffid,'%c');     % Convert .cdf file to strings.
  lineend = find(cdfstr==char(10)); % I guess char(10) represents "return".
                                    % Lineend gives a vector of positions where
                                    % "returns" are.
  numlines = length(lineend);       % Since "return" is at the end of the line,
                                    % number of "returns" tells number of lines.
  linestart = [1 (lineend+1)];      % Add the first line, and take out the last
  linestart = linestart(1:numlines);% "return"

%**************************READ MVA BASE ON THE FIRST LINE*******************
    s = cdfstr(linestart(1):lineend(1));
    ob.SBase = str2num(s(32:37));

%**************************FIND BUS DATA SECTION*****************************
  count = 0;         % Start at first line and look for bus data section.
  looking = 1;
  tofind = 'bus data follows';
  while looking,
        count = count+1;
        if count>numlines,
           fprintf(1,'EOF reached with no bus data found.')
           break
        end%if
        s = cdfstr(linestart(count):lineend(count));
        pos = findstr(tofind,lower(s));   % Look for "BUS DATA FOLLOWS".
        looking = isempty(pos);           % "BUS DATA FOLLOWS" found.
  end%while
  pos2 = findstr('items',lower(s));
  if isempty(pos2), pos2=length(s)+1; end%if
  NBus_claimed = str2num(s(pos+length(tofind):pos2-1));
%**************************BUS DATA READING**********************************
  reading = 1;        % Start reading bus data when bus data section is found.
  ob.NBus = 0;        % Start counting buses.
  while reading
        count = count+1;    % Read the next line after "BUS DATA FOLLOWS".
        if count>numlines,
           fprintf(1,'EOF reached with no warning in bus data section.')
           break
        end%if
        s = cdfstr(linestart(count):lineend(count));
        reading = ~strncmp(s,'-999',4); % Stop reading when the line starts with
                                        % -999 in cols 1-4
        if ~reading,
           break
        end%if
        %*************************************************************
        ob.NBus = ob.NBus+1;
        ob.Busnum(ob.NBus,1)   = str2num(s(1:4));
        if ob.Busnum(ob.NBus,1) < 0, break, end%if
                                        % Negative bus # means end of section
        ob.Bustype(ob.NBus,1)  = str2num(s(26));
        ob.V(ob.NBus,1)        = str2num(s(28:33));
        ob.Angl(ob.NBus,1)     = (str2num(s(34:40)))*pi/180;
                                        % Convert from Degree to Radiant.
        ob.LoadMW(ob.NBus,1)   = str2num(s(41:49));
        ob.LoadMVAR(ob.NBus,1) = str2num(s(50:58));
        ob.GenMW(ob.NBus,1)    = str2num(s(59:67));
        ob.GenMVAR(ob.NBus,1)  = str2num(s(68:75));
        ob.VQmax(ob.NBus,1)    = str2num(s(91:98));
        ob.VQmin(ob.NBus,1)    = str2num(s(99:106));
```

```
      ob.Gshunt(ob.NBus,1)    = str2num(s(107:114));
      ob.Bshunt(ob.NBus,1)    = str2num(s(115:122));
      %************************************************************
  end%while

%************************FIND BRANCH DATA SECTION*****************************
  count = 0;         % Start at first line and look for branch data section.
  looking = 1;
  tofind = 'branch data follows';
  while looking,
      count = count+1;
      if count>numlines,
         fprintf(1,'EOF reached with no branch data found.')
         break
      end%if
      s = cdfstr(linestart(count):lineend(count));
      pos = findstr(tofind,lower(s));   % Look for "BRANCH DATA FOLLOWS".
      looking = isempty(pos);           % "BRANCH DATA FOLLOWS" found.
  end%while
  pos2 = findstr('items',lower(s));
  if isempty(pos2), pos2=length(s)+1; end%if   %???
  NLine_claimed = str2num(s(pos+length(tofind):pos2-1));
%************************BRANCH DATA READING*********************************
  reading = 1;        % Start reading branch data.
  ol.NLine = 0;       % for counting system transmission lines.
  while reading,
      count = count+1;    % Read the next line after "BRANCH DATA FOLLOWS".
      if count>numlines,
         fprintf(1,'EOF reached with no warning in branch data section.')
         break
      end%if
      s = cdfstr(linestart(count):lineend(count));
      reading = ~strncmp(s,'-999',4);
      if ~reading,
         break
      end%if
      %************************************************************
      ol.NLine = ol.NLine+1;
      l.From(ol.NLine,1)     = str2num(s(1:4));
      l.To(ol.NLine,1)       = str2num(s(6:9));
      if l.From(ol.NLine,1) < 0 | l.To(ol.NLine,1) < 0, break, end%if
                             % Negative bus # means end of section
      ol.Linetype(ol.NLine,1)= str2num(s(19));
      l.R(ol.NLine,1)        = str2num(s(20:29));
      l.X(ol.NLine,1)        = str2num(s(30:39));
      l.Bhalf(ol.NLine,1)    = str2num(s(41:49))/2;
      l.LineMVAR2(ol.NLine,1)= str2num(s(57:61));
      l.LineMVAR(ol.NLine,1) = str2num(s(63:67));
      l.T(ol.NLine,1)        = str2num(s(77:82));
      l.PHI(ol.NLine,1)      = str2num(s(84:90));
      l.TPHImin(ol.NLine,1)  = str2num(s(91:97));
      l.TPHImax(ol.NLine,1)  = str2num(s(98:104));
      l.Pi(ol.NLine,1)       = str2num(s(134:139));
      %************************************************************
  end%while

% Use branch data in structure "l" to form branch-data matrices "ol"
      ol.R     = sparse([l.From l.To],[l.To l.From],[l.R l.R], ...
                 ob.NBus,ob.NBus);
      ol.X     = sparse([l.From l.To],[l.To l.From],[l.X l.X], ...
                 ob.NBus,ob.NBus);
      ol.Bhalf = sparse([l.From l.To],[l.To l.From],[l.Bhalf l.Bhalf], ...
                 ob.NBus,ob.NBus);
      ol.Pi    = sparse(l.From, l.To, l.Pi/100, ob.NBus,ob.NBus);
      ol.Linemax = sparse([l.From l.To],[l.To l.From], ...
                   [l.LineMVAR l.LineMVAR], ob.NBus,ob.NBus);
      ol.Linemax2 = sparse([l.From l.To],[l.To l.From], ...
                    [l.LineMVAR2 l.LineMVAR2], ob.NBus,ob.NBus);
```

```
% Based on Linetype, some T/PHI are fixed, some T/PHI are variables
% (see /text/Definition).
        isTfix   = find(ol.Linetype==1);
        isPHIfix = find(ol.Linetype==1 | ol.Linetype==2 | ol.Linetype==3);
        isTvar   = find(ol.Linetype==2 | ol.Linetype==3);
        isPHIvar = find(ol.Linetype==4);

        ol.Tfix   = sparse([l.From(isTfix)],[l.To(isTfix)],[l.T(isTfix)], ...
                    ob.NBus,ob.NBus);
        ol.Tvar   = sparse([l.From(isTvar)],[l.To(isTvar)],[l.T(isTvar)], ...
                    ob.NBus,ob.NBus);
        ol.T      = ol.Tfix + ol.Tvar;
        ol.PHIfix = sparse([l.From(isPHIfix)],[l.To(isPHIfix)], ...
                    [l.PHI(isPHIfix)],ob.NBus,ob.NBus);
        ol.PHIvar = sparse([l.From(isPHIvar)],[l.To(isPHIvar)], ...
                    [l.PHI(isPHIvar)],ob.NBus,ob.NBus);
        ol.PHI    = ol.PHIfix + ol.PHIvar;

        ol.Tmax   = sparse([l.From(isTvar)],[l.To(isTvar)], ...
                    [l.TPHImax(isTvar)],ob.NBus,ob.NBus);
        ol.Tmin   = sparse([l.From(isTvar)],[l.To(isTvar)], ...
                    [l.TPHImin(isTvar)],ob.NBus,ob.NBus);
        ol.PHImax = sparse([l.From(isPHIvar)],[l.To(isPHIvar)], ...
                    [l.TPHImax(isPHIvar)],ob.NBus,ob.NBus);
        ol.PHImin = sparse([l.From(isPHIvar)],[l.To(isPHIvar)], ...
                    [l.TPHImin(isPHIvar)],ob.NBus,ob.NBus);

%*****************************************************************************
% Checking on claimed NBus and NLine

  if NBus_claimed~=ob.NBus
     disp('Number of Bus Data Items is NOT EQUAL to Number of Buses')
  end%if
  if NLine_claimed~=ol.NLine
     disp('Number of Branch Data Items is NOT EQUAL to Number of Lines')
  end%if

%=============================================================================
```

```
  function[og] = ReadGDF(gdffile)

% function[og] = ReadGDF(gdffile)
%
% Read in the data from .gdf file.

%=========================================================================
  gdffid = fopen(gdffile,'r');      % Read .gdf format file.
  gdfstr = fscanf(gdffid,'%c');     % Convert .gdf file to strings.
  lineend = find(gdfstr==char(10)); % char(10)is carrige return.
  numlines = length(lineend);       % Since "return" is at the end of the line,
                                    % number of "returns" tells number of lines.
  linestart = [1 (lineend+1)];      % Add the first line, and take out the last
  linestart = linestart(1:numlines);% "return"

%*********************READING GENERATOR DATA*****************************
  count = 2;                        % I assume the first 2 lines are not data.
                                    % They are the comments of the file.
  og.NGen = 0;                      % Start counting generator buses.
  while count<numlines

        count = count+1;
        s = gdfstr(linestart(count):lineend(count));

        Avail = str2num(s(51));     % Check availability of generator first.
        if Avail == 0,              % If not available, don't read the line,
    if count<numlines,       % skip to the next line in GDF file,
              count = count+1;      % but make sure there is the next line.
              s = gdfstr(linestart(count):lineend(count));
    else
              break                 % <- We are at the last line and we don't
    end%if                    % want to read it.  So, get out of here.
        end%if
        %*********************************************************************
        og.NGen = og.NGen+1;
        og.Busnum(og.NGen,1) = str2num(s(1:5));

        if og.Busnum<0, break, end%if    % Negative bus # means end of section

        og.MWmax(og.NGen,1)   = str2num(s(17:23));
        og.MWmin(og.NGen,1)   = str2num(s(25:29));
        og.MVARmax(og.NGen,1) = str2num(s(31:38));
        og.MVARmin(og.NGen,1) = str2num(s(40:45));
        og.PFree(og.NGen,1)   = str2num(s(47));
        og.QFree(og.NGen,1)   = str2num(s(49));
        og.Alpha(og.NGen,1)   = str2num(s(53:59));
        og.Beta(og.NGen,1)    = str2num(s(61:66));
        og.Gamma(og.NGen,1)   = str2num(s(68:75));
        og.VGmax(og.NGen,1)   = str2num(s(77:81));
        og.VGmin(og.NGen,1)   = str2num(s(83:87));
        og.MWup(og.NGen,1)    = str2num(s(89:93));
        og.MWdn(og.NGen,1)    = str2num(s(95:99));
        og.MVARup(og.NGen,1)  = str2num(s(101:106));
        og.MVARdn(og.NGen,1)  = str2num(s(108:113));
        og.BetaR(og.NGen,1)   = str2num(s(115:120));
        og.GammaR(og.NGen,1)  = str2num(s(122:129));
        %*********************************************************************
  end%while

%=========================================================================
```

```
  function[od] = ReadLDF(ldffile)

% function[od] = ReadLDF(ldffile)
%
% Read in the data from .ldf file.

%=============================================================================
  ldffid = fopen(ldffile,'r');      % Read .ldf format file.
  ldfstr = fscanf(ldffid,'%c');     % Convert .ldf file to strings.
  lineend = find(ldfstr==char(10)); % char(10)is a carrige return.
  numlines = length(lineend);       % Since "return" is at the end of the line,
                                    % number of "returns" tells number of lines.
  linestart = [1 (lineend+1)];      % Add the first line, and take out the last
  linestart = linestart(1:numlines);% "return"

%************************READING LOAD DATA*****************************
  count = 2;                        % I assume the first 2 lines are not data.
                                    % They are the comments of the file.
  od.NLoad = 0;                     % Start counting load buses.
  while count<numlines

        count = count+1;
        s = ldfstr(linestart(count):lineend(count));
        %*********************************************************************
        od.NLoad = od.NLoad+1;
        od.Busnum(od.NLoad,1) = str2num(s(1:5));

        if od.Busnum<0, break, end%if   % Negative bus # means end of section

        od.Loadtype(od.NLoad,1) = str2num(s(15:18));
        od.MWmax(od.NLoad,1)    = str2num(s(20:25));
        od.MWmin(od.NLoad,1)    = str2num(s(27:32));
        od.MVARmax(od.NLoad,1)  = str2num(s(34:40));
        od.MVARmin(od.NLoad,1)  = str2num(s(42:48));
        od.Intr(od.NLoad,1)     = str2num(s(50));
        od.MWmaxInt(od.NLoad,1) = str2num(s(52:59));
        od.Alpha(od.NLoad,1)    = str2num(s(61:68));
        od.Beta(od.NLoad,1)     = str2num(s(70:77));
        od.Gamma(od.NLoad,1)    = str2num(s(79:87));
        od.AlphaI(od.NLoad,1)   = str2num(s(89:96));
        od.BetaI(od.NLoad,1)    = str2num(s(98:104));
        od.GammaI(od.NLoad,1)   = str2num(s(106:113));
        od.PF(od.NLoad,1)       = str2num(s(115:120));
        %*********************************************************************

  end%while

%=============================================================================
```

```
  function[x] = Initial(o)

% function[x] = Initial(o)
%
% This function sets up initial values for variables and multipliers to be
% used through out the program.

%=============================================================================
  x.n.NBus  = o.bus.NBus;
  x.n.NGen  = o.gen.NGen;
  x.n.NLine = o.line.NLine;
  x.n.NLoad = o.load.NLoad;

  x.m.Bshunt = o.bus.Bshunt;
  x.m.SBase  = o.bus.SBase;
  x.m.CBase  = o.bus.SBase;

%*****************************************************************************
% Indexing

  x.ind.Slackbus = o.bus.Busnum(find(o.bus.Bustype==3));
  x.ind.PQbus    = o.bus.Busnum(find(o.bus.Bustype==0));
  x.ind.PVbus    = o.bus.Busnum(find((o.bus.Bustype==2)|(o.bus.Bustype==3)));
  x.ind.Intr     = find(o.load.Intr);        % where o.load.Intr = 1.
  x.ind.UnIntr   = find(~o.load.Intr);       % where o.load.Intr = 0.
  x.ind.LoadbusI = o.load.Busnum(x.ind.Intr);
  x.ind.LoadbusU = o.load.Busnum(x.ind.UnIntr);
  x.ind.Loadbus = o.load.Busnum;

  x.ind.PFree_1 = find(o.gen.PFree);         % where o.gen.PFree = 1.
  x.ind.QFree_1 = find(o.gen.QFree);         % where o.gen.QFree = 1.
  x.ind.PFree_0 = find(~o.gen.PFree);        % where o.gen.PFree = 0.
  x.ind.QFree_0 = find(~o.gen.QFree);        % where o.gen.QFree = 0.
  x.ind.PFree  =  o.gen.Busnum(x.ind.PFree_1);
  x.ind.QFree  =  o.gen.Busnum(x.ind.QFree_1);
  x.ind.PFixed = [o.gen.Busnum(x.ind.PFree_0)
                  x.ind.PQbus               ];
  x.ind.QFixed = [o.gen.Busnum(x.ind.QFree_0)
                  x.ind.PQbus               ];

%*****************************************************************************
% Calculate B matrix.

  Imp = (o.line.R) + j*(o.line.X);
  [f,t,Imp] = find(Imp);
  x.m.Y = sparse(f,t,1./Imp,o.bus.NBus,o.bus.NBus);
  x.m.YB = x.m.Y + j*(o.line.Bhalf);

  x.m.Bline = imag(x.m.YB);                % These are susceptance of line ij.
  x.m.B = x.m.Bline - diag(sum(x.m.Bline)); % This is not 1/YBUS.
  x.m.B(x.ind.Slackbus,:) = [];            % Take out slack bus.
  [x.ind.f, x.ind.t, x.m.Bline_v] = find(triu(x.m.Bline));

%*****************************************************************************
% Calculate some useful numbers that will be refered to later
% i.e. # of variables, # of constraint functions, ...

  x.n.NPFixed = length(x.ind.PFixed);   % number of buses of P fixed.
  x.n.NPFree  = length(x.ind.PFree);    % number of buses of P free.

  x.n.NX = x.n.NBus-1;                  % number of variables, Angl only.
  x.n.NVar = x.n.NX + x.n.NPFree + x.n.NLoad + x.n.NLine;
                                        % number of state variables w/o
                                        % angle of Slack bus.
  x.n.NZ0 = x.n.NVar + x.n.NBus + x.n.NLine;
                                        % number of vector Z w/o Mu's.

%*****************************************************************************
```

```
% Forming matrices for indexing.

  x.m.ft2i    = sparse(x.ind.f,x.ind.t,[1:x.n.NLine],x.n.NBus,x.n.NBus);

%*************************************************************************
% Set Up Indicies for Vectors.
% Indexing vector Z

  x.ind.Slack   = x.ind.Slackbus;              % index of angle of slack bus
                                               % in vector Z.  In DC, it's
                                               % the same as Slackbus.
  x.ind.Angl    = [1:(x.n.NBus-1)]';           next = (x.n.NBus-1);
  x.ind.PG      = [next+1:next+x.n.NPFree]';   next = next + x.n.NPFree;
  x.ind.PL      = [next+1:next+x.n.NLoad]';    next = next + x.n.NLoad;
  x.ind.Ps      = [next+1:next+x.n.NLine]';    next = next + x.n.NLine;
  x.ind.LambdaP = [next+1:next+x.n.NBus]';     next = next + x.n.NBus;
  x.ind.LambdaPs= [next+1:next+x.n.NLine]';

  x.ind.Z0 = [x.ind.Angl; x.ind.PG; x.ind.PL; x.ind.Ps; ...
              x.ind.LambdaP; x.ind.LambdaPs];

%*************************************************************************
% Initial values of Multipliers (LambdaP, LambdaPs).
% LambdaP should be close to (average) Marginal Cost of the system.
% Thus, LambdaP will be initiallized when MC is computed (in PowerFlow.m).

  x.var.LambdaP  = ones(x.n.NBus,1);     % will be changed to MC later.
% x.var.LambdaP  = [27; 28; 28];
  x.var.LambdaPs = ones(x.n.NLine,1);
% x.var.LambdaPs = [1; 0; 0];

% Initial values of Mu's are set when method is chosen (at the end of
% this function).

%*************************************************************************
% Vector of upper and lower limits (for MAIN problem).

  x.lim.PGmax = o.gen.MWmax(x.ind.PFree_1)/x.m.SBase;
  x.lim.PLmax = o.load.MWmax/x.m.SBase;
  [junk, junk, x.lim.Pmax]  = find(triu(o.line.Linemax)./x.m.SBase);

  x.lim.PGmin = o.gen.MWmin(x.ind.PFree_1)/x.m.SBase;
  x.lim.PLmin = o.load.MWmin/x.m.SBase;
  x.lim.Pmin  = -x.lim.Pmax;

  x.lim.Max = [x.lim.PGmax; x.lim.PLmax; x.lim.Pmax];
  x.lim.Min = [x.lim.PGmin; x.lim.PLmin; x.lim.Pmin];

%*************************************************************************
% Initial values of variables, Angl, PG, PL, and Ps.
% Initial value of slack s is set in Modified.

  x.var.Angl = o.bus.Angl;
  x.var.PG   = (x.lim.PGmax+x.lim.PGmin)/2;
  x.var.PL   = ones(x.n.NLoad,1);
  x.var.Ps   = (x.lim.Pmax)/2;

%*************************************************************************
% Prepare data for cost function.

  x.cost.Alpha  = o.gen.Alpha(x.ind.PFree_1);
  x.cost.Beta   = o.gen.Beta(x.ind.PFree_1);
  x.cost.Gamma  = o.gen.Gamma(x.ind.PFree_1);
  x.cost.BetaR  = o.gen.BetaR(x.ind.PFree_1);
  x.cost.GammaR = o.gen.GammaR(x.ind.PFree_1);
  x.cost.AlphaL = o.load.Alpha;
  x.cost.BetaL  = o.load.Beta;
  x.cost.GammaL = o.load.Gamma;
```

```
  x.cost.AlphaI = o.load.AlphaI;
  x.cost.BetaI  = o.load.BetaI;
  x.cost.GammaI = o.load.GammaI;

  x.cost.BetaPu  = x.cost.Beta*(x.m.SBase/x.m.CBase);
  x.cost.GammaPu = x.cost.Gamma*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaLPu  = x.cost.BetaL*(x.m.SBase/x.m.CBase);
  x.cost.GammaLPu = x.cost.GammaL*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaRPu  = x.cost.BetaR*(x.m.SBase/x.m.CBase);
  x.cost.GammaRPu = x.cost.GammaR*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaIPu  = x.cost.BetaI*(x.m.SBase/x.m.CBase);
  x.cost.GammaIPu = x.cost.GammaI*(x.m.SBase^2/x.m.CBase);

  x.cost.d2  = 2*x.cost.GammaPu;
  x.cost.d2L = 2*x.cost.GammaLPu;
  x.cost.d2R = 2*x.cost.GammaRPu;
  x.cost.d2I = 2*x.cost.GammaIPu;

%*************************************************************************
% Information on contingencies (line outages).

  [x.ind.fLOut,x.ind.tLOut,x.m.Pi] = find(o.line.Pi);
  x.m.Pi0 = 1-sum(x.m.Pi);      % probability of no contingencies.
  x.n.K = length(x.m.Pi);       % number of contingencies, k = 1...K.

  if x.n.K==0   % no contingency.
    x.m.Pi = sparse(x.n.NLine,1);
  end %if

%*************************************************************************
% Define constraint type for printing the results.

  x.type.PG   = 1;      x.type.type1 = [ 'PG  ' ];
  x.type.PL   = 2;      x.type.type2 = [ 'PL  ' ];
  x.type.Ps   = 3;      x.type.type3 = [ 'Ps  ' ];

  x.print.Type = [ x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))
                   x.type.Ps(ones(x.n.NLine,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))  ];

  x.print.Name = [ x.type.type1(ones(1,x.n.NPFree),:)
                   x.type.type2(ones(1,x.n.NLoad),:)
                   x.type.type3(ones(1,x.n.NLine),:)
                   x.type.type1(ones(1,x.n.NPFree),:)
                   x.type.type2(ones(1,x.n.NLoad),:)   ];

  x.print.Number = [ (x.ind.PFree)
                     (x.ind.Loadbus)
                     (1:x.n.NLine)'
                     (x.ind.PFree)
                     (x.ind.Loadbus) ];

%*************************************************************************
% Pick the method.  Once you pick it here, it can't be changed throughout
% the whole program.

  %---------------------------------
  clear x.flag.ip;
  flagIP = 2;        % default value.
  x.flag.ip = input(sprintf ...
  (' FP press 0; IP press 1; IPwRho press 2 (default= %2i): ', flagIP));
     if isempty(x.flag.ip)
        x.flag.ip = flagIP;
```

```
   end %if
%----------------------------------

if x.flag.ip
   x.ind.A.PG = [1:x.n.NPFree]';                  next = x.n.NPFree;
   x.ind.A.PL = [next+1:next+x.n.NLoad]';         next = next + x.n.NLoad;
   x.ind.A.Ps = [next+1:next+x.n.NLine]';
   x.ind.aH = [x.ind.A.PG; x.ind.A.Ps];
   x.ind.aL = [x.ind.A.PG; x.ind.A.PL; x.ind.A.Ps];

   x.n.NaH = length(x.ind.aH);           % Number of upper limits.
   x.n.NaL = length(x.ind.aL);           % Number of lower limits.

   x.var.sH  = ones(x.n.NaH,1);
   x.var.sL  = ones(x.n.NaL,1);
   x.var.MuH = 0.1*ones(x.n.NaH,1);
   x.var.MuL = 0.1*ones(x.n.NaL,1);
   x.var.rhoH = 0.1*ones(x.n.NaH,1);
   x.var.rhoL = 0.1*ones(x.n.NaL,1);
   x.var.MurhoH = 1000*ones(x.n.NaH,1);
   x.var.MurhoL = 1000*ones(x.n.NaL,1);

   %----------------------------------
   clear x.printIP;
   printIP = 1e-6;  % default value.
   x.printIP = input(sprintf ...
(' Start showing print statement when Nu is this small (default=%6.4g): ',...
   printIP));
   if isempty(x.printIP)
      x.printIP = printIP;
   end %if
   %----------------------------------
   clear x.var.NuFac
   NuFac = 0.1;    % default value.
   x.var.NuFac = input(sprintf ...
                (' Enter factor of Nu (default=%6.4g): ', NuFac));
   if isempty(x.var.NuFac)
      x.var.NuFac = NuFac;
   end %if
   %----------------------------------
   clear x.var.NuLast
   NuLast = 1e-15;  % default value.
   x.var.NuLast = input(sprintf ...
             (' Enter the last value of Nu (default=%6.4g): ', NuLast));
   if isempty(x.var.NuLast)
      x.var.NuLast = NuLast;
   end %if
   %----------------------------------
   clear x.m.sig
   sig = 1;
   x.m.sig = input(sprintf ...
                (' Enter sigma (default=%4i): ', sig));
   if isempty(x.m.sig)
      x.m.sig = sig;
   end %if
   %----------------------------------
   if x.flag.ip==2
      clear x.cost.rho;
      cost_rho = 1000;   % default value (per additional MW or MVar).
      x.cost.rho =  input(sprintf ...
        (' Enter penalty cost for exceeding limits (default=%6i): ', ...
           cost_rho));
      if isempty(x.cost.rho)
         x.cost.rho = cost_rho;
      end %if
      %----------------------------------
      x.var.Nu = 1/(2*(x.n.NaH+x.n.NaL))* ...
                 sum([x.var.MuH;x.var.MuL;x.var.MurhoH;x.var.MurhoL].*...
```

```
                    [x.var.sH;x.var.sL;x.var.rhoH;x.var.rhoL]);
      end %if x.flag.ip==2
      %---------------------------------
  else % for FP
     x.var.MuH  = [];                  % Mu for X binding at upper limit.
     x.var.MuL  = [];                  % Mu for X binding at lower limit.
     x.ind.aH   = [];                  % index of upper active set
     x.ind.aL   = [];                  % index of lower active set
     x.ind.aMuH = [];                  % index of MuH change sign
     x.ind.aMuL = [];                  % index of MuL change sign

  end %if x.flag.ip

%************************************************************************
% Set up maximum number of iterations for OPF.

  x.n.NIter = 100;           % Set up default value for number of iterations.
  NIter_new = input(sprintf(...
      ' Enter MAX number of iterations for OPF (default=%4i): ', x.n.NIter));

  if ~isempty(NIter_new),
     x.n.NIter = NIter_new;
  end

%************************************************************************
% Iteration number of the whole problem

  x.Iter = 1;

%************************************************************************
% Save initial variables.

  x.init = x;

%************************************************************************
```

```
  function[x] = Modified(o,x,k)

% function[x] = Modified(o,x,k)
%
% This function modifies the structure of the system for each subproblem,
% and set initial values.
% So it does the same thing as Initial, but for subproblems.
%
% For each subproblem, input variables and indicies are from x_main.

%=========================================================================

  x.var = x.init.var;
  x.ind = x.init.ind;

%*************************************************************************
% Modify the system by taking out one line for each subproblem.

  x.n.NLine = o.line.NLine-1;
  x.n.NLoad = nnz(o.load.Intr);
  x.ind.Loadbus = x.ind.LoadbusI;
  x.var.Pl(x.ind.LoadbusU) = x.var.PL(x.ind.UnIntr);
  x.var.PL = x.var.PL(x.ind.Intr);    % Only interruptible PL are variables.
  x.var.Ps(k) = [];                   % Take out line number k.

  fk = [x.ind.fLOut(k); x.ind.tLOut(k)];
  tk = [x.ind.tLOut(k); x.ind.fLOut(k)];

  o.line.Linetype(fk,tk) = 0;
  o.line.R(fk,tk) = 0;
  o.line.X(fk,tk) = 0;
  o.line.Bhalf(fk,tk) = 0;
% o.line.Linemax(fk,tk) = 0;          % don't need this, use Linemax2.
  o.line.Linemax2(fk,tk) = 0;
  o.line.Tfix(fk,tk) = 0;
  o.line.Tvar(fk,tk) = 0;
  o.line.T(fk,tk) = 0;
  o.line.PHIfix(fk,tk) = 0;
  o.line.PHIvar(fk,tk) = 0;
  o.line.PHI(fk,tk) = 0;
  o.line.Tmax(fk,tk) = 0;
  o.line.Tmin(fk,tk) = 0;
  o.line.PHImax(fk,tk) = 0;
  o.line.PHImin(fk,tk) = 0;

%*************************************************************************
% Calculate some useful numbers after one line is taken out.
% i.e. # of variables, # of constraint functions, ...

  x.n.NVar = x.n.NX + x.n.NPFree + x.n.NLoad + x.n.NLine;
                                      % number of state variables w/o
                                      % angle of Slack bus
  x.n.NZ0 = x.n.NVar + x.n.NBus + x.n.NLine;
                                      % number of vector Z w/o Mu's

%*************************************************************************
% Special limits that don't appear in MAIN problem.

  [junk,junk,x.lim.Pmax] = find(triu(o.line.Linemax2)./x.m.SBase);
  x.lim.Pmin  = -x.lim.Pmax;
  x.lim.MWup = o.gen.MWup(x.ind.PFree_1)/x.m.SBase;
  x.lim.MWdn = o.gen.MWdn(x.ind.PFree_1)/x.m.SBase;
  x.lim.MWmaxInt = o.load.MWmaxInt(x.ind.Intr)/x.m.SBase;

%*************************************************************************
% Calculate B matrix based on modified system.

  Imp = (o.line.R) + j*(o.line.X);
```

```
  [f,t,Imp] = find(Imp);
  x.m.Y = sparse(f,t,1./Imp,o.bus.NBus,o.bus.NBus);
  x.m.YB = x.m.Y + j*(o.line.Bhalf);

  x.m.Bline = imag(x.m.YB);                % These are susceptance of line ij.
  x.m.B = x.m.Bline - diag(sum(x.m.Bline)); % This is not 1/YBUS.
  x.m.B(x.ind.Slackbus,:) = [];            % Take out slack bus.
  [x.ind.f, x.ind.t, x.m.Bline_v] = find(triu(x.m.Bline));

%*************************************************************************
% New initial values for each subproblem.  Make sure they are inside
% feasible region because we start out with no binding contraints.

  x.var.PG = min(x.var.PG, 0.90*x.lim.PGmax);
  x.var.PG = max(x.var.PG, 1.10*x.lim.PGmin);

  x.var.PL = min(x.var.PL, 0.90*x.lim.PLmax);
  x.var.PL = max(x.var.PL, x.lim.PLmin);         % PLmin is always zero.

  x.var.Ps = min(x.var.Ps, 0.90*x.lim.Pmax);
  x.var.Ps = max(x.var.Ps, 1.10*x.lim.Pmin);

%*************************************************************************
% Forming matrices for indexing.

  x.m.ft2i    = sparse(x.ind.f,x.ind.t,[1:x.n.NLine],x.n.NBus,x.n.NBus);

%*************************************************************************
% Set Up Indicies for Vectors.
% Indexing vector Z

  x.ind.Slack   = x.ind.Slackbus;              % index of angle of slack bus
                                               % in vector Z.
  x.ind.Angl    = [1:(x.n.NBus-1)]';           next = (x.n.NBus-1);
  x.ind.PG      = [next+1:next+x.n.NPFree]';   next = next + x.n.NPFree;
  x.ind.PL      = [next+1:next+x.n.NLoad]';    next = next + x.n.NLoad;
  x.ind.Ps      = [next+1:next+x.n.NLine]';    next = next + x.n.NLine;
  x.ind.LambdaP = [next+1:next+x.n.NBus]';     next = next + x.n.NBus;
  x.ind.LambdaPs= [next+1:next+x.n.NLine]';
  x.ind.Z0 = [x.ind.Angl; x.ind.PG; x.ind.PL; x.ind.Ps; ...
              x.ind.LambdaP; x.ind.LambdaPs];

%*************************************************************************
% Initial values of Multipliers (LambdaP, LambdaPs).
% LambdaP should be close to (average) Marginal Cost of the system.
% Thus, LambdaP will be initiallized when MC is computed (in PowerFlow.m).

  x.var.LambdaP  = ones(x.n.NBus,1);     % will be changed to MC later.
  x.var.LambdaPs = ones(x.n.NLine,1);

% Initial values of Mu's and slacks in subproblems.

  x.ind.A.PG = [1:x.n.NPFree]';                next = x.n.NPFree;
  x.ind.A.PL = [next+1:next+x.n.NLoad]';       next = next + x.n.NLoad;
  x.ind.A.Ps = [next+1:next+x.n.NLine]';       next = next + x.n.NLine;
  x.ind.A.PGC = [next+1:next+x.n.NPFree]';     next = next + x.n.NPFree;
  x.ind.A.PLC = [next+1:next+x.n.NLoad]';

  x.ind.aH_k = [x.ind.A.PG; x.ind.A.Ps];
  x.ind.aH_k0 = [x.ind.A.PGC; x.ind.A.PLC];
  x.ind.aH = [x.ind.aH_k; x.ind.aH_k0];
  x.ind.aL_k = [x.ind.A.PG; x.ind.A.PL; x.ind.A.Ps];
  x.ind.aL_k0 = [x.ind.A.PGC; x.ind.A.PLC];
  x.ind.aL = [x.ind.aL_k; x.ind.aL_k0];

  x.n.NaH_k  = length(x.ind.aH_k);
  x.n.NaH_k0 = length(x.ind.aH_k0);
  x.n.NaH    = length(x.ind.aH);
```

```
  x.n.NaL_k  = length(x.ind.aL_k);
  x.n.NaL_k0 = length(x.ind.aL_k0);
  x.n.NaL    = length(x.ind.aL);

  x.var.sH  = 0.1*ones(x.n.NaH,1);
  x.var.sL  = 0.1*ones(x.n.NaL,1);
  x.var.MuH = ones(x.n.NaH,1);
  x.var.MuL = ones(x.n.NaL,1);
  x.var.rhoH = ones(x.n.NaH,1);
  x.var.rhoL = ones(x.n.NaL,1);
  x.var.MurhoH = 1000*ones(x.n.NaH,1);
  x.var.MurhoL = 1000*ones(x.n.NaL,1);

%*************************************************************************
% Define constraint type for printing the results.

  x.print.Type = [ x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))
                   x.type.Ps(ones(x.n.NLine,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))  ];

  x.print.Name = [ x.type.type1(ones(1,x.n.NPFree),:)
                   x.type.type2(ones(1,x.n.NLoad),:)
                   x.type.type3(ones(1,x.n.NLine),:)
                   x.type.type1(ones(1,x.n.NPFree),:)
                   x.type.type2(ones(1,x.n.NLoad),:)   ];

% Which line is gone???

  Line_num_new = (1:x.init.n.NLine)';
  Line_num_new(k) = [];
  x.print.Number = [ (x.ind.PFree)
                     (x.ind.Loadbus)
                     Line_num_new
                     (x.ind.PFree)
                     (x.ind.Loadbus) ];

%*************************************************************************
```

```
  function[x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub,W_main,
                                                     g0_main);

% function[x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub,W_main,
%                                                     g0_main)
%
% 1. Form W matrix and vector g for the whole problem (with main and all
%    subproblems).
% 2. Prepare variables and their indicies for deltaZ_all.

%=========================================================================
% Before going inside FOR loop for computing subproblems, prepare variables
% and indicies from main problem.

  x_all.ind.Angl = x_main.ind.Angl;
  x_all.ind.PG = x_main.ind.PG;
  x_all.ind.PL = x_main.ind.PL;
  x_all.ind.Ps = x_main.ind.Ps;
  x_all.ind.Z0 = x_main.ind.Z0;
  x_all.ind.LambdaP = x_main.ind.LambdaP;
  x_all.ind.LambdaPs = x_main.ind.LambdaPs;
  x_all.ind.Slack = x_main.ind.Slack;
  x_all.ind.Slackbus = x_main.ind.Slackbus;

  x_all.ind.MuH = (x_main.n.NZ0+1 : x_main.n.NZ0+x_main.n.NaH)';
  x_all.ind.MuL = (x_main.n.NZ0+x_main.n.NaH+1 :  ...
                   x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL)';
  x_all.ind.MurhoH = (x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL+1 : ...
                      x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL)';
  x_all.ind.MurhoL = (x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL+1 : ...
                      x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL))';
  x_all.ind.sH = (x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL)+1 : ...
                  x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL)';
  x_all.ind.sL = (x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL+1 : ...
                  x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL))';
  x_all.ind.rhoH = (x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL)+1 : ...
                    x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL)';
  x_all.ind.rhoL = (x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL+1 : ...
                    x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL))';

  x_all.var.Z = x_main.var.Z;
  x_all.n.NaH = x_main.n.NaH;
  x_all.n.NaL = x_main.n.NaL;

  next = length(g0_main);
%*************************************************************************
  for k = 1:x_main.n.K,

      x_sub = Limits(x_main,x_sub,k);
      x = x_sub(k);
      [Z,W,g0,norm_g_sub(k)] = Derivatives(x,k);
      x_sub(k).var.Z = Z;

% Compute coupling matrix.

      Coupl = [sparse(x_main.n.NZ0,x_sub(k).n.NZ0), ...
               sparse(x_main.n.NZ0,x_sub(k).n.NaH_k), ...
               spdiags(-ones(x_sub(k).n.NPFree,1), -x_main.n.NX, ...
                       x_main.n.NZ0, x_sub(k).n.NPFree), ...
               spdiags(-ones(x_sub(k).n.NLoad,1), ...
                       -(x_main.n.NX+x_sub(k).n.NPFree), ...
                       x_main.n.NZ0, x_sub(k).n.NLoad), ...
               sparse(x_main.n.NZ0,x_sub(k).n.NaL_k), ...
               spdiags(ones(x_sub(k).n.NPFree,1), -x_main.n.NX, ...
                       x_main.n.NZ0, x_sub(k).n.NPFree), ...
               spdiags(ones(x_sub(k).n.NLoad,1), ...
                       -(x_main.n.NX+x_sub(k).n.NPFree), ...
                       x_main.n.NZ0, x_sub(k).n.NLoad), ...
```

```
            sparse(x_main.n.NZ0,3*(x_sub(k).n.NaH+x_sub(k).n.NaL))
            sparse(4*(x_main.n.NaH+x_main.n.NaL), ...
                   x_sub(k).n.NZ0+4*(x_sub(k).n.NaH+x_sub(k).n.NaL))];

      CouplT = [sparse(x_sub(k).n.NZ0,x_main.n.NZ0+ ...
                       4*(x_main.n.NaH+x_main.n.NaL))
                sparse(x_sub(k).n.NaH+x_sub(k).n.NaL, ...
                       x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL))
                sparse(x_sub(k).n.NaH, x_main.n.NX), ...
                spdiags(-ones(x_main.n.NPFree+x_main.n.NLoad,1), ...
                        -(x_sub(k).n.NPFree+x_sub(k).n.NLine), ...
                        x_sub(k).n.NaH, x_main.n.NVar-x_main.n.NX), ...
                sparse(x_sub(k).n.NaH, x_main.n.NBus+x_main.n.NLine), ...
                sparse(x_sub(k).n.NaH, 4*(x_main.n.NaH+x_main.n.NaL))
                sparse(x_sub(k).n.NaL, x_main.n.NX), ...
                spdiags(ones(x_main.n.NPFree+x_main.n.NLoad,1), ...
                    -(x_sub(k).n.NPFree+x_sub(k).n.NLoad+x_sub(k).n.NLine), ...
                        x_sub(k).n.NaL, x_main.n.NVar-x_main.n.NX), ...
                sparse(x_sub(k).n.NaL, x_main.n.NBus+x_main.n.NLine), ...
                sparse(x_sub(k).n.NaL, 4*(x_main.n.NaH+x_main.n.NaL))
                sparse(2*(x_sub(k).n.NaH+x_sub(k).n.NaL),...
                       x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL)) ];

% Form W matrix and vector g of all subproblems.

      if exist('W_sub')
         W_sub = [W_sub  sparse(length(W_sub),length(W))
                  sparse(length(W),length(W_sub))  W];
         g0_sub = [g0_sub
                   g0    ];
         W_coupl = [W_coupl Coupl];
         W_couplT = [W_couplT; CouplT];
      else
         W_sub = W;
         g0_sub = g0;
         W_coupl = Coupl;
         W_couplT = CouplT;
      end %if exist W_sub

% Form x_all for deltaZ_all.

      x_all.ind.Angl = [x_all.ind.Angl; x_sub(k).ind.Angl+next];
      x_all.ind.PG = [x_all.ind.PG; x_sub(k).ind.PG+next];
      x_all.ind.PL = [x_all.ind.PL; x_sub(k).ind.PL+next];
      x_all.ind.Ps = [x_all.ind.Ps; x_sub(k).ind.Ps+next];
      x_all.ind.Z0 = [x_all.ind.Z0; x_sub(k).ind.Z0+next];
      x_all.ind.LambdaP = [x_all.ind.LambdaP; x_sub(k).ind.LambdaP+next];
      x_all.ind.LambdaPs = [x_all.ind.LambdaPs; x_sub(k).ind.LambdaPs+next];
      x_all.ind.Slack = [x_all.ind.Slack; x_sub(k).ind.Slack+next];
      x_all.ind.Slackbus = [x_all.ind.Slackbus; x_sub(k).ind.Slackbus];

      x_sub(k).ind.MuH = (x_sub(k).n.NZ0+1 : x_sub(k).n.NZ0+x_sub(k).n.NaH)';
      x_sub(k).ind.MuL = (x_sub(k).n.NZ0+x_sub(k).n.NaH+1 :  ...
                          x_sub(k).n.NZ0+x_sub(k).n.NaH+x_sub(k).n.NaL)';
      x_sub(k).ind.MurhoH = (x_sub(k).n.NZ0+x_sub(k).n.NaH+x_sub(k).n.NaL+1 :...
                             x_sub(k).n.NZ0+2*x_sub(k).n.NaH+x_sub(k).n.NaL)';
      x_sub(k).ind.MurhoL = (x_sub(k).n.NZ0+2*x_sub(k).n.NaH+ ...
                             x_sub(k).n.NaL+1 : ...
                             x_sub(k).n.NZ0+2*(x_sub(k).n.NaH+x_sub(k).n.NaL))';
      x_sub(k).ind.sH = (x_sub(k).n.NZ0+2*(x_sub(k).n.NaH+ ...
                         x_sub(k).n.NaL)+1 : x_sub(k).n.NZ0+ ...
                         3*x_sub(k).n.NaH+2*x_sub(k).n.NaL)';
      x_sub(k).ind.sL = (x_sub(k).n.NZ0+3*x_sub(k).n.NaH+ ...
                         2*x_sub(k).n.NaL+1 : ...
                         x_sub(k).n.NZ0+3*(x_sub(k).n.NaH+x_sub(k).n.NaL))';
      x_sub(k).ind.rhoH = (x_sub(k).n.NZ0+3*(x_sub(k).n.NaH+ ...
                           x_sub(k).n.NaL)+1 : ...
                           x_sub(k).n.NZ0+4*x_sub(k).n.NaH+3*x_sub(k).n.NaL)';
```

```
      x_sub(k).ind.rhoL = (x_sub(k).n.NZO+4*x_sub(k).n.NaH+ ...
                           3*x_sub(k).n.NaL+1 : ...
                           x_sub(k).n.NZO+4*(x_sub(k).n.NaH+x_sub(k).n.NaL))';

      x_all.ind.MuH = [x_all.ind.MuH; x_sub(k).ind.MuH+next];
      x_all.ind.MuL = [x_all.ind.MuL; x_sub(k).ind.MuL+next];
      x_all.ind.MurhoH = [x_all.ind.MurhoH; x_sub(k).ind.MurhoH+next];
      x_all.ind.MurhoL = [x_all.ind.MurhoL; x_sub(k).ind.MurhoL+next];
      x_all.ind.sH = [x_all.ind.sH; x_sub(k).ind.sH+next];
      x_all.ind.sL = [x_all.ind.sL; x_sub(k).ind.sL+next];
      x_all.ind.rhoH = [x_all.ind.rhoH; x_sub(k).ind.rhoH+next];
      x_all.ind.rhoL = [x_all.ind.rhoL; x_sub(k).ind.rhoL+next];

      x_all.var.Z = [x_all.var.Z; x_sub(k).var.Z];
      x_all.n.NaH = x_all.n.NaH+x_sub(k).n.NaH;
      x_all.n.NaL = x_all.n.NaL+x_sub(k).n.NaL;

      next = next + x_sub(k).n.NZO+4*(x_sub(k).n.NaH+x_sub(k).n.NaL);

  end %for k
%*****************************************************************************
% Form huge W matrix and vector g which have all subproblems and main problem.

  if isempty(x_sub)
     W_coupl  = [];
     W_couplT = [];
     W_sub    = [];
     g0_sub   = [];
     norm_g_sub = [];
  end %if

  W_all = [W_main     W_coupl
           W_couplT   W_sub];
  g0_all = [g0_main
            g0_sub];

  g_all = W_all*x_all.var.Z + g0_all;

  x_all.var.MuH = x_all.var.Z(x_all.ind.MuH);
  x_all.var.MuL = x_all.var.Z(x_all.ind.MuL);
  x_all.var.MurhoH = x_all.var.Z(x_all.ind.MurhoH);
  x_all.var.MurhoL = x_all.var.Z(x_all.ind.MurhoL);
  x_all.var.sH = x_all.var.Z(x_all.ind.sH);
  x_all.var.sL = x_all.var.Z(x_all.ind.sL);
  x_all.var.rhoH = x_all.var.Z(x_all.ind.rhoH);
  x_all.var.rhoL = x_all.var.Z(x_all.ind.rhoL);

%=============================================================================
```

```
  function[x_sub] = Limits(x_main,x_sub,k)

% function[x_sub] = Limits(x_main,x_sub,k)
%
% Vector of upper and lower limits.
% Limits of PG, PL, and Ps are different in main and subproblems.
% They have to be reset every iteration.

%==========================================================================
% Upper limits (including coupling constraints).

  x_sub(k).lim.PGmax = x_main.lim.PGmax;
  x_sub(k).lim.PLmax = x_main.var.PL;
% x_sub(k).lim.Pmax  = x_sub(k).lim.Pmax;   % Come from Modified.

  x_sub(k).lim.Max = [x_sub(k).lim.PGmax
                      x_sub(k).lim.PLmax
                      x_sub(k).lim.Pmax
                      x_sub(k).lim.MWup
                      zeros(x_sub(k).n.NLoad,1)];

%--------------------------------------------------------------------------
% Lower limits (including coupling constraints).

  x_sub(k).lim.PGmin = x_main.lim.PGmin;
  x_sub(k).lim.PLmin = zeros(x_sub(k).n.NLoad,1);
  x_sub(k).lim.Pmin  = -x_sub(k).lim.Pmax;

  x_sub(k).lim.Min = [x_sub(k).lim.PGmin
                      x_sub(k).lim.PLmin
                      x_sub(k).lim.Pmin
                     -x_sub(k).lim.MWdn
                     -x_sub(k).lim.MWmaxInt];

%**************************************************************************
```

```
  function[Z,W,g0,norm_g] = Derivatives(x,k)

% function[Z,W,g0,norm_g] = Derivatives(x,k)
%
% This funtion computes first and second order derivatives of both main
% and sub problems, and then forms W matrix and vector g.

%=========================================================================
% Form W0 matrix.
% Frist, form an arbitrary matrix which is derivatives of Pij wrt Theta.
% This matrix would be part of A matrix if Pij were a function.

  APij = sparse([x.ind.f;x.ind.t], [1:x.n.NLine 1:x.n.NLine]', ...
                [-x.m.Bline_v;x.m.Bline_v], x.n.NBus, x.n.NLine);
  APij(x.ind.Slackbus,:) = [];

  dPLdLambdaP = sparse(x.n.NLoad,x.n.NBus);
  dPLdLambdaP(1:x.n.NLoad,x.ind.Loadbus) = speye(x.n.NLoad,x.n.NLoad);

  if k==0   % Part of MC of PG and PL which is 2*Gamma*P.
     W0_PG = diag(x.cost.d2*x.m.Pi0);
     W0_PL = diag(-x.cost.d2L*x.m.Pi0);
  elseif (k > 0 & k <= x.n.K)
     W0_PG = diag(x.cost.d2)*x.m.Pi(k) + diag(x.cost.d2R)*x.m.Pi(k);
     W0_PL = diag(-x.cost.d2L)*x.m.Pi(k) + diag(-x.cost.d2I)*x.m.Pi(k);
  end %if k

  W0 = [sparse(x.n.NX,x.n.NX+x.n.NPFree+x.n.NLoad+x.n.NLine)   x.m.B   APij
        sparse(x.n.NPFree,x.n.NX)             W0_PG ...
        sparse(x.n.NPFree,x.n.NLoad)  sparse(x.n.NPFree,x.n.NLine) ...
       -speye(x.n.NPFree,x.n.NBus)  sparse(x.n.NPFree,x.n.NLine)
        sparse(x.n.NLoad,x.n.NX+x.n.NPFree)  W0_PL ...
        sparse(x.n.NLoad,x.n.NLine)  dPLdLambdaP sparse(x.n.NLoad,x.n.NLine)
        sparse(x.n.NLine,x.n.NX+x.n.NPFree+x.n.NLoad+x.n.NLine+x.n.NBus) ...
       -speye(x.n.NLine,x.n.NLine)
        x.m.B'  -speye(x.n.NBus,x.n.NPFree)  dPLdLambdaP' ...
        sparse(x.n.NBus,x.n.NLine+x.n.NBus+x.n.NLine)
        APij'  sparse(x.n.NLine,x.n.NPFree+x.n.NLoad)  ...
       -speye(x.n.NLine,x.n.NLine)   sparse(x.n.NLine,x.n.NBus+x.n.NLine)  ];

%*************************************************************************
% Form A matrix for both FP and IP.
% For subproblems, there are extra columns of coupling constraints in A.

  A = [sparse(x.n.NX,2*x.n.NPFree+2*x.n.NLoad+x.n.NLine)
       speye(x.n.NPFree+x.n.NLoad,x.n.NPFree+x.n.NLoad) ...
       sparse(x.n.NPFree+x.n.NLoad,x.n.NLine) ...
       speye(x.n.NPFree+x.n.NLoad,x.n.NPFree+x.n.NLoad)
       sparse(x.n.NLine,x.n.NPFree+x.n.NLoad)  speye(x.n.NLine,x.n.NLine) ...
       sparse(x.n.NLine,x.n.NPFree+x.n.NLoad)
       sparse(x.n.NBus+x.n.NLine,2*x.n.NPFree+2*x.n.NLoad+x.n.NLine)        ];


%*************************************************************************
% Form W matrix.

  AH = A(:,x.ind.aH);
  AL = A(:,x.ind.aL);

  W = [ W0  AH  -AL  sparse(x.n.NZ0, 3*(x.n.NaH+x.n.NaL))
        sparse(x.n.NaH+x.n.NaL, x.n.NZ0) ...
           -speye(x.n.NaH+x.n.NaL)  -speye(x.n.NaH+x.n.NaL) ...
            sparse(x.n.NaH+x.n.NaL, 2*(x.n.NaH+x.n.NaL))
        [AH'; -AL']  sparse(x.n.NaH+x.n.NaL, 2*(x.n.NaH+x.n.NaL)) ...
            speye(x.n.NaH+x.n.NaL)  -speye(x.n.NaH+x.n.NaL)
        sparse(2*(x.n.NaH+x.n.NaL), x.n.NZ0) ...
            spdiags([x.var.sH; x.var.sL; x.var.rhoH; x.var.rhoL], ...
                    0, 2*(x.n.NaH+x.n.NaL), 2*(x.n.NaH+x.n.NaL)) ...
```

```
          spdiags([x.var.MuH; x.var.MuL; x.var.MurhoH; x.var.MurhoL], ...
                  0, 2*(x.n.NaH+x.n.NaL), 2*(x.n.NaH+x.n.NaL))          ];

%*************************************************************************
% Form vector g0.
% x.sc.dSCdPG, x.sc.dSCdPL come from subproblem.

  if k==0   % Part of MC of PG and PL which is only Beta.
     if x.n.K==0
        BetaRPu_sub = sparse(x.n.NPFree,x.n.NLine);
        BetaIPu_sub = sparse(x.n.NLoad,x.n.NLine);
     else
        BetaRPu_sub = kron(x.cost.BetaRPu,ones(1,x.n.K)); % BetaR of every sub.
        BetaIPu_sub = kron(x.cost.BetaIPu,ones(1,x.n.K)); % BetaI of every sub.
     end %if x.n.K==0
     g0_PG = x.m.Pi0*x.cost.BetaPu - BetaRPu_sub*x.m.Pi;
     g0_PL = -x.m.Pi0*x.cost.BetaLPu + BetaIPu_sub*x.m.Pi;
  elseif (k > 0 & k <= x.n.K)
     g0_PG = x.cost.BetaPu*x.m.Pi(k) + x.cost.BetaRPu*x.m.Pi(k);
     g0_PL =-x.cost.BetaLPu*x.m.Pi(k) - x.cost.BetaIPu*x.m.Pi(k);
  end %if k==0

  g0 = [ sparse(x.n.NX,1)
         g0_PG
         g0_PL
         sparse(x.n.NLine,1)
         sparse(x.n.NBus,1)
         sparse(x.n.NLine,1) ];

%*************************************************************************
% Form initial vector Z only for first iteration of SCOPF and every
% first iteration of each subproblem.

  if x.Iter==1
     x.var.Z = [ x.var.Angl
                 x.var.PG
                 x.var.PL
                 x.var.Ps
                 x.var.LambdaP
                 x.var.LambdaPs
                 x.var.MuH
                 x.var.MuL
                 x.var.MurhoH
                 x.var.MurhoL
                 x.var.sH
                 x.var.sL
                 x.var.rhoH
                 x.var.rhoL     ];

     x.var.Z(x.ind.Slack,:) = [];      % Take out angle of slack bus.
  end % if

%*************************************************************************
% Modify g0 (we are using IPRho here) and form vector g.

  g0 = [ g0
         x.init.cost.rho*ones(x.n.NaH,1)
         x.init.cost.rho*ones(x.n.NaL,1)
        -x.lim.Max(x.ind.aH)
         x.lim.Min(x.ind.aL)
        -x.var.MuH.*x.var.sH - x.init.m.sig*x.var.Nu(ones(x.n.NaH,1))
        -x.var.MuL.*x.var.sL - x.init.m.sig*x.var.Nu(ones(x.n.NaL,1))
        -x.var.MurhoH.*x.var.rhoH - x.init.m.sig*x.var.Nu(ones(x.n.NaH,1))
        -x.var.MurhoL.*x.var.rhoL - x.init.m.sig*x.var.Nu(ones(x.n.NaL,1))  ];

  g = W*x.var.Z + g0;

  Z = x.var.Z;
```

```
%*****************************************************************************
% Take a good look at g.

  g_var = g(1:x.n.NVar);
  g_lambda = g(x.n.NVar+1:x.n.NVar+2*x.n.NBus);
  g_Crho = g(x.n.NZO+1:x.n.NZO+x.n.NaH+x.n.NaL);
  g_slack = g(x.n.NZO+x.n.NaH+x.n.NaL+1:x.n.NZO+2*(x.n.NaH+x.n.NaL));
  g_comp = g(x.n.NZO+2*(x.n.NaH+x.n.NaL)+1:x.n.NZO+4*(x.n.NaH+x.n.NaL));

  norm_g.g_var = norm(g_var);
  norm_g.g_lambda = norm(g_lambda);
  norm_g.g_Crho = norm(g_Crho);
  norm_g.g_slack = norm(g_slack);
  norm_g.g_comp = norm(g_comp);

%*****************************************************************************
```

```
  function[x_all,done] = IPRho(x_all,W,g,done)

% function[x_all,done] = IPRho(x_all,W,g,done)
%
% Compute the problem using Primal-Dual Interior-Point Method with additional
% penalty slacks called "rho" to ensure the feasible solutions of the
% subproblems.

%=============================================================================
% Check to see if norm(g) is close to zero.

% g_norm = norm(g);
  fprintf('\n|g| = %8.4g', norm(g));

%-----------------------------------------------------------------------------
% Compute deltaZ.

  deltaZ = -W\g;

  deltaMuH = deltaZ(x_all.ind.MuH);
  deltaMuL = deltaZ(x_all.ind.MuL);
  deltaMurhoH = deltaZ(x_all.ind.MurhoH);
  deltaMurhoL = deltaZ(x_all.ind.MurhoL);

  deltasH = deltaZ(x_all.ind.sH);
  deltasL = deltaZ(x_all.ind.sL);
  deltarhoH = deltaZ(x_all.ind.rhoH);
  deltarhoL = deltaZ(x_all.ind.rhoL);

%-----------------------------------------------------------------------------
% Find binding constraints.

  isHNeg     = find(deltasH < 0);      % ind of sH that might be negative.
  isLNeg     = find(deltasL < 0);      % ind of sL that might be negative.
  irhoHNeg   = find(deltarhoH < 0);    % ind of rhoH that might be negative.
  irhoLNeg   = find(deltarhoL < 0);    % ind of rhoL that might be negative.
  iMuHNeg    = find(deltaMuH < 0);     % ind of MuH that might be negative.
  iMuLNeg    = find(deltaMuL < 0);     % ind of MuL that might be negative.
  iMurhoHNeg = find(deltaMurhoH < 0);  % ind of MurhoH that might be negative.
  iMurhoLNeg = find(deltaMurhoL < 0);  % ind of MurhoL that might be negative.

% Set value of constant that prevents variables from being zero.

  load const.dat

% Find Primal step.

  alpha_p0 = min([-x_all.var.sH(isHNeg)./deltasH(isHNeg)
                  -x_all.var.sL(isLNeg)./deltasL(isLNeg)
                  -x_all.var.rhoH(irhoHNeg)./deltarhoH(irhoHNeg)
                  -x_all.var.rhoL(irhoLNeg)./deltarhoL(irhoLNeg)] );
  alpha_p = min([1, const*alpha_p0]);
  fprintf('\nalpha_p = %8.4g', alpha_p);

% Find Dual step.

  alpha_d0 = min([-x_all.var.MuH(iMuHNeg)./deltaMuH(iMuHNeg)
                  -x_all.var.MuL(iMuLNeg)./deltaMuL(iMuLNeg)
                  -x_all.var.MurhoH(iMurhoHNeg)./deltaMurhoH(iMurhoHNeg)
                  -x_all.var.MurhoL(iMurhoLNeg)./deltaMurhoL(iMurhoLNeg)] );
  alpha_d = min([1, const*alpha_d0]);
  fprintf('\nalpha_d = %8.4g\n\n', alpha_d);

% The minimum of Primal and Dual steps.

  alpha = min(alpha_p, alpha_d);

%-----------------------------------------------------------------------------
```

```
% Update Z

  x_all.var.Z0     = x_all.var.Z(x_all.ind.Z0) + alpha_p*deltaZ(x_all.ind.Z0);
  x_all.var.sH     = x_all.var.sH + alpha_p*deltasH;
  x_all.var.sL     = x_all.var.sL + alpha_p*deltasL;
  x_all.var.rhoH   = x_all.var.rhoH + alpha_p*deltarhoH;
  x_all.var.rhoL   = x_all.var.rhoL + alpha_p*deltarhoL;
  x_all.var.MuH    = x_all.var.MuH + alpha_d*deltaMuH;
  x_all.var.MuL    = x_all.var.MuL + alpha_d*deltaMuL;
  x_all.var.MurhoH = x_all.var.MurhoH + alpha_d*deltaMurhoH;
  x_all.var.MurhoL = x_all.var.MurhoL + alpha_d*deltaMurhoL;

  x_all.var.Z = zeros(length(deltaZ),1);
  x_all.var.Z(x_all.ind.Z0) = x_all.var.Z0;
  x_all.var.Z(x_all.ind.MuH) = x_all.var.MuH;
  x_all.var.Z(x_all.ind.MuL) = x_all.var.MuL;
  x_all.var.Z(x_all.ind.MurhoH) = x_all.var.MurhoH;
  x_all.var.Z(x_all.ind.MurhoL) = x_all.var.MurhoL;
  x_all.var.Z(x_all.ind.sH) = x_all.var.sH;
  x_all.var.Z(x_all.ind.sL) = x_all.var.sL;
  x_all.var.Z(x_all.ind.rhoH) = x_all.var.rhoH;
  x_all.var.Z(x_all.ind.rhoL) = x_all.var.rhoL;

%*************************************************************************
```

```
  function[x_main,x_sub] = UnPackZ(x_all,x_main,x_sub)

% function[x_main,x_sub] = UnPackZ(x_all,x_main,x_sub)
%
% Take out variables and mutilpliers from vector Z and send them to main
% or subproblems.

%=========================================================================
  x_main.var.Angl = zeros(x_main.n.NBus,1);
  ind_temp = [1:x_main.n.NBus]';
  notSlackbus = find(ind_temp~=x_main.ind.Slackbus);
  x_main.var.Angl(notSlackbus) = x_all.var.Z(x_main.ind.Angl);
  x_main.var.PG = x_all.var.Z(x_main.ind.PG);
  x_main.var.PL = x_all.var.Z(x_main.ind.PL);
  x_main.var.Ps = x_all.var.Z(x_main.ind.Ps);
  x_main.var.LambdaP = x_all.var.Z(x_main.ind.LambdaP);
  x_main.var.LambdaPs = x_all.var.Z(x_main.ind.LambdaPs);

  x_main.var.MuH = x_all.var.Z(x_main.n.NZ0+1 : x_main.n.NZ0+x_main.n.NaH);
  x_main.var.MuL = x_all.var.Z(x_main.n.NZ0+x_main.n.NaH+1 :  ...
                               x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL);
  x_main.var.MurhoH = x_all.var.Z(x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL+1 :...
                      x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL);
  x_main.var.MurhoL =x_all.var.Z(x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL+1: ...
                     x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL));
  x_main.var.sH = x_all.var.Z(x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL)+1: ...
                  x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL);
  x_main.var.sL = x_all.var.Z(x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL+1 : ...
                  x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL));
  x_main.var.rhoH =x_all.var.Z(x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL)+1: ...
                   x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL);
  x_main.var.rhoL =x_all.var.Z(x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL+1: ...
                   x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL));

  x_main.var.Z = [ x_main.var.Angl
                   x_main.var.PG
                   x_main.var.PL
                   x_main.var.Ps
                   x_main.var.LambdaP
                   x_main.var.LambdaPs
                   x_main.var.MuH
                   x_main.var.MuL
                   x_main.var.MurhoH
                   x_main.var.MurhoL
                   x_main.var.sH
                   x_main.var.sL
                   x_main.var.rhoH
                   x_main.var.rhoL     ];
  x_main.var.Z(x_main.ind.Slack,:) = [];  % Take out angle of slack bus.

  next = x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL);

  x_main.Iter = x_main.Iter+1;

  for k = 1:x_main.n.K,
      x_sub(k).var.Angl = zeros(x_sub(k).n.NBus,1);
      ind_temp = [1:x_sub(k).n.NBus]';
      notSlackbus = find(ind_temp~=x_sub(k).ind.Slackbus);
      x_sub(k).var.Angl(notSlackbus) = x_all.var.Z(x_sub(k).ind.Angl+next);
      x_sub(k).var.PG = x_all.var.Z(x_sub(k).ind.PG+next);
      x_sub(k).var.PL = x_all.var.Z(x_sub(k).ind.PL+next);
      x_sub(k).var.Ps = x_all.var.Z(x_sub(k).ind.Ps+next);
      x_sub(k).var.LambdaP = x_all.var.Z(x_sub(k).ind.LambdaP+next);
      x_sub(k).var.LambdaPs = x_all.var.Z(x_sub(k).ind.LambdaPs+next);

      x_sub(k).var.MuH = x_all.var.Z(x_sub(k).ind.MuH+next);
      x_sub(k).var.MuL = x_all.var.Z(x_sub(k).ind.MuL+next);
      x_sub(k).var.MurhoH = x_all.var.Z(x_sub(k).ind.MurhoH+next);
```

```
      x_sub(k).var.MurhoL = x_all.var.Z(x_sub(k).ind.MurhoL+next);
      x_sub(k).var.sH = x_all.var.Z(x_sub(k).ind.sH+next);
      x_sub(k).var.sL = x_all.var.Z(x_sub(k).ind.sL+next);
      x_sub(k).var.rhoH = x_all.var.Z(x_sub(k).ind.rhoH+next);
      x_sub(k).var.rhoL = x_all.var.Z(x_sub(k).ind.rhoL+next);

      x_sub(k).var.Z = [ x_sub(k).var.Angl
                         x_sub(k).var.PG
                         x_sub(k).var.PL
                         x_sub(k).var.Ps
                         x_sub(k).var.LambdaP
                         x_sub(k).var.LambdaPs
                         x_sub(k).var.MuH
                         x_sub(k).var.MuL
                         x_sub(k).var.MurhoH
                         x_sub(k).var.MurhoL
                         x_sub(k).var.sH
                         x_sub(k).var.sL
                         x_sub(k).var.rhoH
                         x_sub(k).var.rhoL     ];

      x_sub(k).var.Z(x_sub(k).ind.Slack,:) = [];  % Take out angle of slack bus.

      next = next + x_sub(k).n.NZO+4*(x_sub(k).n.NaH+x_sub(k).n.NaL);

      x_sub(k).Iter = x_sub(k).Iter+1;

   end %for k

%=========================================================================
```

```
  function[] = Print(x,k,look_g,norm_g)

% function[] = Print(x,k,look_g,norm_g)
%
% This function displays the variables and constraints during each iteration.

%=========================================================================
% Prepare variables for printing out.

  PG = zeros(1,x.n.NBus);
  PG(x.ind.PFree) = x.var.PG;
  PL = zeros(1,x.n.NBus);
  PL(x.ind.Loadbus) = x.var.PL;

  if k==0
     LambdaP_print = x.var.LambdaP/x.m.Pi0;
     LambdaPs_print = x.var.LambdaPs/x.m.Pi0;
     MuH_print = x.var.MuH/x.m.Pi0;
     MuL_print = x.var.MuL/x.m.Pi0;
     MC_gen = zeros(1,x.n.NBus);
     MC_gen(x.ind.PFree) = x.cost.MC;
     MB_load = zeros(1,x.n.NBus);
     MB_load(x.ind.Loadbus) = x.cost.MB;
     Cost = x.cost.Obj/x.m.Pi0;

  else
     Pi_k = x.m.Pi(k);
     LambdaP_print = x.var.LambdaP/Pi_k;
     LambdaPs_print = x.var.LambdaPs/Pi_k;
     MuH_print = x.var.MuH/Pi_k;
     MuL_print = x.var.MuL/Pi_k;
     MC_gen = zeros(1,x.n.NBus);
     MC_gen(x.ind.PFree) = x.cost.MC;
     MB_load = zeros(1,x.n.NBus);
     MB_load(x.ind.Loadbus) = x.cost.MB;
     Cost = x.cost.Obj/Pi_k;
  end %if

  PG_total = sum(x.var.PG);
  PL_total = sum(x.var.PL);
  Loss = PG_total - PL_total;

%=========================================================================
  disp('============================Variables==============================')
  disp(sprintf('Calculated values at the end of iteration %4i are:',x.Iter-1));
  disp([ sprintf('Bus number ')  sprintf(' %6d  ',[1:x.n.NBus])]);
  disp([ sprintf('Angl(rad) =')  sprintf(' %8.4f',x.var.Angl)]);
  disp([ sprintf('Angl(deg) =')  sprintf(' %8.4f',x.var.Angl*180/pi)]);
  disp([ sprintf('PG        =')  sprintf(' %8.4f',PG)]);
  disp([ sprintf('PL        =')  sprintf(' %8.4f',PL)]);
  disp([ sprintf('PI        =')  sprintf(' %8.4f',x.var.PI)]);
  disp([ sprintf('LambdaP   =')  sprintf(' %8.4f',full(LambdaP_print))]);
  disp([ sprintf('MC_gen    =')  sprintf(' %8.4f',MC_gen)]);
  disp([ sprintf('MB_load   =')  sprintf(' %8.4f',MB_load)]);
  fprintf(1,'%3i. Ps(%2i,%2i) = %9.4f      LambdaPs = %9.4f \n',...
          [(1:x.n.NLine)' x.ind.f x.ind.t x.var.Ps LambdaPs_print]');
  disp([ sprintf('PG_total  =')  sprintf(' %12.4f $/hour',PG_total)]);
  disp([ sprintf('PL_total  =')  sprintf(' %12.4f $/hour',PL_total)]);
  disp([ sprintf('Loss      =')  sprintf(' %12.4f $/hour',Loss)]);
  disp([ sprintf('Cost      =')  sprintf(' %12.4f $/hour',Cost)]);
  disp('=========================================================================')
%=========================================================================
% Print Binding constraints in FP.

if ~x.flag.ip
  H_type = x.print.Type(x.ind.aH);
  L_type = x.print.Type(x.ind.aL);
  H_name = x.print.Name(x.ind.aH,:);
```

```
  L_name = x.print.Name(x.ind.aL,:);
  H_num  = x.print.Number(x.ind.aH,:);
  L_num  = x.print.Number(x.ind.aL,:);

  if ~isempty(H_type)
    PsH_bind = find(H_type==3);       % Look for Ps binding at upper limit.
    if ~isempty(PsH_bind)
       PsH_num = H_num(PsH_bind);     % Look up for the num of PsH.
       PsH_f = x.main.ind.f(PsH_num); % This Ps is from line ....
       PsH_t = x.main.ind.t(PsH_num); % ... to line ....
    end %if
  else
     PsH_bind = [];
  end %if
  if ~isempty(L_type)
    PsL_bind = find(L_type==3);       % Look for Ps binding at lower limit.
    if ~isempty(PsL_bind)
       PsL_num = L_num(PsL_bind);     % Look up for the num of PsL.
       PsL_f = x.main.ind.f(PsL_num); % This Ps is from line ....
       PsL_t = x.main.ind.t(PsL_num); % ... to line ....
    end %if
  else
     PsL_bind = [];
  end %if

  disp('================Binding Constratins in FP=========================')
  disp('Binding constraints on upper limits are: ')
  disp([ sprintf('Upper Limits Number:') sprintf('%8i', x.ind.aH)]);
  disp([ sprintf('Upper Limits       :     ') ...
         sprintf('%s',[H_name num2str(H_num,'%-4i')]')]);
  disp([ sprintf('Mu of Upper Limits =') sprintf('%8.4f', full(MuH_print))]);
  if ~isempty(PsH_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsH_num PsH_f PsH_t]')
  end %if
  disp('-------------------------------------------------------------------')
  disp('Binding constraints on lower limits are: ')
  disp([ sprintf('Lower Limits Number:') sprintf('%8i', x.ind.aL)]);
  disp([ sprintf('Lower Limits       :     ') ...
         sprintf('%s',[L_name num2str(L_num,'%-4i')]')]);
  disp([ sprintf('Mu of Lower Limits =') sprintf('%8.4f', full(MuL_print))]);
  if ~isempty(PsL_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsL_num PsL_f PsL_t]')
  end %if
  disp('===================================================================')

end %if ~x.flag.ip

%=============================================================================
% Print information in IP.

if x.flag.ip
  fprintf('Nu = %8.4g\n',x.var.Nu);

  if x.var.Nu <= x.init.printIP

  disp('============================Constraints============================')

  ind_aH_bind = find(x.var.MuH > 1e-4);
  ind_aL_bind = find(x.var.MuL > 1e-4);
  aH_bind = x.ind.aH(ind_aH_bind);
  aL_bind = x.ind.aL(ind_aL_bind);

  H_type = x.print.Type(aH_bind,:);
  L_type = x.print.Type(aL_bind,:);
  H_name = x.print.Name(aH_bind,:);
  L_name = x.print.Name(aL_bind,:);
  H_num  = x.print.Number(aH_bind,:);
  L_num  = x.print.Number(aL_bind,:);
```

```
  if ~isempty(H_type)
    PsH_bind = find(H_type==3);       % Look for Ps binding at upper limit.
    if ~isempty(PsH_bind)
       PsH_num = H_num(PsH_bind);     % Look up for the num of PsH.
       PsH_f = x.init.ind.f(PsH_num); % This Ps is from line ....
       PsH_t = x.init.ind.t(PsH_num); % ... to line ....
    end %if
  else
     PsH_bind = [];
  end %if
  if ~isempty(L_type)
    PsL_bind = find(L_type==3);       % Look for Ps binding at lower limit.
    if ~isempty(PsL_bind)
       PsL_num = L_num(PsL_bind);     % Look up for the num of PsL.
       PsL_f = x.init.ind.f(PsL_num); % This Ps is from line ....
       PsL_t = x.init.ind.t(PsL_num); % ... to line ....
    end %if
  else
     PsL_bind = [];
  end %if

if ~isempty(aH_bind)   % Print this portion if there is any nonzero MuH.
  disp('Binding constraints on upper limits are: ')
  disp([ sprintf('Upper Limits Number:')  sprintf('%9i',aH_bind)]);
  disp([ sprintf('Upper Limits       :     ')  ...
         sprintf('%s',[H_name num2str(H_num,'%-5i')]')]);
  disp([ sprintf('Mu of Upper Limits    =')  ...
         sprintf('%9.4f',MuH_print(ind_aH_bind))]);
  disp([ sprintf('Slack of Upper Limits =')  ...
         sprintf('%9.4f',x.var.sH(ind_aH_bind))]);
  if ~isempty(PsH_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsH_num PsH_f PsH_t]')
  end %if
end %if ~isempty(aH_bind)
if ~isempty(aL_bind)   % Print this portion if there is any nonzero MuL.
  disp('------------------------------------------------------------------------')
  disp('Binding constraints on lower limits are: ')
  disp([ sprintf('Lower Limits Number:')  sprintf('%9i',aL_bind)]);
  disp([ sprintf('Lower Limits       :     ')  ...
         sprintf('%s',[L_name num2str(L_num,'%-5i')]')]);
  disp([ sprintf('Mu of Lower Limits    =')  ...
         sprintf('%9.4f',MuL_print(ind_aL_bind))]);
  disp([ sprintf('Slack of Lower Limits =')  ...
         sprintf('%9.4f',x.var.sL(ind_aL_bind))]);
  if ~isempty(PsL_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsL_num PsL_f PsL_t]')
  end %if
end %if ~isempty(aL_bind)

  if x.flag.ip == 2    % for IP with Rho.
     ind_rhoH_bind = find(x.var.rhoH > 1e-4);
     ind_rhoL_bind = find(x.var.rhoL > 1e-4);
     rhoH_bind = x.ind.aH(ind_rhoH_bind);
     rhoL_bind = x.ind.aL(ind_rhoL_bind);

     Hrho_name = x.print.Name(rhoH_bind,:);
     Lrho_name = x.print.Name(rhoL_bind,:);
     Hrho_num  = x.print.Number(rhoH_bind,:);
     Lrho_num  = x.print.Number(rhoL_bind,:);

if ~isempty(rhoH_bind)   % Print this portion if there is any nonzero MurhoH.
    disp('------------------------------------------------------------------------')
    disp('Exceeded constraints on upper limits are: ')
    disp([ sprintf('Exceeded Upper Limits Number:')  ...
           sprintf('%9i',ind_rhoH_bind)]);
    disp([ sprintf('Exceeded Upper Limits:            ') ...
           sprintf('%s',[Hrho_name num2str(Hrho_num,'%-5i')]')]);
```

```
    disp([ sprintf('Mu of Rho on Upper Limits =')  ...
           sprintf('%9.4f',x.var.MurhoH(ind_rhoH_bind))]);
    disp([ sprintf('Rho on Upper Limits       =')  ...
           sprintf('%9.4f',x.var.rhoH(ind_rhoH_bind))]);
end %if ~isempty(rhoH_bind)
if ~isempty(rhoL_bind)   % Print this portion if there is any nonzero MurhoL.
    disp('------------------------------------------------------------------')
    disp('Exceeded constraints on lower limits are: ')
    disp([ sprintf('Exceeded Lower Limits Number:')  ...
           sprintf('%9i',ind_rhoL_bind)]);
    disp([ sprintf('Exceeded Lower Limits:            ')  ...
           sprintf('%s',[Lrho_name num2str(Lrho_num,'%-5i')]')]);
    disp([ sprintf('Mu of Rho on Lower Limits =')  ...
           sprintf('%9.4f',x.var.MurhoL(ind_rhoL_bind))]);
    disp([ sprintf('Rho on Lower Limits       =')  ...
           sprintf('%9.4f',x.var.rhoL(ind_rhoL_bind))]);
  end %if x.flag.ip ==2
end %if ~isempty(rhoL_bind)
  end %if x.var.Nu
  disp('==================================================================')
end %if x.flag.ip

%=========================================================================
% Print out pieces of g

if look_g

   if input(' Look at g_var? YES=1; NO=0; ');
      fprintf('                      |g_var| = %9.4g\n', norm_g.g_var);
   end

   if input(' Look at g_lambda? YES=1; NO=0; ');
      fprintf('                      |g_lambda| = %9.4g\n', norm_g.g_lambda);
   end

   if input(' Look at g_Crho? YES=1; NO=0; ');
      fprintf('                      |g_Crho| = %9.4g\n', norm_g.g_Crho);
   end

   if input(' Look at g_slack? YES=1; NO=0; ');
      fprintf('                      |g_slack| = %9.4g\n', norm_g.g_slack);
   end

   if input(' Look at g_comp? YES=1; NO=0; ');
      fprintf('                      |g_comp| = %9.4g\n', norm_g.g_comp);
   end

disp('==================================================================')
end %if look_g

%=========================================================================
```

A.2 ESCOPF Program for AC Model

```
  function[]=ESCOPF()
%
% function[]=ESCOPF()
%
% Solve Expected Security Cost Optimal Power Flow using decomposition method.

%=========================================================================
% Read data from ".cdf", ".gdf", and ".ldf".
% Then set up initial values.

  clear
  [o] = ReadDATA;
  [x_main] = Initial(o);              % Set initial values for MAIN.

%*****************************************
pause
  if x_main.n.K==0          % In case of no contingency (base case OPF).
     x_sub = [];
  else
     for k = 1:x_main.n.K,
         x_sub(k) = Modified(o,x_main,k); % Set initial values for SUB.
     end %for k
  end %if

%************WHILE LOOP*********************
  converged = 0;
  while (~converged)

%-----------------------------------------
% Compute W matrix of main problem.

  [x_main,norm_g_main] = PowerFlow(x_main,0);

%-----------------------------------------
% Compute W matrices and set up indicies of the whole problem.

  [x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub);

%-----------------------------------------
% Call subroutine to compute deltaZ of the huge problem.
% In this program, use only IPRho.

  [x_all,converged] = IPRho(x_all,W_all,g_all,converged);

%-----------------------------------------
% Check for Neighbourhood of Central path.

  Nu = [x_all.var.MuH; x_all.var.MuL; x_all.var.MurhoH; x_all.var.MurhoL].* ...
       [x_all.var.sH;  x_all.var.sL;  x_all.var.rhoH;   x_all.var.rhoL  ];
  r = 0.25;
  N2_r = norm(Nu- ...
    x_main.var.Nu*ones(x_all.n.NaH+x_all.n.NaL+x_all.n.NaH+x_all.n.NaL,1)) ...
              -r*x_main.var.Nu;
  fprintf('N2_r = %8.4g \n',N2_r);
  fprintf('N_inf = %8.4g \n',norm(Nu,inf));

% Set flag to tell IP to calculate new Nu when we are in neighborhood of Nu.

  x_main.flag.newNu = 0;          % reset flag every time.
  if N2_r<= 0;
     x_main.flag.newNu = 1;
%    done = input(sprintf('Quit this problem? YES=1;NO=0 : '));
  end %if

%-----------------------------------------
```

```
% Calculate new Nu (or not).

  g_tol = 1e-6;   % Tolerance for gradient.
  hold_Nu = 0;    % Keep changing Nu to smaller value when flag.newNu is 1.

  if x_main.var.Nu <= x_main.var.NuLast
     hold_Nu = 1;              % Hold Nu to this value and try to get g to 0.
     if (norm(g_all)<=g_tol)   % Quit IP when g is this small.
         converged = 1;
         disp('*****************')
         disp('Program converges')
         disp('*****************')
     end % if (norm(g_all ...
  end %if x_main.var.Nu<= ...

  if (x_main.flag.newNu & ~hold_Nu)
     x_main.var.Nu = x_main.var.NuFac*x_main.var.Nu;
     for k = 1:x_main.n.K,
         x_sub(k).var.Nu = x_sub(k).var.NuFac*x_sub(k).var.Nu;
     end %for k
  end %if x_main.flag.newNu

%-------------------------------------------
% Unpack Z

  [x_main,x_sub] = UnPackZ(x_all,x_main,x_sub);

%-------------------------------------------
% Print out results for each iteration.

% look_g = input(sprintf(' Want to see gradient? YES=1; NO=0: '));
  look_g = 0;   % Don't need to look at gradient anymore.

% Objective function.

  CostPG0 = (x_main.cost.Alpha) + ...
            (x_main.cost.Beta).* (x_main.m.SBase*x_main.var.PG) + ...
            (x_main.cost.Gamma).*(x_main.m.SBase*x_main.var.PG).^2;
  Benefit0 = (x_main.cost.AlphaL) + ...
             (x_main.cost.BetaL).*(x_main.m.SBase*x_main.var.PL) + ...
             (x_main.cost.GammaL).*(x_main.m.SBase*x_main.var.PL).^2;
  CostRho0 = x_main.cost.rho*[x_main.var.rhoH; x_main.var.rhoL];

  x_main.cost.Obj = x_main.m.Pi0*( sum(CostPG0)-sum(Benefit0)+sum(CostRho0) );

% Interrupted load.

  x_main.var.PI = zeros(x_main.n.NBus,1);

  disp('Print statement for MAIN');
  if converged
     pause
  end % if converged
  Print(x_main,0,look_g,norm_g_main);

%*****************************************
  for k = 1:x_main.n.K,
%     look_g = input(sprintf(' Want to see gradient? YES=1; NO=0: '));
      look_g = 0;

% Objective function.
      CostPGk = (x_sub(k).cost.Alpha) + ...
                (x_sub(k).cost.Beta).* (x_sub(k).m.SBase*x_sub(k).var.PG) + ...
                (x_sub(k).cost.Gamma).*(x_sub(k).m.SBase*x_sub(k).var.PG).^2;
      Benefitk = (x_sub(k).cost.AlphaL) + ...
                 (x_sub(k).cost.BetaL).*(x_sub(k).m.SBase*x_sub(k).var.PL) + ...
                 (x_sub(k).cost.GammaL).*(x_sub(k).m.SBase*x_sub(k).var.PL).^2;
      CostIntr = (x_sub(k).cost.AlphaI) + ...
```

```
                (x_sub(k).cost.BetaI).* ...
                (x_sub(k).m.SBase*(x_main.var.PL-x_sub(k).var.PL)) + ...
                (x_sub(k).cost.GammaI).* ...
                (x_sub(k).m.SBase*(x_main.var.PL-x_sub(k).var.PL));
      CostRhok = x_sub(k).cost.rho*[x_sub(k).var.rhoH; x_sub(k).var.rhoL];

      x_sub(k).cost.Obj = x_sub(k).m.Pi(k)*( sum(CostPGk)-sum(Benefitk)+ ...
                                             sum(CostIntr)+sum(CostRhok) );

% Interrupted load.

      PI = x_main.var.PL - x_sub(k).var.PL;
      x_sub(k).var.PI = zeros(x_main.n.NBus,1);
      x_sub(k).var.PI(x_sub(k).ind.Loadbus) = PI;

      fprintf(1,'Print statement for SUBPROBLEM  %2i which is line (%2i,%2i).\n',[k x_main.ind.fLOut(k) x_main.inc

      if converged
         pause
      end %if converged
      Print(x_sub(k),k,look_g,norm_g_sub(k));

      x_main.cost.Obj = x_main.cost.Obj+x_sub(k).cost.Obj;

  end %for k

%*******************************************
% Print out total objective function, once ESCOPF is done.

  disp([ sprintf('Expected Security Cost  =')  sprintf(' %12.4f $/hour',x_main.cost.Obj)]);

%-------------------------------------------
  end %while

%*******************************************
```

```
  function[o] = ReadDATA()

% function[o] = ReadDATA()
%
% This function must be run at the beginning of the SCOPF program to read
% data from data files, i.e. CDF and GDF files.
% The data is read only one time through out the whole program regardless
% how many times the program calls main or subproblems.
%
% Only vectors or matrices of useful variables are passed to other .m files.
% This function also sets the neccessary convergence tolerances for SCOPF.

%=============================================================================
% Read CDF and GDF data of the selected ieee file.

  Filename = input('Case name: ','s');
  cdffile = [ Filename '.cdf' ];
  gdffile = [ Filename '.gdf' ];
  ldffile = [ Filename '.ldf' ];

% Create a structure containing bus data, branch data, and generator data
% coming out from rcdf & rgdf
  [o.bus,o.line] = ReadCDF(cdffile);
  [o.gen]        = ReadGDF(gdffile);
  [o.load]       = ReadLDF(ldffile);
  disp('DATA HAS BEEN READ IN');

%*****************************************************************************
```

```
 function[x] = Initial(o)

% function[x] = Initial(o)
%
% This function must be run at the beginning of main or subproblems
% every time those problems are called in SCOPF.
%
% This function obtains raw data from CDF and GDF files and initially
% sets up and calculates all neccessary values that need to be done once
% through out each main/subproblem.
%
% Note: Since most of the stuff in "o" won't be needed for further
% calculations, or they must be modified before used, the structure "o"
% won't be passed through the program and a new structures "x" and "m"
% are built for storing the modified stuff from "o".

%=============================================================================
% First, lets grab something from "o" that doesn't need to be evaluated,
% and put it in a new structure because we are going to throw away "o".

  x.n.NBus  = o.bus.NBus;
  x.n.NGen  = o.gen.NGen;
  x.n.NLine = o.line.NLine;
  x.n.NLoad = o.load.NLoad;

  x.var.V = o.bus.V;
  x.var.Angl = o.bus.Angl;

  x.m.Bshunt = o.bus.Bshunt;
  x.m.SBase  = o.bus.SBase;
  x.m.CBase  = o.bus.SBase;

%*****************************************************************************
% Calculate matrix of actual line admittances (Y and B w/o T and PHI)

  Imp = (o.line.R) + j*(o.line.X);
  [f,t,Imp] = find(Imp);
  x.m.Y = sparse(f,t,1./Imp,o.bus.NBus,o.bus.NBus);
  x.m.YB = x.m.Y + j*(o.line.Bhalf);

%*****************************************************************************
% Indexing

  x.ind.Slackbus = o.bus.Busnum(find(o.bus.Bustype==3));
  x.ind.PQbus    = o.bus.Busnum(find(o.bus.Bustype==0));
  x.ind.PVbus    = o.bus.Busnum(find((o.bus.Bustype==2)|(o.bus.Bustype==3)));
  x.ind.Intr     = find(o.load.Intr);        % where o.load.Intr = 1.
  x.ind.UnIntr   = find(~o.load.Intr);       % where o.load.Intr = 0.
  x.ind.LoadbusI = o.load.Busnum(x.ind.Intr);
  x.ind.LoadbusU = o.load.Busnum(x.ind.UnIntr);
  x.ind.Loadbus = o.load.Busnum;

  x.ind.PFree_1 = find(o.gen.PFree);         % where o.gen.PFree = 1.
  x.ind.QFree_1 = find(o.gen.QFree);         % where o.gen.QFree = 1.
  x.ind.PFree_0 = find(~o.gen.PFree);        % where o.gen.PFree = 0.
  x.ind.QFree_0 = find(~o.gen.QFree);        % where o.gen.QFree = 0.
  x.ind.PFree  =  o.gen.Busnum(x.ind.PFree_1);
  x.ind.QFree  =  o.gen.Busnum(x.ind.QFree_1);
  x.ind.PFixed = [o.gen.Busnum(x.ind.PFree_0)
                  x.ind.PQbus               ];
  x.ind.PFixed = sort(x.ind.PFixed);
  x.ind.QFixed = [o.gen.Busnum(x.ind.QFree_0)
                  x.ind.PQbus               ];
  x.ind.QFixed = sort(x.ind.QFixed);

%*****************************************************************************
% Indexing
```

```
  [x.ind.f, x.ind.t, Y_v] = find(x.m.Y);
  [x.ind.fT, x.ind.tT, x.m.T_v] = find(o.line.T);
  [x.ind.fTfix, x.ind.tTfix, x.m.Tfix_v] = find(o.line.Tfix);
  [x.ind.fTr, x.ind.tTr, x.var.Tr] = find(o.line.Tvar);
  [x.ind.fPHI, x.ind.tPHI, x.m.PHI_v] = find(o.line.PHI);
  [x.ind.fPHIfix, x.ind.tPHIfix, x.m.PHIfix_v] = find(o.line.PHIfix);
  [x.ind.fPhi, x.ind.tPhi, x.var.Phi] = find(o.line.PHIvar);

%************************************************************************
% Initial values for per unit real&reactive power.
% Pg & Qg are what we read initially from CDF file (with the size of NBus),
% they stay unchanged for buses that have fixed generators.
% For the buses with free generator, they will be replaced with the updated
% PG & QG after each iteration.

  x.var.Pg = o.bus.GenMW/x.m.SBase;
  x.var.Qg = o.bus.GenMVAR/x.m.SBase;

% Pl & Ql are what we read initially from CDF file (with the size of NBus),
% they are zeors for buses that don't have load.
% For the buses that have load, they will be replaced with the updated
% PL & QL after each iteration.

  x.var.Pl = o.bus.LoadMW/x.m.SBase;
  x.var.Ql = o.bus.LoadMVAR/x.m.SBase;

  x.var.PL = x.var.Pl(x.ind.Loadbus);

% Note: PG, QG, PL are variables!!!

% QL is a linear function of PL, QL = c * PL, where c = sqrt(1-PF^2)/PF.
% The coefficient c is computed here for all loadbuses.

  x.m.LCoeff = sparse(x.n.NBus,1);
  x.m.LCoeff(x.ind.LoadbusI) = sqrt(1-o.load.PF(x.ind.Intr).^2)./ ...
                                    o.load.PF(x.ind.Intr);
  x.m.LCoeff(x.ind.LoadbusU) = sqrt(1-o.load.PF(x.ind.UnIntr).^2)./ ...
                                    o.load.PF(x.ind.UnIntr);

%************************************************************************
% Calculate some useful numbers that will be refered to later
% i.e. # of variables, # of constraint functions, ...

  x.n.NPFixed = length(x.ind.PFixed);   % number of buses of P fixed
  x.n.NQFixed = length(x.ind.QFixed);   % number of buses of Q fixed
  x.n.NPFree  = length(x.ind.PFree);    % number of buses of P free
  x.n.NQFree  = length(x.ind.QFree);    % number of buses of Q free

  x.n.NTr   = length(x.var.Tr);         % number of variable T
  x.n.NPhi = length(x.var.Phi);         % number of variable PHI
  x.n.NI    = 2*(x.n.NLine);            % number of line currents

  x.n.NX = x.n.NBus + (x.n.NBus-1) + x.n.NTr + x.n.NPhi;
  x.n.NVar = x.n.NX + x.n.NPFree + x.n.NQFree + x.n.NLoad + x.n.NI;
                                        % number of state variables w/o
                                        % angle of Slack bus
  x.n.NZO = x.n.NVar + x.n.NPFixed + x.n.NPFree + x.n.NQFixed + x.n.NQFree + ...
            x.n.NI;                     % number of vector Z w/o Mu's

%************************************************************************
% Forming matrices for indexing.

  x.m.ft2i    = sparse(x.ind.f,x.ind.t,[1:x.n.NI],x.n.NBus,x.n.NBus);
  x.m.ft2iTr  = sparse(x.ind.fTr,x.ind.tTr,[1:x.n.NTr],x.n.NBus,x.n.NBus);
  x.m.ft2iPhi = sparse(x.ind.fPhi,x.ind.tPhi,[1:x.n.NPhi],x.n.NBus,x.n.NBus);

%************************************************************************
% Set Up Indicies for Vectors.
```

```
% Indexing vector Z

  x.ind.Slack   = x.n.NBus + x.ind.Slackbus;    % index of angle of slack bus
                                                % in vector Z.
  x.ind.V       = [1:x.n.NBus]';                next = x.n.NBus;
  x.ind.Angl    = [next+1:next+(x.n.NBus-1)]';  next = next +(x.n.NBus-1);
  x.ind.Tr      = [next+1:next+x.n.NTr]';       next = next + x.n.NTr;
  x.ind.Phi     = [next+1:next+x.n.NPhi]';      next = next + x.n.NPhi;
  x.ind.PG      = [next+1:next+x.n.NPFree]';    next = next + x.n.NPFree;
  x.ind.QG      = [next+1:next+x.n.NQFree]';    next = next + x.n.NQFree;
  x.ind.PL      = [next+1:next+x.n.NLoad]';     next = next + x.n.NLoad;
  x.ind.Is      = [next+1:next+x.n.NI]';        next = next + x.n.NI;
  x.ind.LambdaP = [next+1:next+x.n.NBus]';      next = next + x.n.NBus;
  x.ind.LambdaQ = [next+1:next+x.n.NBus]';      next = next + x.n.NBus;
  x.ind.LambdaI = [next+1:next+x.n.NI]';
  x.ind.Z0 = [x.ind.V; x.ind.Angl; x.ind.Tr; x.ind.Phi; x.ind.PG; x.ind.QG;...
              x.ind.PL; x.ind.Is; x.ind.LambdaP; x.ind.LambdaQ; x.ind.LambdaI];

%*************************************************************************
% Initial values of Multipliers (LambdaP, LambdaQ).
% LambdaQ should initially be set close to zero,
% but LambdaP should be close to (average) Marginal Cost of the system.
% Thus, LambdaP will be initiallized when MC is computed (in PowerFlow.m).

  x.var.LambdaP = ones(x.n.NBus,1);     % will be changed to MC later.
  x.var.LambdaQ = -0.1*ones(x.n.NBus,1);
  x.var.LambdaI = -0.1*ones(x.n.NI,1);

%*************************************************************************
% Vector of upper and lower limits (for MAIN problem).

  x.lim.Vmax(o.gen.Busnum,1) = o.gen.VGmax;
  x.lim.Vmax(o.bus.Busnum(x.ind.PQbus)) = o.bus.VQmax(x.ind.PQbus);
  x.lim.Anglmax = inf*ones(x.n.NBus-1,1);
  x.lim.Trmax = nonzeros(o.line.Tmax);
  x.lim.Phimax = nonzeros(o.line.PHImax);
  x.lim.PGmax = o.gen.MWmax(x.ind.PFree_1)/x.m.SBase;
  x.lim.QGmax = o.gen.MVARmax(x.ind.QFree_1)/x.m.SBase;
  x.lim.PLmax = o.load.MWmax/x.m.SBase;
  x.lim.QLmax = o.load.MVARmax/x.m.SBase;    % Should we have this????
  [junk, junk, x.lim.Imax]  = find(o.line.Linemax./x.m.SBase);

  x.lim.Vmin(o.gen.Busnum,1) = o.gen.VGmin;
  x.lim.Vmin(o.bus.Busnum(x.ind.PQbus)) = o.bus.VQmin(x.ind.PQbus);
  x.lim.Anglmin = -x.lim.Anglmax;
  x.lim.Trmin = nonzeros(o.line.Tmin);
  x.lim.Phimin = nonzeros(o.line.PHImin);
  x.lim.PGmin = o.gen.MWmin(x.ind.PFree_1)/x.m.SBase;
  x.lim.QGmin = o.gen.MVARmin(x.ind.QFree_1)/x.m.SBase;
  x.lim.PLmin = o.load.MWmin/x.m.SBase;
  x.lim.QLmin = o.load.MVARmin/x.m.SBase;
  x.lim.Imin = -inf*ones(size(x.lim.Imax));

  x.lim.Max = [x.lim.Vmax; x.lim.Anglmax; x.lim.Trmax; x.lim.Phimax; ...
               x.lim.PGmax; x.lim.QGmax; x.lim.PLmax; x.lim.Imax];
  x.lim.Min = [x.lim.Vmin; x.lim.Anglmin; x.lim.Trmin; x.lim.Phimin; ...
               x.lim.PGmin; x.lim.QGmin; x.lim.PLmin; x.lim.Imin];

%*************************************************************************
% Prepare data for cost function.

  x.cost.Alpha  = o.gen.Alpha(x.ind.PFree_1);
  x.cost.Beta   = o.gen.Beta(x.ind.PFree_1);
  x.cost.Gamma  = o.gen.Gamma(x.ind.PFree_1);
  x.cost.BetaR  = o.gen.BetaR(x.ind.PFree_1);
  x.cost.GammaR = o.gen.GammaR(x.ind.PFree_1);
  x.cost.AlphaL = o.load.Alpha;
  x.cost.BetaL  = o.load.Beta;
```

```
  x.cost.GammaL = o.load.Gamma;
  x.cost.AlphaI = o.load.AlphaI;
  x.cost.BetaI  = o.load.BetaI;
  x.cost.GammaI = o.load.GammaI;

  x.cost.BetaPu  = x.cost.Beta*(x.m.SBase/x.m.CBase);
  x.cost.GammaPu = x.cost.Gamma*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaLPu  = x.cost.BetaL*(x.m.SBase/x.m.CBase);
  x.cost.GammaLPu = x.cost.GammaL*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaRPu  = x.cost.BetaR*(x.m.SBase/x.m.CBase);
  x.cost.GammaRPu = x.cost.GammaR*(x.m.SBase^2/x.m.CBase);

  x.cost.BetaIPu  = x.cost.BetaI*(x.m.SBase/x.m.CBase);
  x.cost.GammaIPu = x.cost.GammaI*(x.m.SBase^2/x.m.CBase);

  x.cost.d2  = 2*(x.cost.GammaPu);
  x.cost.d2L = 2*(x.cost.GammaLPu);
  x.cost.d2R = 2*(x.cost.GammaRPu);
  x.cost.d2I = 2*(x.cost.GammaIPu);

%*************************************************************************
% Information on contingencies (line outages).

  [x.ind.fLOut,x.ind.tLOut,x.m.Pi] = find(o.line.Pi);
  x.n.K = length(x.m.Pi);       % Number of contingencies, k = 1...K.
  x.m.Pi0 = 1-sum(x.m.Pi);     % Probability of no contingencies.

  if x.n.K==0   % no contingency.
     x.m.Pi = sparse(x.n.NLine,1);
  end %if

%*************************************************************************
% Define constraint type for printing the results.

  x.type.V    = 1; x.type.type1 = [ 'V   ' ];
  x.type.Angl = 2; x.type.type2 = [ 'Angl' ];
  x.type.Tr   = 3; x.type.type3 = [ 'Tr  ' ];
  x.type.Phi  = 4; x.type.type4 = [ 'Phi ' ];
  x.type.PG   = 5; x.type.type5 = [ 'PG  ' ];
  x.type.QG   = 6; x.type.type6 = [ 'QG  ' ];
  x.type.PL   = 7; x.type.type7 = [ 'PL  ' ];
  x.type.Is   = 8; x.type.type8 = [ 'Is  ' ];

  x.print.Type = [ x.type.V(ones(x.n.NBus,1))
                   x.type.Angl(ones(x.n.NBus-1,1))
                   x.type.Tr(ones(x.n.NTr,1))
                   x.type.Phi(ones(x.n.NPhi,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.QG(ones(x.n.NQFree,1))
                   x.type.PL(ones(x.n.NLoad,1))
                   x.type.Is(ones(x.n.NI,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))    ];

  x.print.Name = [ x.type.type1(ones(1,x.n.NBus),:)
                   x.type.type2(ones(1,x.n.NBus-1),:)
                   x.type.type3(ones(1,x.n.NTr),:)
                   x.type.type4(ones(1,x.n.NPhi),:)
                   x.type.type5(ones(1,x.n.NPFree),:)
                   x.type.type6(ones(1,x.n.NQFree),:)
                   x.type.type7(ones(1,x.n.NLoad),:)
                   x.type.type8(ones(1,x.n.NI),:)
                   x.type.type5(ones(1,x.n.NPFree),:)
                   x.type.type7(ones(1,x.n.NLoad),:)  ];

  x.print.Number = [ (1:x.n.NBus)'
```

```
                    (1:x.n.NBus)'
                    (1:x.n.NTr)'
                    (1:x.n.NPhi)'
                    (x.ind.PFree)
                    (x.ind.QFree)
                    (x.ind.Loadbus)
                    (1:x.n.NI)'
                    (x.ind.PFree)
                    (x.ind.Loadbus) ];
  x.print.Number(x.ind.Slack) =[];        % Take off Angle of Slack bus.

%*************************************************************************
%*************************************************************************
%*************************************************************************
%*************************************************************************
% Pick the method.  Once you pick it here, it can't be changed throughout
% the whole program.

  %----------------------------------
  clear x.flag.ip;
  flagIP = 2;        % default value.
  x.flag.ip = input(sprintf ...
  (' FP press 0; IP press 1; IPwRho press 2 (default= %2i): ', flagIP));
     if isempty(x.flag.ip)
        x.flag.ip = flagIP;
     end %if
  %----------------------------------
% Set up indicies of vector of upper and lower limits.
% Since constraint function consists of all state variables, indexing of
% A matrix (and limits) is the same as above (w/o LambdaP, LambdaQ, LambdaI).
% But there are coupling constraints which will be taken care of in Modified.

  if x.flag.ip
     x.ind.aH = [x.ind.V; x.ind.PG; x.ind.QG; x.ind.Is];
     x.ind.aL = [x.ind.V; x.ind.PG; x.ind.QG; x.ind.PL];

     x.n.NaH = length(x.ind.aH);           % Number of upper limits.
     x.n.NaL = length(x.ind.aL);           % Number of lower limits.

     x.var.sH  = 1*ones(x.n.NaH,1);
     x.var.sL  = 1*ones(x.n.NaL,1);
     x.var.MuH = 0.01*ones(x.n.NaH,1);
     x.var.MuL = 0.01*ones(x.n.NaL,1);
     x.var.rhoH = 0.01*ones(x.n.NaH,1);
     x.var.rhoL = 0.01*ones(x.n.NaL,1);
     x.var.MurhoH = 1000*ones(x.n.NaH,1);
     x.var.MurhoL = 1000*ones(x.n.NaL,1);

     %----------------------------------
     clear x.printIP;
     printIP = 1e-6;  % default value.
     x.printIP = input(sprintf ...
  (' Start showing print statement when Nu is this small (default=%6.4g): ',...
     printIP));
     if isempty(x.printIP)
        x.printIP = printIP;
     end %if
     %----------------------------------
     clear x.var.NuFac
     NuFac = 0.1;    % default value.
     x.var.NuFac = input(sprintf ...
                  (' Enter factor of Nu (default=%6.4g): ', NuFac));
     if isempty(x.var.NuFac)
        x.var.NuFac = NuFac;
     end %if
     %----------------------------------
     clear x.var.NuLast
     NuLast = 1e-15;  % default value.
```

```
    x.var.NuLast = input(sprintf ...
              (' Enter the last value of Nu (default=%6.4g): ', NuLast));
    if isempty(x.var.NuLast)
       x.var.NuLast = NuLast;
    end %if
    %-----------------------------------
    clear x.m.sig
    sig = 1;
    x.m.sig = input(sprintf ...
                 (' Enter sigma (default=%4i): ', sig));
    if isempty(x.m.sig)
       x.m.sig = sig;
    end %if
    %-----------------------------------
    if x.flag.ip==2
       clear x.cost.rho;
       cost_rho = 1000;   % default value (per additional MW or MVar).
       x.cost.rho =  input(sprintf ...
         (' Enter penalty cost for exceeding limits (default=%6i): ', ...
            cost_rho));
       if isempty(x.cost.rho)
          x.cost.rho = cost_rho;
       end %if
       %-----------------------------------
       x.var.Nu = 1/(2*(x.n.NaH+x.n.NaL))* ...
                  sum([x.var.MuH;x.var.MuL;x.var.MurhoH;x.var.MurhoL].*...
                      [x.var.sH;x.var.sL;x.var.rhoH;x.var.rhoL]);
       end %if x.flag.ip==2
       %-----------------------------------
  else % for FP
     x.var.MuH  = [];                  % Mu for X binding at upper limit.
     x.var.MuL  = [];                  % Mu for X binding at lower limit.
     x.ind.aH   = [];                  % index of upper active set
     x.ind.aL   = [];                  % index of lower active set
     x.ind.aMuH = [];                  % index of MuH change sign
     x.ind.aMuL = [];                  % index of MuL change sign

  end %if x.flag.ip

%******************************************************************************
% Set up maximum number of iterations for OPF.

  x.n.NIter = 100;           % Set up default value for number of iterations.
  NIter_new = input(sprintf(...
      ' Enter MAX number of iterations for OPF (default=%4i): ', x.n.NIter));

  if ~isempty(NIter_new),
     x.n.NIter = NIter_new;
  end

%******************************************************************************
% Iteration number of the whole problem

  x.Iter = 1;

%******************************************************************************
% Save initial variables.

  x.init = x;

%******************************************************************************
```

```
  function[x] = Modified(o,x,k)

% function[x] = Modified(o,x,k)
%
% This function modifies the structure of the system for each subproblem.
% and set initial values.
% So it does the same thing as Initial, but for subproblems.
%
% For each subproblem, input variables and indicies are from x_main.

%=========================================================================

  x.var = x.init.var;
  x.ind = x.init.ind;

%*************************************************************************
% Modify the system by taking out one line for each subproblem.

  x.n.NLine = o.line.NLine-1;
  x.n.NLoad = nnz(o.load.Intr);
  x.ind.Loadbus = x.ind.LoadbusI;
  x.var.Pl(x.ind.LoadbusU) = x.var.PL(x.ind.UnIntr);
  x.var.PL = x.var.PL(x.ind.Intr);  % Only interruptible PL are variables.

  fk = [x.ind.fLOut(k); x.ind.tLOut(k)];
  tk = [x.ind.tLOut(k); x.ind.fLOut(k)];

  o.line.Linetype(fk,tk) = 0;
  o.line.R(fk,tk) = 0;
  o.line.X(fk,tk) = 0;
  o.line.Bhalf(fk,tk) = 0;
% o.line.Linemax(fk,tk) = 0;        % don't need this, use Linemax2.
  o.line.Linemax2(fk,tk) = 0;
  o.line.Tfix(fk,tk) = 0;
  o.line.Tvar(fk,tk) = 0;
  o.line.T(fk,tk) = 0;
  o.line.PHIfix(fk,tk) = 0;
  o.line.PHIvar(fk,tk) = 0;
  o.line.PHI(fk,tk) = 0;
  o.line.Tmax(fk,tk) = 0;
  o.line.Tmin(fk,tk) = 0;
  o.line.PHImax(fk,tk) = 0;
  o.line.PHImin(fk,tk) = 0;

%*************************************************************************
% Calculate some useful numbers after one line is taken out.
% i.e. # of variables, # of constraint functions, ...

  x.n.NTr   = length(x.var.Tr);          % number of variable T          %!!!!
  x.n.NPhi = length(x.var.Phi);          % number of variable PHI        %!!!!
  x.n.NI    = 2*(x.n.NLine);             % number of line currents       %!!!!

  x.n.NX = x.n.NBus + (x.n.NBus-1) + x.n.NTr + x.n.NPhi;                 %!!!!
  x.n.NVar = x.n.NX + x.n.NPFree + x.n.NQFree + x.n.NLoad + x.n.NI;      %!!!!
                                         % number of state variables w/o
                                         % angle of Slack bus
  x.n.NZO = x.n.NVar + x.n.NPFixed + x.n.NPFree + x.n.NQFixed + x.n.NQFree + ...
            x.n.NI;                      % number of vector Z w/o Mu's   %!!!!

%*************************************************************************
% Special limits that don't appear in MAIN problem.

  [junk,junk,Imax] = find(o.line.Linemax2);
  x.lim.Imax = Imax./x.m.SBase;
  x.lim.Imin  = -inf*ones(x.n.NI,1);
  x.lim.MWup = o.gen.MWup(x.ind.PFree_1)/x.m.SBase;
  x.lim.MWdn = o.gen.MWdn(x.ind.PFree_1)/x.m.SBase;
  x.lim.MWmaxInt = o.load.MWmaxInt(x.ind.Intr)/x.m.SBase;
```

```
%*************************************************************************
% Calculate matrix of actual line admittances (Y and B w/o T and PHI)

  Imp = (o.line.R) + j*(o.line.X);
  [f,t,Imp] = find(Imp);
  x.m.Y = sparse(f,t,1./Imp,o.bus.NBus,o.bus.NBus);
  x.m.YB = x.m.Y + j*(o.line.Bhalf);

%*************************************************************************
% Indexing

  [x.ind.f, x.ind.t, Y_v] = find(x.m.Y);
  [x.ind.fT, x.ind.tT, x.m.T_v] = find(o.line.T);
  [x.ind.fTfix, x.ind.tTfix, x.m.Tfix_v] = find(o.line.Tfix);
  [x.ind.fTr, x.ind.tTr, x.var.Tr] = find(o.line.Tvar);
  [x.ind.fPHI, x.ind.tPHI, x.m.PHI_v] = find(o.line.PHI);
  [x.ind.fPHIfix, x.ind.tPHIfix, x.m.PHIfix_v] = find(o.line.PHIfix);
  [x.ind.fPhi, x.ind.tPhi, x.var.Phi] = find(o.line.PHIvar);

%*************************************************************************
% Forming matrices for indexing.

  x.m.ft2i    = sparse(x.ind.f,x.ind.t,[1:x.n.NI],x.n.NBus,x.n.NBus);   %!!!!
  x.m.ft2iTr  = sparse(x.ind.fTr,x.ind.tTr,[1:x.n.NTr],x.n.NBus,x.n.NBus);%!!!!
  x.m.ft2iPhi = sparse(x.ind.fPhi,x.ind.tPhi,[1:x.n.NPhi],x.n.NBus,x.n.NBus);%!!

%*************************************************************************
% Set Up Indicies for Vectors.
% Indexing vector Z

  x.ind.Slack   = x.n.NBus + x.ind.Slackbus;    % index of angle of slack bus
                                                % in vector Z.
  x.ind.V       = [1:x.n.NBus]';                next = x.n.NBus;
  x.ind.Angl    = [next+1:next+(x.n.NBus-1)]';  next = next +(x.n.NBus-1);
  x.ind.Tr      = [next+1:next+x.n.NTr]';       next = next + x.n.NTr;  %!!!!
  x.ind.Phi     = [next+1:next+x.n.NPhi]';      next = next + x.n.NPhi; %!!!!
  x.ind.PG      = [next+1:next+x.n.NPFree]';    next = next + x.n.NPFree;%!!!!
  x.ind.QG      = [next+1:next+x.n.NQFree]';    next = next + x.n.NQFree;%!!!!
  x.ind.PL      = [next+1:next+x.n.NLoad]';     next = next + x.n.NLoad; %!!!!
  x.ind.Is      = [next+1:next+x.n.NI]';        next = next + x.n.NI;   %!!!!
  x.ind.LambdaP = [next+1:next+x.n.NBus]';      next = next + x.n.NBus; %!!!!
  x.ind.LambdaQ = [next+1:next+x.n.NBus]';      next = next + x.n.NBus; %!!!!
  x.ind.LambdaI = [next+1:next+x.n.NI]';                                %!!!!
  x.ind.Z0 = [x.ind.V; x.ind.Angl; x.ind.Tr; x.ind.Phi; x.ind.PG; x.ind.QG;...
              x.ind.PL; x.ind.Is; x.ind.LambdaP; x.ind.LambdaQ; x.ind.LambdaI];
                                                                        %!!!!
% Since constraint function consists of all state variables, indexing of
% A matrix (and limits) is the same as above (w/o LambdaP, LambdaQ, LambdaI).

%*************************************************************************
% Initial values of Multipliers (LambdaIs, Mu, ...), and indicies for
% subproblems.

  x.var.LambdaI = -1*ones(2*x.n.NLine,1);

% Indicies of constraints in subproblems (including coupling constraints).

                                                next = x.n.NVar;
  x.ind.PGC = [next+1:next+x.n.NPFree]';        next = next + x.n.NPFree;
  x.ind.PLC = [next+1:next+x.n.NLoad]';

  x.ind.aH_k  = [x.ind.V; x.ind.PG; x.ind.QG; x.ind.Is];
  x.ind.aH_k0 = [x.ind.PGC; x.ind.PLC];
  x.ind.aH    = [x.ind.aH_k; x.ind.aH_k0];
  x.ind.aL_k  = [x.ind.V; x.ind.PG; x.ind.QG; x.ind.PL];
  x.ind.aL_k0 = [x.ind.PGC; x.ind.PLC];
  x.ind.aL    = [x.ind.aL_k; x.ind.aL_k0];
```

```
  x.n.NaH_k  = length(x.ind.aH_k);
  x.n.NaH_k0 = length(x.ind.aH_k0);
  x.n.NaH    = length(x.ind.aH);
  x.n.NaL_k  = length(x.ind.aL_k);
  x.n.NaL_k0 = length(x.ind.aL_k0);
  x.n.NaL    = length(x.ind.aL);

  x.var.sH  = 1*ones(x.n.NaH,1);
  x.var.sL  = 1*ones(x.n.NaL,1);
  x.var.MuH = 0.01*ones(x.n.NaH,1);
  x.var.MuL = 0.01*ones(x.n.NaL,1);
  x.var.rhoH = 0.01*ones(x.n.NaH,1);
  x.var.rhoL = 0.01*ones(x.n.NaL,1);
  x.var.MurhoH = 1000*ones(x.n.NaH,1);
  x.var.MurhoL = 1000*ones(x.n.NaL,1);

%*************************************************************************
% Define constraint type for printing the results.

  x.print.Type = [ x.type.V(ones(x.n.NBus,1))
                   x.type.Angl(ones(x.n.NBus-1,1))
                   x.type.Tr(ones(x.n.NTr,1))
                   x.type.Phi(ones(x.n.NPhi,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.QG(ones(x.n.NQFree,1))
                   x.type.PL(ones(x.n.NLoad,1))
                   x.type.Is(ones(x.n.NI,1))
                   x.type.PG(ones(x.n.NPFree,1))
                   x.type.PL(ones(x.n.NLoad,1))    ];

  x.print.Name = [ x.type.type1(ones(1,x.n.NBus),:)
                   x.type.type2(ones(1,x.n.NBus-1),:)
                   x.type.type3(ones(1,x.n.NTr),:)
                   x.type.type4(ones(1,x.n.NPhi),:)
                   x.type.type5(ones(1,x.n.NPFree),:)
                   x.type.type6(ones(1,x.n.NQFree),:)
                   x.type.type7(ones(1,x.n.NLoad),:)
                   x.type.type8(ones(1,x.n.NI),:)
                   x.type.type5(ones(1,x.n.NPFree),:)
                   x.type.type7(ones(1,x.n.NLoad),:)   ];

% Which line is gone???

  Line_num_new = (1:x.init.n.NI)';
  no_line  = x.init.m.ft2i(x.ind.fLOut(k),x.ind.tLOut(k));
  no_line2 = x.init.m.ft2i(x.ind.tLOut(k),x.ind.fLOut(k));
  [noline,junk,junk]  = find(Line_num_new==no_line);
  [noline2,junk,junk] = find(Line_num_new==no_line2);
  Line_num_new(no_line)  = [];
  Line_num_new(no_line2) = [];

  x.print.Number = [ (1:x.n.NBus)'
                     (1:x.n.NBus)'
                     (1:x.n.NTr)'
                     (1:x.n.NPhi)'
                     (x.ind.PFree)
                     (x.ind.QFree)
                     (x.ind.Loadbus)
                     (Line_num_new)
                     (x.ind.PFree)
                     (x.ind.Loadbus) ];

  x.print.Number(x.ind.Slack) =[];        % Take off Angle of Slack bus.

%*************************************************************************
```

```
  function[cost] = CostFunc(SBase,cost,PG0,PG,PL0,PL,k,Intr,Pi0,Pi)

% function[x.cost] = CostFunc(x.m.SBase, x.cost, x.main.var.PG, ...
%                     x.var.PG, x.main.var.PL, x.var.PL, x.k, x.ind.Intr, ...
%                     x.m.Pi0, x.m.Pi)
%
% This function calculates cost of generation.
% The inputs of this function are coefficients of cost curves, which is the
% structure "x.cost", and the real power generated, which is "x.var.PG".
% The output of this function will be put in the structure "x.cost".

%=============================================================================
% Cost of generation and its derivatives.
  cost.d2 = 2*(cost.GammaPu);
  cost.MC = (cost.BetaPu) + 2*(cost.GammaPu).*(PG);

% For benefit curves of both types of load buses (interruptible or not).
  if k==0
     cost.d2L = 2*(cost.GammaLPu);
     cost.MB  = cost.BetaLPu + 2*(cost.GammaLPu.* PL);
  else
     cost.d2L = 2*(cost.GammaLPu(Intr));
     cost.MB  = cost.BetaLPu(Intr) + 2*(cost.GammaLPu(Intr).* PL);
  end %if

% Cost of generator ramping up/down, and its derivatives.
  cost.d2R = 2*(cost.GammaRPu);
  cost.MCR = (cost.BetaRPu) + 2*(cost.GammaRPu).*(PG-PG0);

% Cost of interruption and its derivatives.
  cost.d2I = 2*(cost.GammaIPu(Intr));
  cost.MCI = (cost.BetaIPu(Intr))+2*(cost.GammaIPu(Intr)).*(PL-PL0(Intr));

%-----------------------------------------------------------------------------
% Compute total cost.

  if k==0
     CostPG  = (cost.Alpha) + (cost.Beta).*(SBase*PG) + ...
                              (cost.Gamma).*(SBase*PG).^2;
     Benefit = (cost.AlphaL) + (cost.BetaL).*(SBase*PL) + ...
                               (cost.GammaL).*(SBase*PL).^2;
     cost.Total = Pi0*( sum(CostPG) - sum(Benefit) );
  else
     CostPG  = (cost.Alpha) + (cost.Beta).*(SBase*PG) + ...
                              (cost.Gamma).*(SBase*PG).^2  + ...
                              (cost.BetaR).*(SBase*(abs(PG-PG0))) + ...
                              (cost.GammaR).*(SBase*(abs(PG-PG0))).^2;
     CostIntr = (cost.AlphaI(Intr)) + ...
                        (cost.BetaI(Intr)).*(SBase*(PL0(Intr)-PL)) + ...
                        (cost.GammaI(Intr)).*(SBase*(PL0(Intr)-PL)).^2;
     Benefit  = (cost.AlphaL(Intr)) + ...
                        (cost.BetaL(Intr)).*(SBase*PL) + ...
                        (cost.GammaL(Intr)).*(SBase*PL).^2;
     cost.Total  = sum(CostPG) + sum(CostIntr) - sum(Benefit);

  end %if

%=============================================================================
```

```
  function[x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub);

% function[x_all,x_sub,W_all,g_all,norm_g_sub] = All(o,x_main,x_sub)
%
% 1. Form W matrix and vector g for the whole problem (with main and all
%    subproblems).
% 2. Prepare variables and their indicies for deltaZ_all.

%=========================================================================
% Before going inside FOR loop for computing subproblems, prepare variables
% and indicies from main problem.

  x_all.ind.V = x_main.ind.V;
  x_all.ind.Angl = x_main.ind.Angl;
  x_all.ind.Tr = x_main.ind.Tr;
  x_all.ind.Phi = x_main.ind.Phi;
  x_all.ind.PG = x_main.ind.PG;
  x_all.ind.QG = x_main.ind.QG;
  x_all.ind.PL = x_main.ind.PL;
  x_all.ind.Is = x_main.ind.Is;
  x_all.ind.Z0 = x_main.ind.Z0;
  x_all.ind.LambdaP = x_main.ind.LambdaP;
  x_all.ind.LambdaQ = x_main.ind.LambdaQ;
  x_all.ind.LambdaI = x_main.ind.LambdaI;
  x_all.ind.Slack = x_main.ind.Slack;
  x_all.ind.Slackbus = x_main.ind.Slackbus;

  x_all.ind.MuH = (x_main.n.NZ0+1 : x_main.n.NZ0+x_main.n.NaH)';
  x_all.ind.MuL = (x_main.n.NZ0+x_main.n.NaH+1 :  ...
                   x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL)';
  x_all.ind.MurhoH = (x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL+1 : ...
                      x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL)';
  x_all.ind.MurhoL = (x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL+1 : ...
                      x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL))';
  x_all.ind.sH = (x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL)+1 : ...
                  x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL)';
  x_all.ind.sL = (x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL+1 : ...
                  x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL))';
  x_all.ind.rhoH = (x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL)+1 : ...
                    x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL)';
  x_all.ind.rhoL = (x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL+1 : ...
                    x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL))';

  x_all.var.Z = x_main.var.Z;
  x_all.n.NaH = x_main.n.NaH;
  x_all.n.NaL = x_main.n.NaL;

  next = length(x_main.var.Z);
%*************************************************************************
  for k = 1:x_main.n.K,

      x_sub = Limits(x_main,x_sub,k);
      [x] = x_sub(k);       % Need this because x_sub = PowerFlow(x_sub(1),1)
                            % will lose x_sub(2), x_sub(3), ....
      [x,norm_g_sub(k)] = PowerFlow(x,k);
      x_sub(k) = x;

% Compute coupling matrix.

      Coupl = [sparse(x_main.n.NZ0,x_sub(k).n.NZ0), ...
               sparse(x_main.n.NZ0,x_sub(k).n.NaH_k), ...
               spdiags(-ones(x_sub(k).n.NPFree,1), -x_main.n.NX, ...
                       x_main.n.NZ0, x_sub(k).n.NPFree), ...
               spdiags(-ones(x_sub(k).n.NLoad,1), ...
                       -(x_main.n.NX+x_sub(k).n.NPFree+x_sub(k).n.NQFree), ...
                       x_main.n.NZ0, x_sub(k).n.NLoad), ...
               sparse(x_main.n.NZ0,x_sub(k).n.NaL_k), ...
               spdiags(ones(x_sub(k).n.NPFree,1), -x_main.n.NX, ...
```

```
                x_main.n.NZO, x_sub(k).n.NPFree), ...
            spdiags(ones(x_sub(k).n.NLoad,1), ...
                -(x_main.n.NX+x_sub(k).n.NPFree+x_sub(k).n.NQFree), ...
                x_main.n.NZO, x_sub(k).n.NLoad), ...
            sparse(x_main.n.NZO,3*(x_sub(k).n.NaH+x_sub(k).n.NaL))
            sparse(4*(x_main.n.NaH+x_main.n.NaL), ...
                x_sub(k).n.NZO+4*(x_sub(k).n.NaH+x_sub(k).n.NaL))];

    CouplT = [sparse(x_sub(k).n.NZO,x_main.n.NZO+ ...
                4*(x_main.n.NaH+x_main.n.NaL))
            sparse(x_sub(k).n.NaH+x_sub(k).n.NaL, ...
                x_main.n.NZO+4*(x_main.n.NaH+x_main.n.NaL))
            sparse(x_sub(k).n.NaH, x_main.n.NX), ...
            spdiags(-ones(x_main.n.NPFree,1), -(x_sub(k).n.NaH_k), ...
                x_sub(k).n.NaH, x_main.n.NPFree), ...
            sparse(x_sub(k).n.NaH, x_main.n.NQFree), ...
            spdiags(-ones(x_main.n.NLoad,1), ...
                -(x_sub(k).n.NaH_k+x_sub(k).n.NPFree), ...
                x_sub(k).n.NaH, x_main.n.NLoad), ...
            sparse(x_sub(k).n.NaH, x_main.n.NI), ...
            sparse(x_sub(k).n.NaH, 2*x_main.n.NBus+x_main.n.NI), ...
            sparse(x_sub(k).n.NaH, 4*(x_main.n.NaH+x_main.n.NaL))
            sparse(x_sub(k).n.NaL, x_main.n.NX), ...
            spdiags(ones(x_main.n.NPFree,1), -(x_sub(k).n.NaL_k), ...
                x_sub(k).n.NaL, x_main.n.NPFree), ...
            sparse(x_sub(k).n.NaL, x_main.n.NQFree), ...
            spdiags(ones(x_main.n.NLoad,1), ...
                -(x_sub(k).n.NaL_k+x_sub(k).n.NPFree), ...
                x_sub(k).n.NaL, x_main.n.NLoad), ...
            sparse(x_sub(k).n.NaL, x_main.n.NI), ...
            sparse(x_sub(k).n.NaL, 2*x_main.n.NBus+x_main.n.NI), ...
            sparse(x_sub(k).n.NaL, 4*(x_main.n.NaH+x_main.n.NaL))
            sparse(2*(x_sub(k).n.NaH+x_sub(k).n.NaL), ...
                x_main.n.NZO+4*(x_main.n.NaH+x_main.n.NaL)) ];

% Form W matrix and vector g of all subproblems.

    if exist('W_sub')
       W_sub = [W_sub  sparse(length(W_sub),length(x_sub(k).m.W))
                sparse(length(x_sub(k).m.W),length(W_sub))  x_sub(k).m.W];
       g_sub = [g_sub
                x_sub(k).m.g ];
       W_coupl = [W_coupl Coupl];
       W_couplT = [W_couplT; CouplT];

       MuPGHC = MuPGHC+x_sub(k).var.MuPGHC; % These are extra terms in dLdPG
       MuPGLC = MuPGLC+x_sub(k).var.MuPGLC; %
       MuPLHC = MuPLHC+x_sub(k).var.MuPLHC; % and dLdPL for k = 0.
       MuPLLC = MuPLLC+x_sub(k).var.MuPLLC;
    else
       W_sub = x_sub(k).m.W;
       g_sub = x_sub(k).m.g;
       W_coupl = Coupl;
       W_couplT = CouplT;

       MuPGHC = x_sub(k).var.MuPGHC;
       MuPGLC = x_sub(k).var.MuPGLC;
       MuPLHC = x_sub(k).var.MuPLHC;
       MuPLLC = x_sub(k).var.MuPLLC;
    end %if exist W_sub

% Form x_all for deltaZ_all.

    x_all.ind.V = [x_all.ind.V; x_sub(k).ind.V+next];
    x_all.ind.Angl = [x_all.ind.Angl; x_sub(k).ind.Angl+next];
    x_all.ind.Tr = [x_all.ind.Tr; x_sub(k).ind.Tr+next];
    x_all.ind.Phi = [x_all.ind.Phi; x_sub(k).ind.Phi+next];
    x_all.ind.PG = [x_all.ind.PG; x_sub(k).ind.PG+next];
```

```
      x_all.ind.QG = [x_all.ind.QG; x_sub(k).ind.QG+next];
      x_all.ind.PL = [x_all.ind.PL; x_sub(k).ind.PL+next];
      x_all.ind.Is = [x_all.ind.Is; x_sub(k).ind.Is+next];
      x_all.ind.Z0 = [x_all.ind.Z0; x_sub(k).ind.Z0+next];
      x_all.ind.LambdaP = [x_all.ind.LambdaP; x_sub(k).ind.LambdaP+next];
      x_all.ind.LambdaQ = [x_all.ind.LambdaQ; x_sub(k).ind.LambdaQ+next];
      x_all.ind.LambdaI = [x_all.ind.LambdaI; x_sub(k).ind.LambdaI+next];
      x_all.ind.Slack = [x_all.ind.Slack; x_sub(k).ind.Slack+next];
      x_all.ind.Slackbus = [x_all.ind.Slackbus; x_sub(k).ind.Slackbus];

      x_sub(k).ind.MuH = (x_sub(k).n.NZ0+1 : x_sub(k).n.NZ0+x_sub(k).n.NaH)';
      x_sub(k).ind.MuL = (x_sub(k).n.NZ0+x_sub(k).n.NaH+1 :  ...
                          x_sub(k).n.NZ0+x_sub(k).n.NaH+x_sub(k).n.NaL)';
      x_sub(k).ind.MurhoH = (x_sub(k).n.NZ0+x_sub(k).n.NaH+x_sub(k).n.NaL+1 :...
                             x_sub(k).n.NZ0+2*x_sub(k).n.NaH+x_sub(k).n.NaL)';
      x_sub(k).ind.MurhoL = (x_sub(k).n.NZ0+2*x_sub(k).n.NaH+ ...
                             x_sub(k).n.NaL+1 : ...
                             x_sub(k).n.NZ0+2*(x_sub(k).n.NaH+x_sub(k).n.NaL))';
      x_sub(k).ind.sH = (x_sub(k).n.NZ0+2*(x_sub(k).n.NaH+ ...
                         x_sub(k).n.NaL)+1 : x_sub(k).n.NZ0+ ...
                         3*x_sub(k).n.NaH+2*x_sub(k).n.NaL)';
      x_sub(k).ind.sL = (x_sub(k).n.NZ0+3*x_sub(k).n.NaH+ ...
                         2*x_sub(k).n.NaL+1 : ...
                         x_sub(k).n.NZ0+3*(x_sub(k).n.NaH+x_sub(k).n.NaL))';
      x_sub(k).ind.rhoH = (x_sub(k).n.NZ0+3*(x_sub(k).n.NaH+ ...
                           x_sub(k).n.NaL)+1 : ...
                           x_sub(k).n.NZ0+4*x_sub(k).n.NaH+3*x_sub(k).n.NaL)';
      x_sub(k).ind.rhoL = (x_sub(k).n.NZ0+4*x_sub(k).n.NaH+ ...
                           3*x_sub(k).n.NaL+1 : ...
                           x_sub(k).n.NZ0+4*(x_sub(k).n.NaH+x_sub(k).n.NaL))';

      x_all.ind.MuH = [x_all.ind.MuH; x_sub(k).ind.MuH+next];
      x_all.ind.MuL = [x_all.ind.MuL; x_sub(k).ind.MuL+next];
      x_all.ind.MurhoH = [x_all.ind.MurhoH; x_sub(k).ind.MurhoH+next];
      x_all.ind.MurhoL = [x_all.ind.MurhoL; x_sub(k).ind.MurhoL+next];
      x_all.ind.sH = [x_all.ind.sH; x_sub(k).ind.sH+next];
      x_all.ind.sL = [x_all.ind.sL; x_sub(k).ind.sL+next];
      x_all.ind.rhoH = [x_all.ind.rhoH; x_sub(k).ind.rhoH+next];
      x_all.ind.rhoL = [x_all.ind.rhoL; x_sub(k).ind.rhoL+next];

      x_all.var.Z = [x_all.var.Z; x_sub(k).var.Z];
      x_all.n.NaH = x_all.n.NaH+x_sub(k).n.NaH;
      x_all.n.NaL = x_all.n.NaL+x_sub(k).n.NaL;

      next = next + length(x_sub(k).var.Z);

  end %for k
%*************************************************************************
% Form huge W matrix and vector g which have all subproblems and main problem.

  if isempty(x_sub)
     W_coupl  = [];
     W_couplT = [];
     W_sub    = [];
     g_sub    = [];
     norm_g_sub = [];

     MuPGHC = sparse(x_main.n.NPFree,1);
     MuPGLC = sparse(x_main.n.NPFree,1);
     MuPLHC = sparse(x_main.n.NLoad,1);
     MuPLLC = sparse(x_main.n.NLoad,1);
  end %if

  W_all = [x_main.m.W     W_coupl
           W_couplT       W_sub];

% This will take care of Mu's of the coupling constraints.
  x_main.m.g(x_main.ind.PG) = x_main.m.g(x_main.ind.PG) - MuPGHC + MuPGLC;
```

```
  x_main.m.g(x_main.ind.PL) = x_main.m.g(x_main.ind.PL) - MuPLHC + MuPLLC;

  g_all = [x_main.m.g
           g_sub];

  x_all.var.MuH = x_all.var.Z(x_all.ind.MuH);
  x_all.var.MuL = x_all.var.Z(x_all.ind.MuL);
  x_all.var.MurhoH = x_all.var.Z(x_all.ind.MurhoH);
  x_all.var.MurhoL = x_all.var.Z(x_all.ind.MurhoL);
  x_all.var.sH = x_all.var.Z(x_all.ind.sH);
  x_all.var.sL = x_all.var.Z(x_all.ind.sL);
  x_all.var.rhoH = x_all.var.Z(x_all.ind.rhoH);
  x_all.var.rhoL = x_all.var.Z(x_all.ind.rhoL);

  x_all.var.Nu = x_main.var.Nu;  % Nu of subproblems are the same as this, too.

%=========================================================================
```

```
  function[x_sub] = Limits(x_main,x_sub,k)

% function[x_sub] = Limits(x_main,x_sub,k)
%
% Vector of upper and lower limits.
% Limits of PG, PL, and Is are different in main and subproblems.
% They have to be reset every iteration.

%=========================================================================
% Upper limits (including coupling constraints).

  x_sub(k).lim.PGmax = x_main.lim.PGmax;
  x_sub(k).lim.QGmax = x_main.lim.QGmax;
  x_sub(k).lim.PLmax = x_main.var.PL;

  x_sub(k).lim.Max = [x_sub(k).lim.Vmax
                      x_sub(k).lim.Anglmax
                      x_sub(k).lim.Trmax
                      x_sub(k).lim.Phimax
                      x_sub(k).lim.PGmax
                      x_sub(k).lim.QGmax
                      x_sub(k).lim.PLmax
                      x_sub(k).lim.Imax        % below are coupling limits.
                      x_main.var.PG+x_sub(k).lim.MWup
                      x_main.var.PL                  ];

%-------------------------------------------------------------------------
% Lower limits (including coupling constraints).

  x_sub(k).lim.PGmin = x_main.lim.PGmin;
  x_sub(k).lim.QGmin = x_main.lim.QGmin;
  x_sub(k).lim.PLmin = zeros(x_sub(k).n.NLoad,1);

  x_sub(k).lim.Min = [x_sub(k).lim.Vmin
                      x_sub(k).lim.Anglmin
                      x_sub(k).lim.Trmin
                      x_sub(k).lim.Phimin
                      x_sub(k).lim.PGmin
                      x_sub(k).lim.QGmin
                      x_sub(k).lim.PLmin
                      x_sub(k).lim.Imin        % below are coupling limits.
                      x_main.var.PG-x_sub(k).lim.MWdn
                      x_main.var.PL-x_sub(k).lim.MWmaxInt ];

%*************************************************************************
```

```
  function[x,norm_g] = PowerFlow(x,k)

% function[x,norm_g] = PowerFlow(x,k)
%
% There are 2 parts in this function:
% Part1: Calculates all power flow functions to be used in OPF program.
% These power flow functions are:
% - Currents
% - Complex power
% - Real & Reactive power
% - Mismatch power flow equations
% - Generation cost
%
% Part2:
% - Calculates first order and second order derivatives.
% - Forms Jacobian, Hessian, and A  matrices.
% - Uses J, H, and A to build W matrix and vector g.

%=============================================================================
% Terms involving Voltages and Angles.

  Vcomp = (x.var.V) .* exp(j*x.var.Angl);
  Vcomp_diag = spdiags(Vcomp,[0],x.n.NBus,x.n.NBus);
  V_diag = spdiags(x.var.V,[0],x.n.NBus,x.n.NBus);
  ejAngl = exp(j*x.var.Angl);
  ejAngl_diag = spdiags((ejAngl),[0],x.n.NBus,x.n.NBus);

%=============================================================================
% Terms involving Y & B & T & PHI

  Tminus1  = sparse([x.ind.fTfix; x.ind.fTr], [x.ind.tTfix; x.ind.tTr],...
                    [x.m.Tfix_v-1; x.var.Tr-1], x.n.NBus, x.n.NBus);
  ejPHIminus1  = sparse([x.ind.fPHIfix; x.ind.fPhi], ...
                        [x.ind.tPHIfix; x.ind.tPhi], ...
                        [(exp(j*x.m.PHI_v)-1); (exp(j*x.var.Phi)-1)], ...
                        x.n.NBus,x.n.NBus);
  TejPHIminus1 = Tminus1.*ejPHIminus1 + Tminus1 + ejPHIminus1;

  YBPHIT   = TejPHIminus1.*x.m.YB + x.m.YB;
  YBPHIT_v = nonzeros(YBPHIT);
  YBPHI    = ejPHIminus1.*x.m.YB + x.m.YB;
  YBPHI_v  = nonzeros(YBPHI);
  YPHIT    = (TejPHIminus1.').*x.m.Y + x.m.Y;
                                    % Transpost because Tji,PHIji, but Yij.
  YPHIT_v  = nonzeros(YPHIT);
  YPHI     = (ejPHIminus1.').*x.m.Y + x.m.Y;
  YPHI_v   = nonzeros(YPHI);

% This special Yp is for calculating complex current (I) only.

  Yp = sparse([1:2*x.n.NLine  1:2*x.n.NLine], [x.ind.f ; x.ind.t], ...
              [nonzeros(YBPHIT) -nonzeros(YPHIT)], 2*x.n.NLine, x.n.NBus);

%=============================================================================
% Current Calculations

  I     = Yp*Vcomp;             % This is a vector of complex currents.
  Ireal = real(I);
  Iimag = imag(I);
  Iconj = conj(I);
  Iabs  = abs(I);
  Iabs2 = Iabs.^2;              % = |I|^2

  Ishunt = (Vcomp.*x.m.Bshunt);
  Imatrix = sparse(x.ind.f, x.ind.t, I, x.n.NBus, x.n.NBus);
  Imatrix_conj = conj(Imatrix);

%=============================================================================
```

```
% Power Flow Calculations

  Smatrix = spdiags(Vcomp,[0],x.n.NBus,x.n.NBus)* ...
            (Tminus1.*ejPHIminus1.*Imatrix_conj + ...
             ejPHIminus1.* Imatrix_conj + Tminus1.* Imatrix_conj + ...
             Imatrix_conj);
                                % This is equal to Vcomp*T*ejPHI*Imatrix_conj
                                % (see exact equation).

  Sshunt  = (Vcomp).* conj(Ishunt);
  Sbus    = sum(Smatrix.').' + Sshunt;
                                % S1 = S11+S12+....+Sshunt1.
  x.var.Pbus  = real(Sbus);     % Vector of calculated real power.
  x.var.Qbus  = imag(Sbus);     % Vector of calculated reactive power.

%=========================================================================
% Setting up Constraint Functions.
% For the first iteration, make sure that the initial values of variables
% PG, QG, PL, and Is are in feasible region. (Recall: QL is not a variable!!!)
%
% Note: PL guessed comes from Initial.
%       QL guessed is supposed to be calculated here from PL.
%
% First, make sure initial variables are feasible for the very first time
% of OPF.

  if x.Iter==1
     x.var.V = min(x.var.V, 0.99*x.lim.Vmax);
     x.var.V = max(x.var.V, 1.01*x.lim.Vmin);

     x.var.PL = min(x.var.PL, 0.95*x.lim.PLmax);
     x.var.PL = max(x.var.PL, 1.05*x.lim.PLmin);

     x.var.PG = x.var.Pbus(x.ind.PFree) + x.var.Pl(x.ind.PFree);
     x.var.PG = min(x.var.PG, 0.95*x.lim.PGmax);
     x.var.PG = max(x.var.PG, 1.05*x.lim.PGmin);

     x.var.QL = (x.var.PL).*x.m.LCoeff(x.ind.Loadbus);

     x.var.QG = x.var.Qbus(x.ind.QFree) + x.var.Ql(x.ind.QFree);
     x.var.QG = min(x.var.QG, 0.95*x.lim.QGmax);
     x.var.QG = max(x.var.QG, 1.05*x.lim.QGmin);

     x.var.Is = min(Iabs, 0.90*x.lim.Imax);
  end%if

% Vector of inequality constraint functions.
  x.var.Fn = [ x.var.V
               x.var.Angl
               x.var.Tr
               x.var.Phi
               x.var.PG
               x.var.QG
               x.var.PL
               x.var.Is
               x.var.PG
               x.var.PL ];

  x.var.Fn(x.ind.Slack,:) = [];      % Take out angle of slack bus.

%=========================================================================
% Generation cost calculation.

  if x.Iter==1                       % The very first iteration, x.main is
     x.init.var.PG = x.var.PG;       % not set yet.
     x.init.var.PL = x.var.PL;
  end %if
```

```
  [x.cost] = CostFunc(x.m.CBase, x.cost, x.init.var.PG, x.var.PG, ...
                      x.init.var.PL, x.var.PL, k, x.ind.Intr, ...
                      x.m.Pi0, x.m.Pi);

% For the 1st iteration, LambdaP from Initial.m should be initialized
% to the average marginal cost.
  if x.Iter==1
    x.var.LambdaP(x.ind.Loadbus) = (x.cost.MB);
    x.var.LambdaP(x.ind.PFree) = (x.cost.MC);
  end %if

% If using IP with rho, add additional charge for exceeding the limits.

  if x.flag.ip==2
     x.cost.Total = x.cost.Total + sum(x.cost.rho*x.m.SBase* ...
                                       [x.var.rhoH; x.var.rhoL]);
  end %if

%=========================================================================
% Compute mismatches and get ready for print statements.

% Print statement shows (total) power generated at gen bus, which is
% (Pbus + Pl) and mismatch power at load bus and fixed gen bus
% Mismatch Equations.  Pg, Qg, Pl, Ql are used because we need to have the
% vectors of the same size (NBus).

% First, replace PG in Pg, QG in Qg, PL in Pl, and QL in Ql.
  x.var.Pg(x.ind.PFree)   = x.var.PG;
  x.var.Qg(x.ind.QFree)   = x.var.QG;
  x.var.Pl(x.ind.Loadbus) = x.var.PL;
  x.var.Ql(x.ind.Loadbus) = x.var.QL;

  DeltaP = x.var.Pbus + x.var.Pl - x.var.Pg;
  DeltaQ = x.var.Qbus + x.var.Ql - x.var.Qg;

  x.print.P(x.ind.PFree,1)  = x.var.PG;
  x.print.P(x.ind.PFixed,1) = DeltaP(x.ind.PFixed);
  x.print.Q(x.ind.QFree,1)  = x.var.QG;
  x.print.Q(x.ind.QFixed,1) = DeltaQ(x.ind.QFixed);

%=========================================================================
%=========================================================================
% Derivatives.
% 1st order derivatives of I wrt X

  dIdV_f = (ejAngl_diag)*(YBPHIT);
  dIdV_t = (-YPHIT)*(ejAngl_diag);
  dIdV_f_v = dIdV_f(find(x.m.ft2i));
  dIdV_t_v = dIdV_t(find(x.m.ft2i));

% Don't take derivatives of I wrt Angl of Slack bus
  f_not_slack = find(x.ind.f ~= x.ind.Slackbus);
  t_not_slack = find(x.ind.t ~= x.ind.Slackbus);
  f_addAngl   = x.ind.f+x.n.NBus-1;
  t_addAngl   = x.ind.t+x.n.NBus-1;
  f_Angl      = f_addAngl(f_not_slack);
  t_Angl      = t_addAngl(t_not_slack);

  dIdAngl_f = (j*Vcomp_diag)*(YBPHIT);
  dIdAngl_t = (-YPHIT)*(j*Vcomp_diag);
  dIdAngl_f_v = dIdAngl_f(find(x.m.ft2i));
  dIdAngl_t_v = dIdAngl_t(find(x.m.ft2i));
  dIdAngl_f_v = dIdAngl_f_v(f_not_slack);
  dIdAngl_t_v = dIdAngl_t_v(t_not_slack);

  find_Tr_ft = ind2(x.m.ft2i,x.ind.fTr,x.ind.tTr);
  find_Tr_tf = ind2(x.m.ft2i,x.ind.tTr,x.ind.fTr);
```

```
  dIdTr_ft = (YBPHI_v(find_Tr_ft)).*(Vcomp(x.ind.fTr));
  dIdTr_tf = (-YPHI_v(find_Tr_tf)).*(Vcomp(x.ind.fTr));

  find_Phi_ft = ind2(x.m.ft2i,x.ind.fPhi,x.ind.tPhi);
  find_Phi_tf = ind2(x.m.ft2i,x.ind.tPhi,x.ind.fPhi);

  dIdPhi_ft = j*(YBPHIT_v(find_Phi_ft)).*(Vcomp(x.ind.fPhi));
  dIdPhi_tf = -j*(YPHIT_v(find_Phi_tf)).*(Vcomp(x.ind.fPhi));

  dIdX = sparse( [x.ind.f; x.ind.t; f_Angl; t_Angl; x.ind.Tr; x.ind.Tr; ...
                  x.ind.Phi; x.ind.Phi], ...
                 [(1:x.n.NI)'; (1:x.n.NI)'; f_not_slack; t_not_slack; ...
                   find_Tr_ft; find_Tr_tf; find_Phi_ft; find_Phi_tf], ...
                 [dIdV_f_v; dIdV_t_v; dIdAngl_f_v; dIdAngl_t_v; ...
                  dIdTr_ft; dIdTr_tf; dIdPhi_ft; dIdPhi_tf], ...
                  x.n.NX, x.n.NI );

%=========================================================================
  conj_dIdV_f = conj(dIdV_f);
  conj_dIdV_t = conj(dIdV_t);
  conj_dIdAngl_f = conj(dIdAngl_f);
  conj_dIdAngl_t = conj(dIdAngl_t);
  conj_dIdTr_ft = conj(dIdTr_ft);
  conj_dIdTr_tf = conj(dIdTr_tf);
  conj_dIdPhi_ft = conj(dIdPhi_ft);
  conj_dIdPhi_tf = conj(dIdPhi_tf);

  TejPHIminus1_Tft = ind2(TejPHIminus1,x.ind.fTr,x.ind.tTr);
  TejPHIminus1_Ttf = ind2(TejPHIminus1,x.ind.tTr,x.ind.fTr);
  ejPHIminus1_Tft  = ind2(ejPHIminus1,x.ind.fTr,x.ind.tTr);
  ejPHIminus1_Ttf  = ind2(ejPHIminus1,x.ind.tTr,x.ind.fTr);
  TejPHIminus1_Phift = ind2(TejPHIminus1,x.ind.fPhi,x.ind.tPhi);
  TejPHIminus1_Phitf = ind2(TejPHIminus1,x.ind.tPhi,x.ind.fPhi);
%=========================================================================
% 1st order derivatives of S wrt X

  tmp = (V_diag*conj_dIdV_f) + Imatrix_conj;
  dSdV_f = ejAngl_diag * (TejPHIminus1.*tmp + tmp);
  dSdV_t = Vcomp_diag * (TejPHIminus1.*conj_dIdV_t + conj_dIdV_t);
  dSdV_f_v = nonzeros(dSdV_f);
  dSdV_t_v = nonzeros(dSdV_t);

  tmp = conj_dIdAngl_f + j*Imatrix_conj;
  dSdAngl_f = Vcomp_diag * (TejPHIminus1.*tmp + tmp);
  dSdAngl_t = Vcomp_diag * (TejPHIminus1.*conj_dIdAngl_t + conj_dIdAngl_t);
  dSdAngl_f_v = dSdAngl_f(find(x.m.ft2i));
  dSdAngl_t_v = dSdAngl_t(find(x.m.ft2i));
  dSdAngl_f_v = dSdAngl_f_v(f_not_slack);
  dSdAngl_t_v = dSdAngl_t_v(t_not_slack);

  tmp = (TejPHIminus1_Tft.*conj_dIdTr_ft + conj_dIdTr_ft) + ...
        (ejPHIminus1_Tft.*Iconj(find_Tr_ft) + Iconj(find_Tr_ft));
  dSdTr_ft = Vcomp(x.ind.fTr).*tmp;
  dSdTr_tf = Vcomp(x.ind.tTr).* ...
           (TejPHIminus1_Ttf.*conj_dIdTr_tf + conj_dIdTr_tf);

  tmp = conj_dIdPhi_ft + j*Iconj(find_Phi_ft);
  dSdPhi_ft = Vcomp(x.ind.fPhi).* (TejPHIminus1_Phift.*tmp + tmp);
  dSdPhi_tf = Vcomp(x.ind.tPhi).* ...
             (TejPHIminus1_Phitf.*conj_dIdPhi_tf + conj_dIdPhi_tf);

  dSdX = sparse([x.ind.f; x.ind.t; f_Angl; t_Angl; ...
                 x.ind.Tr; x.ind.Tr; x.ind.Phi; x.ind.Phi],...
                [(1:x.n.NI)'; (1:x.n.NI)'; f_not_slack; t_not_slack; ...
                 find_Tr_ft; find_Tr_tf; find_Phi_ft; find_Phi_tf], ...
                [dSdV_f_v; dSdV_t_v; dSdAngl_f_v; dSdAngl_t_v; ...
                 dSdTr_ft; dSdTr_tf; dSdPhi_ft; dSdPhi_tf], ...
                x.n.NX,x.n.NI);
```

```
  dPdX = real(dSdX);
  dQdX = imag(dSdX);

%=========================================================================
% 1st order derivatives of |I| wrt X

  dIdX_real = real(dIdX);
  dIdX_imag = imag(dIdX);

  dIabsdX = dIdX_real * spdiags((Ireal./Iabs),0,x.n.NI,x.n.NI) + ...
            dIdX_imag * spdiags((Iimag./Iabs),0,x.n.NI,x.n.NI);
%=========================================================================
% 2nd order derivatives of I wrt X

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt |V|
% sdIdV are all zeros.

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt Angl

  sdIdAngl_f = -Vcomp_diag * YBPHIT;
  sdIdAngl_t = YPHIT * Vcomp_diag;
  sdIdAngl_f_v = sdIdAngl_f(find(x.m.ft2i));
  sdIdAngl_t_v = sdIdAngl_t(find(x.m.ft2i));
  sdIdAngl_f_v = sdIdAngl_f_v(f_not_slack);
  sdIdAngl_t_v = sdIdAngl_t_v(t_not_slack);

  v_Angl = [sdIdAngl_f_v
            sdIdAngl_t_v];
  i_Angl = [f_Angl
            t_Angl];
  j_Angl = [f_Angl
            t_Angl];
  k_Angl = [f_not_slack
            t_not_slack];

%-------------------------------------------------------------------------
% 2nd order derivative of I wrt Tr
% sdIdTr are all zeros

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt Phi

  sdIdPhi_ft = -YBPHIT_v(find_Phi_ft).*Vcomp(x.ind.fPhi);
  sdIdPhi_tf = YPHIT_v(find_Phi_tf).*Vcomp(x.ind.fPhi);

  v_Phi = [sdIdPhi_ft
           sdIdPhi_tf];
  i_Phi = [x.ind.Phi
           x.ind.Phi];
  j_Phi = [x.ind.Phi
           x.ind.Phi];
  k_Phi = [find_Phi_ft
           find_Phi_tf];

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt |V| and Angl

  sdIdVdAngl_f = (j*ejAngl_diag)*YBPHIT;
  sdIdVdAngl_t = -YPHIT*(j*ejAngl_diag);
  sdIdVdAngl_f_v = sdIdVdAngl_f(find(x.m.ft2i));
  sdIdVdAngl_t_v = sdIdVdAngl_t(find(x.m.ft2i));
  sdIdVdAngl_f_v = sdIdVdAngl_f_v(f_not_slack);
  sdIdVdAngl_t_v = sdIdVdAngl_t_v(t_not_slack);

  v_VAngl = [sdIdVdAngl_f_v
```

```
            sdIdVdAngl_t_v];
  i_VAngl = [x.ind.f(f_not_slack)
             x.ind.t(t_not_slack)];
  j_VAngl = [f_Angl
             t_Angl];
  k_VAngl = k_Angl;

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt |V| and Tr

  sdIdVdTr_fft = YBPHI_v(find_Tr_ft).*ejAngl(x.ind.fTr);
  sdIdVdTr_ttf = -YPHI_v(find_Tr_tf).*ejAngl(x.ind.fTr);

  v_VTr = [sdIdVdTr_fft
           sdIdVdTr_ttf];
  i_VTr = [x.ind.fTr
           x.ind.fTr];
  j_VTr = [x.ind.Tr
           x.ind.Tr];
  k_VTr = [find_Tr_ft
           find_Tr_tf];

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt |V| and Phi

  sdIdVdPhi_fft = j*YBPHIT_v(find_Phi_ft).*ejAngl(x.ind.fPhi);
  sdIdVdPhi_ttf =-j*YPHIT_v(find_Phi_tf).*ejAngl(x.ind.fPhi);

  v_VPhi = [sdIdVdPhi_fft
            sdIdVdPhi_ttf];
  i_VPhi = [x.ind.fPhi
            x.ind.fPhi];
  j_VPhi = [x.ind.Phi
            x.ind.Phi];
  k_VPhi = k_Phi;

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt Angl and Tr

% For x.ind.fTr same as Slack bus, no derivatives of I wrt Angl and Tr.

  if ~isempty(x.var.Tr) % If there is no Tr, the 1st line of this part will
                        % give a wrong value.;
    fTr_not_slack = find(x.ind.fTr~=x.ind.Slackbus);
    fTr_used = x.ind.fTr(fTr_not_slack);
    tTr_used = x.ind.tTr(fTr_not_slack);
    Tr_ind_used = x.ind.Tr(fTr_not_slack);
    find_Tr_used_ft = find_Tr_ft(fTr_not_slack); % Only when Angl is involved.
    find_Tr_used_tf = find_Tr_tf(fTr_not_slack);

    sdIdAngldTr_fft = j*YBPHI_v(find_Tr_used_ft).*Vcomp(fTr_used);
    sdIdAngldTr_ttf =-j*YPHI_v(find_Tr_used_tf).*Vcomp(fTr_used);

    conj_sdIdAngldTr_fft = conj(sdIdAngldTr_fft);
    conj_sdIdAngldTr_ttf = conj(sdIdAngldTr_ttf);

    v_AnglTr = [sdIdAngldTr_fft
                sdIdAngldTr_ttf];
    i_AnglTr = [fTr_used+x.n.NBus-1
                fTr_used+x.n.NBus-1];
    j_AnglTr = [Tr_ind_used
                Tr_ind_used];
    k_AnglTr = [find_Tr_used_ft
                find_Tr_used_tf];

  else
    v_AnglTr = [];
    i_AnglTr = [];
```

```
    j_AnglTr = [];
    k_AnglTr = [];

  end%if ~isempty...

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt Angl and Phi

% For x.ind.fPhi same as Slack bus, no derivatives of I wrt Angl and Phi.

  if ~isempty(x.var.Phi) % If there is no Phi, the 1st line of this part will
                         % give a wrong value.;
    fPhi_not_slack = find(x.ind.fPhi~=x.ind.Slackbus);
    fPhi_used = x.ind.fPhi(fPhi_not_slack);
    tPhi_used = x.ind.tPhi(fPhi_not_slack);
    Phi_ind_used = x.ind.Phi(fPhi_not_slack);
    find_Phi_used_ft = find_Phi_ft(fPhi_not_slack);
    find_Phi_used_tf = find_Phi_tf(fPhi_not_slack);

    sdIdAngldPhi_fft = -YBPHIT_v(find_Phi_used_ft).*Vcomp(fPhi_used);
    sdIdAngldPhi_ttf =  YPHIT_v(find_Phi_used_tf).*Vcomp(fPhi_used);

    v_AnglPhi = [sdIdAngldPhi_fft
                 sdIdAngldPhi_ttf];
    i_AnglPhi = [fPhi_used+x.n.NBus-1
                 fPhi_used+x.n.NBus-1];
    j_AnglPhi = [Phi_ind_used
                 Phi_ind_used];
    k_AnglPhi = [find_Phi_used_ft
                 find_Phi_used_tf];

    conj_sdIdAngldPhi_fft = conj(sdIdAngldPhi_fft);
    conj_sdIdAngldPhi_ttf = conj(sdIdAngldPhi_ttf);

  else
    v_AnglPhi = [];
    i_AnglPhi = [];
    j_AnglPhi = [];
    k_AnglPhi = [];

  end%if ~isempty...

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt Tr and Phi

% The derivatives of I wrt Tr and Phi are computed only when Tr and Phi
% are at the same location.  So first, we have to find out if this
% situation exists.  If it does, what is its location?

  TrPhi = nonzeros(ind2(x.m.ft2iTr,x.ind.fPhi,x.ind.tPhi));
                                % List of Tr at same location as Phi.
  PhiTr = nonzeros(ind2(x.m.ft2iPhi,x.ind.fTr,x.ind.tTr));
                                % List of Phi at same location as Tr.
  if ~isempty(TrPhi)
    fTrPhi = x.ind.fTr(TrPhi);      % f of line that has both Tr and Phi.
                                    % (same as fPhiTr=x.ind.fPhi(PhiTr).)
    tTrPhi = x.ind.tTr(TrPhi);      % t of line that has both Tr and Phi.
                                    % (same as tPhiTr=x.ind.tPhi(PhiTr).)
    Tr_ind_TrPhi = x.ind.Tr(TrPhi);
    Phi_ind_PhiTr = x.ind.Phi(PhiTr);
    find_TrPhi_ft = ind2(x.m.ft2i,fTrPhi,tTrPhi);
    find_TrPhi_tf = ind2(x.m.ft2i,tTrPhi,fTrPhi);

    sdIdTrdPhi_ft = j*YBPHI_v(find_TrPhi_ft).* Vcomp(fTrPhi);
    sdIdTrdPhi_tf =-j*YPHI_v(find_TrPhi_tf).* Vcomp(fTrPhi);

    v_TrPhi = [sdIdTrdPhi_ft
               sdIdTrdPhi_tf];
```

```
    i_TrPhi = [Tr_ind_TrPhi
               Tr_ind_TrPhi];
    j_TrPhi = [Phi_ind_PhiTr
               Phi_ind_PhiTr];
    k_TrPhi = [find_TrPhi_ft
               find_TrPhi_tf];

    conj_sdIdTrdPhi_ft = conj(sdIdTrdPhi_ft);
    conj_sdIdTrdPhi_tf = conj(sdIdTrdPhi_tf);
  else
    v_TrPhi = [];
    i_TrPhi = [];
    j_TrPhi = [];
    k_TrPhi = [];

    conj_sdIdTrdPhi_ft = [];
    conj_sdIdTrdPhi_tf = [];
  end %if

%-------------------------------------------------------------------------
% 2nd order derivatives of I wrt X
  sdIdX = spalloc((x.n.NI*x.n.NX),x.n.NX,0);

  iI = [i_Angl; i_Phi; i_VAngl; i_VTr; i_VPhi; i_AnglTr; i_AnglPhi; i_TrPhi];

  jI = [j_Angl; j_Phi; j_VAngl; j_VTr; j_VPhi; j_AnglTr; j_AnglPhi; j_TrPhi];

  kI = [k_Angl; k_Phi; k_VAngl; k_VTr; k_VPhi; k_AnglTr; k_AnglPhi; k_TrPhi];

  vI = [v_Angl; v_Phi; v_VAngl; v_VTr; v_VPhi; v_AnglTr; v_AnglPhi; v_TrPhi];

  sdIdX = assign3(sdIdX, [iI; jI], [jI; iI], [kI; kI], [vI; vI], ...
                  x.n.NX, x.n.NI);

%=========================================================================
% 2nd order derivatives of |I| wrt X

% Put each column of dIdX next to each other x.n.NX times
% [dI1dX ... dI1dX, dI2dX ... dI2dX, ...]
a = 1:x.n.NI;                       % a and aa are temporary variables
aa = a(ones(x.n.NX,1),:);           % for making some weird but useful matrices.
dIdX_stack = dIdX(:,aa(:));
dIabsdX_stack = dIabsdX(:,aa(:));

dIdX_diag = spdiags(dIdX(:),0,x.n.NX*x.n.NI,x.n.NX*x.n.NI);

dIdX_real_sq = real(dIdX_diag)*real(dIdX_stack.');
dIdX_imag_sq = imag(dIdX_diag)*imag(dIdX_stack.');
dIabsdX_real_sq = real(dIdX_diag)*dIabsdX_stack.';
dIabsdX_imag_sq = imag(dIdX_diag)*dIabsdX_stack.';

Ireal_diag_stack = kron(spdiags(Ireal,0,x.n.NI,x.n.NI),speye(x.n.NX,x.n.NX));
Iimag_diag_stack = kron(spdiags(Iimag,0,x.n.NI,x.n.NI),speye(x.n.NX,x.n.NX));
Iabs_diag_stack  = kron(spdiags(Iabs,0,x.n.NI,x.n.NI),speye(x.n.NX,x.n.NX));
Iabs_inv_stack = kron(spdiags((1./Iabs),0,x.n.NI,x.n.NI),speye(x.n.NX,x.n.NX));
Iabs2_inv_stack = Iabs_inv_stack.*Iabs_inv_stack;

sdIabsdX = Iabs_inv_stack*( Ireal_diag_stack*real(sdIdX) + dIdX_real_sq + ...
                            Iimag_diag_stack*imag(sdIdX) + dIdX_imag_sq )...
         - Iabs2_inv_stack*( Ireal_diag_stack*dIabsdX_real_sq + ...
                             Iimag_diag_stack*dIabsdX_imag_sq );

%=========================================================================
% Some useful terms evaluated from derivatives of I wrt X to be used in
% derivatives of S wrt X.

  conj_sdIdAngl_f = conj(sdIdAngl_f);
  conj_sdIdAngl_t = conj(sdIdAngl_t);
```

```
  conj_sdIdPhi_ft = conj(sdIdPhi_ft);
  conj_sdIdPhi_tf = conj(sdIdPhi_tf);
  conj_sdIdVdAngl_f = conj(sdIdVdAngl_f);
  conj_sdIdVdAngl_t = conj(sdIdVdAngl_t);
  conj_sdIdVdTr_fft = conj(sdIdVdTr_fft);
  conj_sdIdVdTr_ttf = conj(sdIdVdTr_ttf);
  conj_sdIdVdPhi_fft = conj(sdIdVdPhi_fft);
  conj_sdIdVdPhi_ttf = conj(sdIdVdPhi_ttf);
% conj_sdIdAngldTr_fft = conj(sdIdAngldTr_fft);
% conj_sdIdAngldTr_ttf = conj(sdIdAngldTr_ttf);
% conj_sdIdAngldPhi_fft = conj(sdIdAngldPhi_fft);
% conj_sdIdAngldPhi_ttf = conj(sdIdAngldPhi_ttf);

%=========================================================================
% 2nd order derivatives of S wrt X

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt |V|

  sdSdV_f  = ejAngl_diag * 2 *(TejPHIminus1.* conj_dIdV_f + conj_dIdV_f);
  sdSdV_ft = ejAngl_diag * (TejPHIminus1.* conj_dIdV_t + conj_dIdV_t);
  sdSdV_f_v  = nonzeros(sdSdV_f);
  sdSdV_ft_v = nonzeros(sdSdV_ft);

  vs_V = [sdSdV_f_v
          sdSdV_ft_v];
  is_V = [x.ind.f
          x.ind.f];
  js_V = [x.ind.f
          x.ind.t];
  ks_V = [(1:x.n.NI)'
          (1:x.n.NI)'];

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt Angl

% Don't take derivatives of S wrt slackbus angles.
  ft_not_slack = find((x.ind.f~=x.ind.Slackbus) & (x.ind.t~=x.ind.Slackbus));
  ft_Angl = f_addAngl(ft_not_slack);
  tf_Angl = t_addAngl(ft_not_slack);

  tmp = conj_sdIdAngl_f + 2*j*conj_dIdAngl_f - Imatrix_conj;
  sdSdAngl_f = Vcomp_diag * (TejPHIminus1.* tmp + tmp);
  sdSdAngl_t = Vcomp_diag * (TejPHIminus1.* conj_sdIdAngl_t + conj_sdIdAngl_t);
  sdSdAngl_ft = j*Vcomp_diag*(TejPHIminus1.* conj_dIdAngl_t + conj_dIdAngl_t);
  sdSdAngl_f_v  = sdSdAngl_f(find(x.m.ft2i));
  sdSdAngl_t_v  = sdSdAngl_t(find(x.m.ft2i));
  sdSdAngl_ft_v = sdSdAngl_ft(find(x.m.ft2i));
  sdSdAngl_f_v = sdSdAngl_f_v(f_not_slack);
  sdSdAngl_t_v = sdSdAngl_t_v(t_not_slack);
  sdSdAngl_ft_v = sdSdAngl_ft_v(ft_not_slack);

  vs_Angl = [sdSdAngl_f_v
             sdSdAngl_t_v
             sdSdAngl_ft_v];
  is_Angl = [f_Angl
             t_Angl
             ft_Angl];
  js_Angl = [f_Angl
             t_Angl
             tf_Angl];
  ks_Angl = [f_not_slack
             t_not_slack
             ft_not_slack];

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt Tr
```

```
  sdSdTr_ft = 2*Vcomp(x.ind.fTr).*(ejPHIminus1_Tft.* conj_dIdTr_ft + ...
              conj_dIdTr_ft);

  vs_Tr = sdSdTr_ft;
  is_Tr = [x.ind.Tr];
  js_Tr = [x.ind.Tr];
  ks_Tr = [find_Tr_ft];

%--------------------------------------------------------------------------
% 2nd order derivatives of S wrt Phi

  tmp = conj_sdIdPhi_ft + 2*j*conj_dIdPhi_ft - Iconj(find_Phi_ft);
  sdSdPhi_ft = Vcomp(x.ind.fPhi).*(TejPHIminus1_Phift.* tmp + tmp);
  sdSdPhi_tf = Vcomp(x.ind.tPhi).* ...
               (TejPHIminus1_Phitf.* conj_sdIdPhi_tf + conj_sdIdPhi_tf);

  vs_Phi = [sdSdPhi_ft
            sdSdPhi_tf];
  is_Phi = i_Phi;
  js_Phi = j_Phi;
  ks_Phi = k_Phi;

%--------------------------------------------------------------------------
% 2nd order derivatives of S wrt |V| and Angl

  f_t_not_slack = x.ind.f(t_not_slack);          % For S wrt V_f, Angl_t.
  t_f_not_slack = x.ind.t(f_not_slack);          % For S wrt V_t, Angl_f.

  tmp = Vcomp_diag * (conj_sdIdVdAngl_f + j*conj_dIdV_f) + ...
        ejAngl_diag * (conj_dIdAngl_f + j*Imatrix_conj);
  sdSdVdAngl_f = TejPHIminus1.*tmp + tmp;
  sdSdVdAngl_t = Vcomp_diag*(TejPHIminus1.* conj_sdIdVdAngl_t + ...
                             conj_sdIdVdAngl_t);
  sdSdVdAngl_ft = ejAngl_diag*(TejPHIminus1.* conj_dIdAngl_t + conj_dIdAngl_t);
  sdSdVdAngl_tf = j*Vcomp_diag*(TejPHIminus1.* conj_dIdV_t + conj_dIdV_t);

  sdSdVdAngl_f_v  = sdSdVdAngl_f(find(x.m.ft2i));
  sdSdVdAngl_t_v  = sdSdVdAngl_t(find(x.m.ft2i));
  sdSdVdAngl_ft_v = sdSdVdAngl_ft(find(x.m.ft2i));
  sdSdVdAngl_tf_v = sdSdVdAngl_tf(find(x.m.ft2i));
  sdSdVdAngl_f_v  = sdSdVdAngl_f_v(f_not_slack);
  sdSdVdAngl_t_v  = sdSdVdAngl_t_v(t_not_slack);
  sdSdVdAngl_ft_v = sdSdVdAngl_ft_v(t_not_slack);
  sdSdVdAngl_tf_v = sdSdVdAngl_tf_v(f_not_slack);

  vs_VAngl = [sdSdVdAngl_f_v
              sdSdVdAngl_t_v
              sdSdVdAngl_ft_v
              sdSdVdAngl_tf_v];
  is_VAngl = [i_VAngl
              f_t_not_slack
              t_f_not_slack];
  js_VAngl = [j_VAngl
              t_Angl
              f_Angl];
  ks_VAngl = [k_VAngl
              t_not_slack
              f_not_slack];

%--------------------------------------------------------------------------
% 2nd order derivatives of S wrt |V| and Tr

  tmp1 = Vcomp(x.ind.fTr).* conj_sdIdVdTr_fft;
  tmp2 = Vcomp(x.ind.fTr).* ind2(conj_dIdV_f,x.ind.fTr,x.ind.tTr);
  tmp3 = ejAngl(x.ind.fTr).* conj_dIdTr_ft;
  tmp4 = ejAngl(x.ind.fTr).* Iconj(find_Tr_ft);
  sdSdVdTr_fft = (TejPHIminus1_Tft.* tmp1 + tmp1) + ...
                 (ejPHIminus1_Tft.* tmp2 + tmp2) + ...
```

```
                (TejPHIminus1_Tft.* tmp3 + tmp3) + ...
                (ejPHIminus1_Tft.* tmp4 + tmp4);

  tmp = Vcomp(x.ind.tTr).* conj_sdIdVdTr_ttf;
  sdSdVdTr_ttf = TejPHIminus1_Ttf.* tmp + tmp;

  tmp = ejAngl(x.ind.tTr).* conj_dIdTr_tf;
  sdSdVdTr_ftf = TejPHIminus1_Ttf.* tmp + tmp;

  tmp = Vcomp(x.ind.fTr).* ind2(conj_dIdV_t,x.ind.fTr,x.ind.tTr);
  sdSdVdTr_tft = ejPHIminus1_Tft.* tmp + tmp;

  vs_VTr = [sdSdVdTr_fft
            sdSdVdTr_ttf
            sdSdVdTr_ftf
            sdSdVdTr_tft];
  is_VTr = [i_VTr
            x.ind.tTr
            x.ind.tTr];
  js_VTr = [j_VTr
            j_VTr];
  ks_VTr = [k_VTr
            k_VTr];

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt |V| and Phi

  tmp = x.var.V(x.ind.fPhi).* ...
         (conj_sdIdVdPhi_fft + j*ind2(conj_dIdV_f,x.ind.fPhi,x.ind.tPhi)) +...
         conj_dIdPhi_ft + j*Iconj(find_Phi_ft);
  sdSdVdPhi_fft = ejAngl(x.ind.fPhi).* (TejPHIminus1_Phift.* tmp + tmp);

  tmp = Vcomp(x.ind.tPhi).* conj_sdIdVdPhi_ttf;
  sdSdVdPhi_ttf = TejPHIminus1_Phitf.* tmp + tmp;

  tmp = ejAngl(x.ind.tPhi).* conj_dIdPhi_tf;
  sdSdVdPhi_ftf = TejPHIminus1_Phitf.* tmp + tmp;

  tmp = j*Vcomp(x.ind.fPhi).* ind2(conj_dIdV_t,x.ind.fPhi,x.ind.tPhi);
  sdSdVdPhi_tft = TejPHIminus1_Phift.* tmp + tmp;

  vs_VPhi = [sdSdVdPhi_fft
             sdSdVdPhi_ttf
             sdSdVdPhi_ftf
             sdSdVdPhi_tft];
  is_VPhi = [i_VPhi
             x.ind.tPhi
             x.ind.tPhi];
  js_VPhi = [j_VPhi
             j_VPhi];
  ks_VPhi = [k_VPhi
             k_VPhi];

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt Angl and Tr

% For x.ind.fTr same as Slack bus, no derivatives of S wrt Angl and Tr.
% (Already done in 2nd derivatives of I wrt Angl and Tr).
%   fTr_not_slack = find(x.ind.fTr~=x.ind.Slackbus);
%   fTr_used = x.ind.fTr(fTr_not_slack);
%   tTr_used = x.ind.tTr(fTr_not_slack);
%   Tr_ind_used = x.ind.Tr(fTr_not_slack);
%   find_Tr_used_ft = find_Tr_ft(fTr_not_slack); % Only when Angl is involved.
%   find_Tr_used_tf = find_Tr_tf(fTr_not_slack);

% For x.ind.tTr same as Slack bus, no derivatives of S wrt Angl and Tr.

  if ~isempty(x.var.Tr) % If there is no Tr, the 1st line of this part will
```

```
                        % give a wrong value.;
    tTr_not_slack = find(x.ind.tTr~=x.ind.Slackbus);
    fTr_used2 = x.ind.fTr(tTr_not_slack);
    tTr_used2 = x.ind.tTr(tTr_not_slack);
    Tr_ind_used2 = x.ind.Tr(tTr_not_slack);
    find_Tr_used2_ft = find_Tr_ft(tTr_not_slack); % Only when Angl is involved.
    find_Tr_used2_tf = find_Tr_tf(tTr_not_slack);

    tmp1 = conj_sdIdAngldTr_fft + j*conj_dIdTr_ft(fTr_not_slack);
    tmp2 = ind2(conj_dIdAngl_f,fTr_used,tTr_used) + j*Iconj(find_Tr_used_ft);
    sdSdAngldTr_fft = Vcomp(fTr_used).* ...
                      ((TejPHIminus1_Tft(fTr_not_slack).* tmp1 + tmp1) + ...
                       (ejPHIminus1_Tft(fTr_not_slack).* tmp2 + tmp2));

    tmp = Vcomp(tTr_used).* conj_sdIdAngldTr_ttf;
    sdSdAngldTr_ttf = TejPHIminus1_Ttf(fTr_not_slack).* tmp +tmp;

    tmp = j*Vcomp(tTr_used2).* conj_dIdTr_tf(tTr_not_slack);
    sdSdAngldTr_ftf = TejPHIminus1_Ttf(tTr_not_slack).* tmp + tmp;

    tmp = Vcomp(fTr_used2).* ind2(conj_dIdAngl_t,fTr_used2,tTr_used2);
    sdSdAngldTr_tft = ejPHIminus1_Tft(tTr_not_slack).* tmp + tmp;

    vs_AnglTr = [sdSdAngldTr_fft
                 sdSdAngldTr_ttf
                 sdSdAngldTr_ftf
                 sdSdAngldTr_tft];
    is_AnglTr = [i_AnglTr
                 tTr_used2+x.n.NBus-1
                 tTr_used2+x.n.NBus-1];
    js_AnglTr = [j_AnglTr
                 Tr_ind_used2
                 Tr_ind_used2];
    ks_AnglTr = [k_AnglTr
                 find_Tr_used2_tf
                 find_Tr_used2_ft];

  else
    vs_AnglTr = [];
    is_AnglTr = [];
    js_AnglTr = [];
    ks_AnglTr = [];

  end% if

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt Angl and Phi

% For x.ind.fPhi same as Slack bus, no derivatives of S wrt Angl and Phi.
% (Already done in 2nd derivatives of I wrt Angl and Phi).
%   fPhi_not_slack = find(x.ind.fPhi~=x.ind.Slackbus);
%   fPhi_used = x.ind.fPhi(fPhi_not_slack);
%   tPhi_used = x.ind.tPhi(fPhi_not_slack);
%   Phi_ind_used = x.ind.Phi(fPhi_not_slack);
%   find_Phi_used_ft = find_Phi_ft(fPhi_not_slack);
%   find_Phi_used_tf = find_Phi_tf(fPhi_not_slack);

  if ~isempty(x.var.Phi) % If there is no Phi, the 1st line of this part will
                         % give a wrong value.;
    tPhi_not_slack = find(x.ind.tPhi~=x.ind.Slackbus);
    fPhi_used2 = x.ind.fPhi(tPhi_not_slack);
    tPhi_used2 = x.ind.tPhi(tPhi_not_slack);
    Phi_ind_used2 = x.ind.Phi(tPhi_not_slack);
    find_Phi_used2_ft = find_Phi_ft(tPhi_not_slack);
    find_Phi_used2_tf = find_Phi_tf(tPhi_not_slack);

% Special TejPHIminus1's that are used here only.
    TejPHIminus1_Phift_used = ind2(TejPHIminus1,fPhi_used,tPhi_used);
```

```
    TejPHIminus1_Phitf_used = ind2(TejPHIminus1,tPhi_used,fPhi_used);
    TejPHIminus1_Phift_used2 = ind2(TejPHIminus1,fPhi_used2,tPhi_used2);
    TejPHIminus1_Phitf_used2 = ind2(TejPHIminus1,tPhi_used2,fPhi_used2);

    tmp = Vcomp(fPhi_used).*(conj_sdIdAngldPhi_fft + ...
                             j*ind2(conj_dIdAngl_f,fPhi_used,tPhi_used) +...
                             j*conj_dIdPhi_ft(fPhi_not_slack) ...
                             - Iconj(find_Phi_used_ft));
    sdSdAngldPhi_fft = TejPHIminus1_Phift_used.* tmp + tmp;

    tmp = Vcomp(tPhi_used).* conj_sdIdAngldPhi_ttf;
    sdSdAngldPhi_ttf = TejPHIminus1_Phitf_used.* tmp + tmp;

    tmp = j*Vcomp(tPhi_used2).* conj_dIdPhi_tf(tPhi_not_slack);
    sdSdAngldPhi_ftf = TejPHIminus1_Phitf_used2.* tmp + tmp;

    tmp = j*Vcomp(fPhi_used2).* ind2(conj_dIdAngl_t,fPhi_used2,tPhi_used2);
    sdSdAngldPhi_tft = TejPHIminus1_Phift_used2.* tmp + tmp;

    vs_AnglPhi = [sdSdAngldPhi_fft
                  sdSdAngldPhi_ttf
                  sdSdAngldPhi_ftf
                  sdSdAngldPhi_tft];
    is_AnglPhi = [i_AnglPhi
                  tPhi_used2+x.n.NBus-1
                  tPhi_used2+x.n.NBus-1];
    js_AnglPhi = [j_AnglPhi
                  Phi_ind_used2
                  Phi_ind_used2];
    ks_AnglPhi = [k_AnglPhi
                  find_Phi_used2_tf
                  find_Phi_used2_ft];

  else
    vs_AnglPhi = [];
    is_AnglPhi = [];
    js_AnglPhi = [];
    ks_AnglPhi = [];

  end %if

%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt Tr and Phi

% The derivatives of I wrt Tr and Phi are computed only when Tr and Phi
% are at the same location.
% (Already done in 2nd derivatives of I wrt Tr and Phi).
%   TrPhi = nonzeros(ind2(x.m.ft2iTr,x.ind.fPhi,x.ind.tPhi));
%                                  % List of Tr at same location as Phi.
%   PhiTr = nonzeros(ind2(x.m.ft2iPhi,x.ind.fTr,x.ind.tTr));
%                                  % List of Phi at same location as Tr.
%   fTrPhi = x.ind.fTr(TrPhi);     % f of line that has both Tr and Phi.
%                                  % (same as fPhiTr=x.ind.fPhi(PhiTr).)
%   tTrPhi = x.ind.tTr(TrPhi);     % t of line that has both Tr and Phi.
%                                  % (same as tPhiTr=x.ind.tPhi(PhiTr).)
%   Tr_ind_TrPhi = x.ind.Tr(TrPhi);
%   Phi_ind_PhiTr = x.ind.Phi(PhiTr);
%   find_TrPhi_ft = ind2(x.m.ft2i,fTrPhi,tTrPhi);
%   find_TrPhi_tf = ind2(x.m.ft2i,tTrPhi,fTrPhi);

  if ~isempty(TrPhi)
    tmp1 = conj_sdIdTrdPhi_ft + j*conj_dIdTr_ft(TrPhi);
    tmp2 = TejPHIminus1_Tft(TrPhi).*tmp1 + tmp1;
    tmp3 = conj_dIdPhi_ft(PhiTr) + j*Iconj(find_TrPhi_ft);
    tmp4 = ejPHIminus1_Tft(TrPhi).*tmp3 + tmp3;
    sdSdTrdPhi_ft = Vcomp(fTrPhi).* (tmp2 + tmp4);

    tmp = Vcomp(tTrPhi).*sdIdTrdPhi_tf;
```

```
    sdSdTrdPhi_tf = TejPHIminus1_Ttf(TrPhi).*tmp + tmp;

  else
    sdSdTrdPhi_ft = [];
    sdSdTrdPhi_tf = [];

  end %if

% The derivatives of S wrt Tr and Phi are also computed when Tr and Phi are at
% the different end of the same line.  That is Tr_ij and Phi_ji.

  TrPhi2 = nonzeros(ind2(x.m.ft2iTr,x.ind.tPhi,x.ind.fPhi));
                                    % List of Tr_ij that has Phi_ji.
  PhiTr2 = nonzeros(ind2(x.m.ft2iPhi,x.ind.tTr,x.ind.fTr));
                                    % List of Phi_ij that has Tr_ji.
  if ~isempty(TrPhi2)
    fTrPhi2 = x.ind.fTr(TrPhi2);    % f of Tr(ij) which is same as t of Phi(ji).
        % (same as tPhiTr2=x.ind.tPhi(PhiTr2).)
    tTrPhi2 = x.ind.tTr(TrPhi2);    % t of Tr(ij) which is same as f of Phi(ji).
        % (same as fPhiTr2=x.ind.fPhi(PhiTr2).)
    Tr_ind_TrPhi2  = x.ind.Tr(TrPhi2);
    Phi_ind_PhiTr2 = x.ind.Phi(PhiTr2);
    find_TrPhi2_ft = ind2(x.m.ft2i,fTrPhi2,tTrPhi2); % = find_PhiTr2_tf
    find_TrPhi2_tf = ind2(x.m.ft2i,tTrPhi2,fTrPhi2); % = find_PhiTr2_ft

    tmp = Vcomp(fTrPhi2).*conj_dIdPhi_tf(PhiTr2);
    sdSdTrdPhi_fttf = ejPHIminus1_Tft(TrPhi2).*tmp + tmp;

    tmp = Vcomp(tTrPhi2).* conj_dIdTr_tf(TrPhi2);
    sdSdTrdPhi_tfft = j*TejPHIminus1_Phift(PhiTr2).*tmp + tmp;

  else
    sdSdTrdPhi_fttf = [];
    sdSdTrdPhi_tfft = [];

    Tr_ind_TrPhi2  = [];
    Phi_ind_PhiTr2 = [];
    find_TrPhi2_ft = [];
    find_TrPhi2_tf = [];

  end %if

  vs_TrPhi = [sdSdTrdPhi_ft
              sdSdTrdPhi_tf
              sdSdTrdPhi_fttf
              sdSdTrdPhi_tfft];
  is_TrPhi = [i_TrPhi
              Tr_ind_TrPhi2
              Tr_ind_TrPhi2];
  js_TrPhi = [j_TrPhi
              Phi_ind_PhiTr2
              Phi_ind_PhiTr2];
  ks_TrPhi = [k_TrPhi
              find_TrPhi2_ft
              find_TrPhi2_tf];
%-------------------------------------------------------------------------
% 2nd order derivatives of S wrt X
  sdSdX = spalloc((x.n.NI*x.n.NX),x.n.NX,0);

  iS = [is_V; is_Angl; is_Tr; is_Phi; is_VAngl; is_VTr; is_VPhi; ...
        is_AnglTr; is_AnglPhi; is_TrPhi];

  jS = [js_V; js_Angl; js_Tr; js_Phi; js_VAngl; js_VTr; js_VPhi; ...
        js_AnglTr; js_AnglPhi; js_TrPhi];

  kS = [ks_V; ks_Angl; ks_Tr; ks_Phi; ks_VAngl; ks_VTr; ks_VPhi; ...
        ks_AnglTr; ks_AnglPhi; ks_TrPhi];
```

```
  vS = [vs_V; vs_Angl; vs_Tr; vs_Phi; vs_VAngl; vs_VTr; vs_VPhi; ...
        vs_AnglTr; vs_AnglPhi; vs_TrPhi];

  sdSdX = assign3(sdSdX, [iS; jS], [jS; iS], [kS; kS], [vS; vS], ...
                  x.n.NX, x.n.NI);

% Thus, 2nd order derivatives of Pij & Qij wrt to X
  sdPdX = real(sdSdX);
  sdQdX = imag(sdSdX);

%=========================================================================
% 1st order derivatives of P and Q wrt X

% Create a temporary matrix to convert dPij/dX to dPbus/dX.
% This matrix has 1's at the columns corresponding to x.ind.f
% (i.e we want P2 = P21+P23+P24+P25), and the rows corresponding to 1-x.n.NI
% (i.e the order of Pij).

  line2bus = sparse([1:x.n.NI],x.ind.f,ones(x.n.NI,1),x.n.NI,x.n.NBus);

  dPdX_bus = dPdX*line2bus;
  dQdX_bus = dQdX*line2bus;

  dPdX_fixed = dPdX_bus(:,x.ind.PFixed);     % goes to Jacobian.
  dPdX_free  = dPdX_bus(:,x.ind.PFree);      % goes to AMatrix.
  dQdX_fixed = dQdX_bus(:,x.ind.QFixed);     % goes to Jacobian.
  dQdX_free  = dQdX_bus(:,x.ind.QFree);      % goes to AMatrix.

%=========================================================================
% 2nd order derivatives of Lagrange function(L) wrt X
% which is part of Hessian in W matrix.

% Multipliers corresponding to Pij, Qij, Iij.
  Pmult = x.var.LambdaP(x.ind.f);
  Qmult = x.var.LambdaQ(x.ind.f);
  Imult = x.var.LambdaI(x.ind.f);

  LambdaP_kron = kron(Pmult',speye(x.n.NX));
  LambdaQ_kron = kron(Qmult',speye(x.n.NX));
  LambdaI_kron = kron(Imult',speye(x.n.NX));

  sdLdX = (LambdaP_kron*sdPdX) + (LambdaQ_kron*sdQdX) + (LambdaI_kron*sdIabsdX);
%=========================================================================
% 2nd order derivatives of Security cost (SC) wrt PG & PL.

  sdSCdPG = sparse(x.n.NPFree*x.n.K,x.n.NPFree);  % ????????
  Pi_PG = kron(x.m.Pi',speye(x.n.NPFree));

  sdSCdPL = sparse(x.n.NLoad*x.n.K,x.n.NLoad);    % ????????
  Pi_PL = kron(x.m.Pi',speye(x.n.NLoad));

  sdSCdPGPL = sparse(x.n.NPFree*x.n.K,x.n.NLoad); % ????????
  Pi_PGPL = kron(x.m.Pi',speye(x.n.NPFree));

%=========================================================================
% Form Jacobian matrix.

  dPGdX = spalloc(x.n.NPFree,x.n.NBus,x.n.NPFree);
  dPGdX(:,x.ind.PFree) = -speye(x.n.NPFree,x.n.NPFree);
  dQGdX = spalloc(x.n.NQFree,x.n.NBus,x.n.NQFree);
  dQGdX(:,x.ind.QFree) = -speye(x.n.NQFree,x.n.NQFree);
  dPLdX = spalloc(x.n.NLoad,x.n.NBus,x.n.NLoad);
  dPLdX(:,x.ind.Loadbus) = speye(x.n.NLoad,x.n.NLoad);
  dQLdPL = spdiags(x.m.LCoeff,0,x.n.NBus,x.n.NBus);
  dQLdPL = dQLdPL(x.ind.Loadbus,:);

  J = [ dPdX_bus     dQdX_bus                        dIabsdX
        dPGdX        sparse(x.n.NPFree,x.n.NBus)     sparse(x.n.NPFree,x.n.NI)
```

```
        sparse(x.n.NQFree,x.n.NBus)    dQGdX          sparse(x.n.NQFree,x.n.NI)
        dPLdX        dQLdPL                           sparse(x.n.NLoad,x.n.NI)
        sparse(x.n.NI,x.n.NBus) sparse(x.n.NI,x.n.NBus) -speye(x.n.NI,x.n.NI) ];

%=========================================================================
% Form Hessian matrix.
% Some terms in Hessian of main problem and subproblem are different
% due to the objective functions.  Those terms are sdLdPG, sdLdPL, and sdLdPGPL.
% sdLdQG and sdLdIs are empty and already included in H.

  if (k==0)                          % Main problem!
    sdLdPG = x.m.Pi0*spdiags(x.cost.d2,0,x.n.NPFree,x.n.NPFree);
    sdLdPL = x.m.Pi0*(-spdiags(x.cost.d2L,0,x.n.NLoad,x.n.NLoad));
  elseif (k > 0 & k <= x.n.K)        % Subproblem, k= 1,...,K.
    sdLdPG = (spdiags(x.cost.d2,0,x.n.NPFree,x.n.NPFree) + ...
              spdiags(x.cost.d2R,0,x.n.NPFree,x.n.NPFree))*x.m.Pi(k);
    sdLdPL = (-spdiags(x.cost.d2I,0,x.n.NLoad,x.n.NLoad) - ...
               spdiags(x.cost.d2L,0,x.n.NLoad,x.n.NLoad))*x.m.Pi(k);
  end %if k==0

  H = [sdLdX                      sparse(x.n.NX,x.n.NVar-x.n.NX)
       sparse(x.n.NPFree,x.n.NX)  sdLdPG   ...
       sparse(x.n.NPFree,x.n.NQFree+x.n.NLoad+x.n.NI)
       sparse(x.n.NQFree,x.n.NVar)
       sparse(x.n.NLoad,x.n.NX+x.n.NPFree+x.n.NQFree) ...
       sdLdPL                     sparse(x.n.NLoad,x.n.NI)
       sparse(x.n.NI,x.n.NVar)                                          ];

%=========================================================================
% Form W0 matrix which has no binding constraints.

  x.m.W0 = [ H     J
             J.'   sparse(2*x.n.NBus+x.n.NI,2*x.n.NBus+x.n.NI) ];

%=========================================================================
% Form A matrix (1st order derivatives of L wrt constrained variables).
% AH and AL are columns of A corresponding to binding constraints.

  x.m.A = [ speye(x.n.NVar,x.n.NVar) ...
            [sparse(x.n.NX,x.n.NPFree+x.n.NLoad)
             speye(x.n.NPFree,x.n.NPFree+x.n.NLoad)
             sparse(x.n.NQFree,x.n.NPFree+x.n.NLoad)
             sparse(x.n.NLoad,x.n.NPFree)  speye(x.n.NLoad,x.n.NLoad)
             sparse(x.n.NI,x.n.NPFree+x.n.NLoad)]
            sparse(2*x.n.NBus+x.n.NI,x.n.NVar+x.n.NPFree+x.n.NLoad) ];

  x.m.AH = x.m.A(:,x.ind.aH);
  x.m.AL = x.m.A(:,x.ind.aL);

%=========================================================================
% Form g matrix.
% If there are binding constraints, the derivatives of L wrt variables (X)
% are modified.
% Some of the terms in g matrix of main problem and subproblem are different
% due to the objective functions.  Those are dLdPG and dLdPL.

%-------------------------------------------------------------------------
  if k==0
     MuH_v = x.m.AH*x.var.MuH;
     MuL_v = x.m.AL*x.var.MuL;

     MuXH = MuH_v(1:x.n.NX);         % For dLdX.
     MuXL = MuL_v(1:x.n.NX);         % For dLdX.

     MuPGH = MuH_v(x.ind.PG);  % For dLdPG.  Also need these Mu's in SC.
     MuPGL = MuL_v(x.ind.PG);  % For dLdPG.

     MuQGH = MuH_v(x.ind.QG);  % For dLdQG.
```

```
    MuQGL = MuL_v(x.ind.QG);  % For dLdQG.

    MuPLH = MuH_v(x.ind.PL);  % For dLdPL.
    MuPLL = MuL_v(x.ind.PL);  % For dLdPL.

    MuIsH = MuH_v(x.ind.Is);        % For dLdIs.
    MuIsL = MuL_v(x.ind.Is);        % For dLdIs.
  else
    AH_k  = x.m.A(:,x.ind.aH_k);
    AH_k0 = x.m.A(:,x.ind.aH_k0);
    MuH_v = AH_k*x.var.MuH(1:x.n.NaH_k);
    MuH_k0 = AH_k0*x.var.MuH(x.n.NaH_k+1:x.n.NaH);

    AL_k  = x.m.A(:,x.ind.aL_k);
    AL_k0 = x.m.A(:,x.ind.aL_k0);
    MuL_v = AL_k*x.var.MuL(1:x.n.NaL_k);
    MuL_k0 = AL_k0*x.var.MuL(x.n.NaL_k+1:x.n.NaL);

    MuXH = MuH_v(1:x.n.NX);         % For dLdX.
    MuXL = MuL_v(1:x.n.NX);         % For dLdX.

    MuPGH = MuH_v(x.ind.PG);        % For dLdPG.
    MuPGL = MuL_v(x.ind.PG);        % For dLdPG.

    MuQGH = MuH_v(x.ind.QG);        % For dLdQG.
    MuQGL = MuL_v(x.ind.QG);        % For dLdQG.

    MuPLH = MuH_v(x.ind.PL);        % For dLdPL.
    MuPLL = MuL_v(x.ind.PL);        % For dLdPL.

    MuIsH = MuH_v(x.ind.Is);        % For dLdIs.
    MuIsL = MuL_v(x.ind.Is);        % For dLdIs.

    x.var.MuPGHC = MuH_k0(x.ind.PG);  % For main, need to sum these over k.
    x.var.MuPGLC = MuL_k0(x.ind.PG);

    x.var.MuPLHC = MuH_k0(x.ind.PL);
    x.var.MuPLLC = MuL_k0(x.ind.PL);
  end %if k==0

%-------------------------------------------------------------------------
  dLdX = dPdX_bus*x.var.LambdaP + dQdX_bus*x.var.LambdaQ + ...
         dIabsdX*x.var.LambdaI + MuXH - MuXL;
  dLdQG = -x.var.LambdaQ(x.ind.QFree) + (MuQGH - MuQGL);
  dLdIs = -x.var.LambdaI + (MuIsH - MuIsL);

  if k==0     % Main problem!
    if x.n.K==0
      MCR_sub = sparse(x.n.NPFree,x.n.NLine);
      MCI_sub = sparse(x.n.NLoad,x.n.NLine);
    else
      MCR_sub = kron(x.cost.MCR,ones(1,x.n.K)); % MCR of every sub.
      MCI_sub = kron(x.cost.MCI,ones(1,x.n.K)); % MCI of every sub.
    end %if x.n.K==0

    dLdPG = x.m.Pi0*x.cost.MC - MCR_sub*x.m.Pi - ...
            x.var.LambdaP(x.ind.PFree) + (MuPGH - MuPGL);
    dLdPL = x.m.Pi0*(-x.cost.MB) + MCI_sub*x.m.Pi + ...
            x.var.LambdaP(x.ind.Loadbus) + ...
            x.var.LambdaQ(x.ind.Loadbus).*x.m.LCoeff(x.ind.Loadbus) + ...
            (MuPLH - MuPLL);
  elseif (k > 0 & k <= x.n.K)          % Subproblem, k= 1,...,K.
    dLdPG = x.m.Pi(k)*(x.cost.MC + x.cost.MCR) - ...
            x.var.LambdaP(x.ind.PFree) + ...
           (MuPGH - MuPGL + x.var.MuPGHC - x.var.MuPGLC);
    dLdPL = x.m.Pi(k)*(-x.cost.MCI - x.cost.MB) + ...
            x.var.LambdaP(x.ind.Loadbus) + ...
            x.var.LambdaQ(x.ind.Loadbus).*x.m.LCoeff(x.ind.Loadbus) + ...
```

```
          (MuPLH - MuPLL + x.var.MuPLHC - x.var.MuPLLC);
  end %if k==0

%-------------------------------------------------------------------------
  dLdLambdaP = spalloc(x.n.NBus,1,x.n.NBus);
  dLdLambdaQ = spalloc(x.n.NBus,1,x.n.NBus);

  Pout = x.var.Pbus+x.var.Pl;          % Total P coming out of each bus.
  Qout = x.var.Qbus+x.var.Ql;          % Total Q coming out of each bus.

  dLdLambdaP_free  = Pout(x.ind.PFree) - x.var.PG;
  dLdLambdaP_fixed = Pout(x.ind.PFixed) - x.var.Pg(x.ind.PFixed);
  dLdLambdaQ_free  = Qout(x.ind.QFree) - x.var.QG;
  dLdLambdaQ_fixed = Qout(x.ind.QFixed) - x.var.Qg(x.ind.QFixed);

  dLdLambdaP(x.ind.PFree)  = dLdLambdaP_free;
  dLdLambdaP(x.ind.PFixed) = dLdLambdaP_fixed;
  dLdLambdaQ(x.ind.QFree)  = dLdLambdaQ_free;
  dLdLambdaQ(x.ind.QFixed) = dLdLambdaQ_fixed;

  dLdLambdaI = Iabs - x.var.Is;
%-------------------------------------------------------------------------
  x.m.g0 = [ dLdX
             dLdPG
             dLdQG
             dLdPL
             dLdIs
             dLdLambdaP
             dLdLambdaQ
             dLdLambdaI ];

%-------------------------------------------------------------------------
% Form vector Z at first iteration.

    if x.Iter==1
     x.var.Z = [ x.var.V
                 x.var.Angl
                 x.var.Tr
                 x.var.Phi
                 x.var.PG
                 x.var.QG
                 x.var.PL
                 x.var.Is
                 x.var.LambdaP
                 x.var.LambdaQ
                 x.var.LambdaI
                 x.var.MuH
                 x.var.MuL
                 x.var.MurhoH
                 x.var.MurhoL
                 x.var.sH
                 x.var.sL
                 x.var.rhoH
                 x.var.rhoL     ];

     x.var.Z(x.ind.Slack,:) = [];      % Take out angle of slack bus.
  end % if

%-------------------------------------------------------------------------
% Form W matrix using W0, AH and AL from PowerFlow.
% This W is only for IPRho.

  x.m.W = [ x.m.W0  x.m.AH  -x.m.AL ...
              sparse(x.n.NZ0, 3*(x.n.NaH+x.n.NaL))
            sparse(x.n.NaH+x.n.NaL, x.n.NZ0) ...
              -speye(x.n.NaH+x.n.NaL)     -speye(x.n.NaH+x.n.NaL) ...
              sparse(x.n.NaH+x.n.NaL,2*(x.n.NaH+x.n.NaL))
            [x.m.AH';-x.m.AL'] sparse(x.n.NaH+x.n.NaL, 2*(x.n.NaH+x.n.NaL)) ...
```

```
            speye(x.n.NaH+x.n.NaL)  -speye(x.n.NaH+x.n.NaL)
         sparse(2*(x.n.NaH+x.n.NaL), x.n.NZ0) ...
            spdiags([x.var.sH; x.var.sL; x.var.rhoH; x.var.rhoL], ...
                    0, 2*(x.n.NaH+x.n.NaL), 2*(x.n.NaH+x.n.NaL)) ...
            spdiags([x.var.MuH; x.var.MuL; x.var.MurhoH; x.var.MurhoL], ...
                    0, 2*(x.n.NaH+x.n.NaL), 2*(x.n.NaH+x.n.NaL))    ];

%-------------------------------------------------------------------------
% Modify vector g.  This vector is only for IPRho.

  x.m.g = [ x.m.g0
            x.init.cost.rho*ones(x.n.NaH,1) - x.var.MuH - x.var.MurhoH
            x.init.cost.rho*ones(x.n.NaL,1) - x.var.MuL - x.var.MurhoL
            x.var.Fn(x.ind.aH) - x.lim.Max(x.ind.aH) - x.var.rhoH + x.var.sH
            x.lim.Min(x.ind.aL) - x.var.Fn(x.ind.aL) - x.var.rhoL + x.var.sL
            x.var.MuH.*x.var.sH - x.init.m.sig*x.var.Nu(ones(x.n.NaH,1))
            x.var.MuL.*x.var.sL - x.init.m.sig*x.var.Nu(ones(x.n.NaL,1))
            x.var.MurhoH.*x.var.rhoH - x.init.m.sig*x.var.Nu(ones(x.n.NaH,1))
            x.var.MurhoL.*x.var.rhoL - x.init.m.sig*x.var.Nu(ones(x.n.NaL,1)) ];
%-------------------------------------------------------------------------
% Take a good look at g.

  g_var = x.m.g(1:x.n.NVar);
  g_lambda = x.m.g(x.n.NVar+1:x.n.NVar+2*x.n.NBus);
  g_Crho = x.m.g(x.n.NZ0+1:x.n.NZ0+x.n.NaH+x.n.NaL);
  g_slack = x.m.g(x.n.NZ0+x.n.NaH+x.n.NaL+1:x.n.NZ0+2*(x.n.NaH+x.n.NaL));
  g_comp = x.m.g(x.n.NZ0+2*(x.n.NaH+x.n.NaL)+1:x.n.NZ0+4*(x.n.NaH+x.n.NaL));

  norm_g.g_var = norm(g_var);
  norm_g.g_lambda = norm(g_lambda);
  norm_g.g_Crho = norm(g_Crho);
  norm_g.g_slack = norm(g_slack);
  norm_g.g_comp = norm(g_comp);

%=========================================================================
```

```
  function[x_all,converged] = IPRho(x_all,W,g,converged)

% function[x_all,converged] = IPRho(x_all,W,g,converged)

%=========================================================================
% Check to see if norm(g) is close to zero.

  fprintf('\n|g| = %8.4g\n', norm(g));

%-------------------------------------------------------------------------
% Calculate deltaZ.

  deltaZ = W\(-g);

  deltaMuH = deltaZ(x_all.ind.MuH);
  deltaMuL = deltaZ(x_all.ind.MuL);
  deltaMurhoH = deltaZ(x_all.ind.MurhoH);
  deltaMurhoL = deltaZ(x_all.ind.MurhoL);

  deltasH = deltaZ(x_all.ind.sH);
  deltasL = deltaZ(x_all.ind.sL);
  deltarhoH = deltaZ(x_all.ind.rhoH);
  deltarhoL = deltaZ(x_all.ind.rhoL);

%-------------------------------------------------------------------------
% Find binding constraints.  Only constratints in this IP are s,rho,Mu,Murho >0.

  iMuHNeg = find(deltaMuH < 0);        % ind of MuH that might be negative.
  iMuLNeg = find(deltaMuL < 0);        % ind of MuL that might be negative.
  iMurhoHNeg = find(deltaMurhoH < 0);  % ind of MurhoH that might be negative.
  iMurhoLNeg = find(deltaMurhoL < 0);  % ind of MurhoL that might be negative.
  isHNeg  = find(deltasH < 0);         % ind of sH that might be negative.
  isLNeg  = find(deltasL < 0);         % ind of sL that might be negative.
  irhoHNeg   = find(deltarhoH < 0);    % ind of rhoH that might be negative.
  irhoLNeg   = find(deltarhoL < 0);    % ind of rhoL that might be negative.

% Set value of constant that prevents variables from being zero.

  load const.dat

% Find Primal step.

  alpha_p0 = min([-x_all.var.sH(isHNeg)./deltasH(isHNeg)
                  -x_all.var.sL(isLNeg)./deltasL(isLNeg)
                  -x_all.var.rhoH(irhoHNeg)./deltarhoH(irhoHNeg)
                  -x_all.var.rhoL(irhoLNeg)./deltarhoL(irhoLNeg)]);
  alpha_p = min([1, const*alpha_p0]);
  fprintf('\nalpha_P = %8.4g', alpha_p);

% Find Dual step.

  alpha_d0 = min([-x_all.var.MuH(iMuHNeg)./deltaMuH(iMuHNeg)
                  -x_all.var.MuL(iMuLNeg)./deltaMuL(iMuLNeg)
                  -x_all.var.MurhoH(iMurhoHNeg)./deltaMurhoH(iMurhoHNeg)
                  -x_all.var.MurhoL(iMurhoLNeg)./deltaMurhoL(iMurhoLNeg)]);
  alpha_d = min([1, const*alpha_d0]);
  fprintf('\nalpha_d = %8.4g \n\n', alpha_d);

% The minimum of Primal and Dual steps.

  alpha = min(alpha_p, alpha_d);

%-------------------------------------------------------------------------
% Time to update vector Z.

  x_all.var.Z0     = x_all.var.Z(x_all.ind.Z0) + alpha_p*deltaZ(x_all.ind.Z0);
  x_all.var.sH     = x_all.var.sH + alpha_p*deltasH;
  x_all.var.sL     = x_all.var.sL + alpha_p*deltasL;
```

```
  x_all.var.rhoH   = x_all.var.rhoH + alpha_p*deltarhoH;
  x_all.var.rhoL   = x_all.var.rhoL + alpha_p*deltarhoL;
  x_all.var.MuH    = x_all.var.MuH + alpha_d*deltaMuH;
  x_all.var.MuL    = x_all.var.MuL + alpha_d*deltaMuL;
  x_all.var.MurhoH = x_all.var.MurhoH + alpha_d*deltaMurhoH;
  x_all.var.MurhoL = x_all.var.MurhoL + alpha_d*deltaMurhoL;

  x_all.var.Z = zeros(length(deltaZ),1);
  x_all.var.Z(x_all.ind.Z0) = x_all.var.Z0;
  x_all.var.Z(x_all.ind.MuH) = x_all.var.MuH;
  x_all.var.Z(x_all.ind.MuL) = x_all.var.MuL;
  x_all.var.Z(x_all.ind.MurhoH) = x_all.var.MurhoH;
  x_all.var.Z(x_all.ind.MurhoL) = x_all.var.MurhoL;
  x_all.var.Z(x_all.ind.sH) = x_all.var.sH;
  x_all.var.Z(x_all.ind.sL) = x_all.var.sL;
  x_all.var.Z(x_all.ind.rhoH) = x_all.var.rhoH;
  x_all.var.Z(x_all.ind.rhoL) = x_all.var.rhoL;

%--------------------------------------------------------------------------
```

```
function[x_main,x_sub] = UnPackZ(x_all,x_main,x_sub)

%=========================================================================

x_main.var.V = x_all.var.Z(x_main.ind.V);
x_main.var.Angl = zeros(x_main.n.NBus,1);
ind_temp = [1:x_main.n.NBus]';
notSlackbus = find(ind_temp~=x_main.ind.Slackbus);
x_main.var.Angl(notSlackbus) = x_all.var.Z(x_main.ind.Angl);
x_main.var.Tr = x_all.var.Z(x_main.ind.Tr);
x_main.var.Phi = x_all.var.Z(x_main.ind.Phi);
x_main.var.PG = x_all.var.Z(x_main.ind.PG);
x_main.var.QG = x_all.var.Z(x_main.ind.QG);
x_main.var.PL = x_all.var.Z(x_main.ind.PL);
x_main.var.Is = x_all.var.Z(x_main.ind.Is);
x_main.var.LambdaP = x_all.var.Z(x_main.ind.LambdaP);
x_main.var.LambdaQ = x_all.var.Z(x_main.ind.LambdaQ);
x_main.var.LambdaI = x_all.var.Z(x_main.ind.LambdaI);

x_main.var.MuH = x_all.var.Z(x_main.n.NZ0+1 : x_main.n.NZ0+x_main.n.NaH);
x_main.var.MuL = x_all.var.Z(x_main.n.NZ0+x_main.n.NaH+1 :  ...
                             x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL);
x_main.var.MurhoH = x_all.var.Z(x_main.n.NZ0+x_main.n.NaH+x_main.n.NaL+1 :...
                    x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL);
x_main.var.MurhoL =x_all.var.Z(x_main.n.NZ0+2*x_main.n.NaH+x_main.n.NaL+1: ...
                   x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL));
x_main.var.sH = x_all.var.Z(x_main.n.NZ0+2*(x_main.n.NaH+x_main.n.NaL)+1: ...
                x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL);
x_main.var.sL = x_all.var.Z(x_main.n.NZ0+3*x_main.n.NaH+2*x_main.n.NaL+1 : ...
                x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL));
x_main.var.rhoH =x_all.var.Z(x_main.n.NZ0+3*(x_main.n.NaH+x_main.n.NaL)+1: ...
                 x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL);
x_main.var.rhoL =x_all.var.Z(x_main.n.NZ0+4*x_main.n.NaH+3*x_main.n.NaL+1: ...
                 x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL));

x_main.var.Z = [ x_main.var.V
                 x_main.var.Angl
                 x_main.var.Tr
                 x_main.var.Phi
                 x_main.var.PG
                 x_main.var.QG
                 x_main.var.PL
                 x_main.var.Is
                 x_main.var.LambdaP
                 x_main.var.LambdaQ
                 x_main.var.LambdaI
                 x_main.var.MuH
                 x_main.var.MuL
                 x_main.var.MurhoH
                 x_main.var.MurhoL
                 x_main.var.sH
                 x_main.var.sL
                 x_main.var.rhoH
                 x_main.var.rhoL     ];

x_main.var.Z(x_main.ind.Slack,:) = [];  % Take out angle of slack bus.

next = x_main.n.NZ0+4*(x_main.n.NaH+x_main.n.NaL);

x_main.Iter = x_main.Iter+1;

for k = 1:x_main.n.K,
    x_sub(k).var.V = x_all.var.Z(x_sub(k).ind.V+next);
    x_sub(k).var.Angl = zeros(x_sub(k).n.NBus,1);
    ind_temp = [1:x_sub(k).n.NBus]';
    notSlackbus = find(ind_temp~=x_sub(k).ind.Slackbus);
    x_sub(k).var.Angl(notSlackbus) = x_all.var.Z(x_sub(k).ind.Angl+next);
    x_sub(k).var.Tr = x_all.var.Z(x_sub(k).ind.Tr+next);
```

```
      x_sub(k).var.Phi = x_all.var.Z(x_sub(k).ind.Phi+next);
      x_sub(k).var.PG = x_all.var.Z(x_sub(k).ind.PG+next);
      x_sub(k).var.QG = x_all.var.Z(x_sub(k).ind.QG+next);
      x_sub(k).var.PL = x_all.var.Z(x_sub(k).ind.PL+next);
      x_sub(k).var.Is = x_all.var.Z(x_sub(k).ind.Is+next);
      x_sub(k).var.LambdaP = x_all.var.Z(x_sub(k).ind.LambdaP+next);
      x_sub(k).var.LambdaQ = x_all.var.Z(x_sub(k).ind.LambdaQ+next);
      x_sub(k).var.LambdaI = x_all.var.Z(x_sub(k).ind.LambdaI+next);

      x_sub(k).var.MuH = x_all.var.Z(x_sub(k).ind.MuH+next);
      x_sub(k).var.MuL = x_all.var.Z(x_sub(k).ind.MuL+next);
      x_sub(k).var.MurhoH = x_all.var.Z(x_sub(k).ind.MurhoH+next);
      x_sub(k).var.MurhoL = x_all.var.Z(x_sub(k).ind.MurhoL+next);
      x_sub(k).var.sH = x_all.var.Z(x_sub(k).ind.sH+next);
      x_sub(k).var.sL = x_all.var.Z(x_sub(k).ind.sL+next);
      x_sub(k).var.rhoH = x_all.var.Z(x_sub(k).ind.rhoH+next);
      x_sub(k).var.rhoL = x_all.var.Z(x_sub(k).ind.rhoL+next);

      x_sub(k).var.Z = [ x_sub(k).var.V
                         x_sub(k).var.Angl
                         x_sub(k).var.Tr
                         x_sub(k).var.Phi
                         x_sub(k).var.PG
                         x_sub(k).var.QG
                         x_sub(k).var.PL
                         x_sub(k).var.Is
                         x_sub(k).var.LambdaP
                         x_sub(k).var.LambdaQ
                         x_sub(k).var.LambdaI
                         x_sub(k).var.MuH
                         x_sub(k).var.MuL
                         x_sub(k).var.MurhoH
                         x_sub(k).var.MurhoL
                         x_sub(k).var.sH
                         x_sub(k).var.sL
                         x_sub(k).var.rhoH
                         x_sub(k).var.rhoL     ];

      x_sub(k).var.Z(x_sub(k).ind.Slack,:) = [];  % Take out angle of slack bus.

      next = next + x_sub(k).n.NZO+4*(x_sub(k).n.NaH+x_sub(k).n.NaL);

      x_sub(k).Iter = x_sub(k).Iter+1;

   end %for k

%=============================================================================
```

```
  function[] = Print(x,k,look_g,norm_g)

% function[] = Print(x,k,look_g,norm_g)
% This function Displays the variables during each iteration.

%=============================================================================
% Prepare variables for printing out.

  PG = zeros(1,x.n.NBus);
  PG(x.ind.PFree) = x.var.PG;
  PL = zeros(1,x.n.NBus);
  PL(x.ind.Loadbus) = x.var.PL;

  if k==0
     LambdaP_print = x.var.LambdaP/x.m.Pi0;
     LambdaQ_print = x.var.LambdaQ/x.m.Pi0;
     LambdaI_print = x.var.LambdaI/x.m.Pi0;
     MuH_print = x.var.MuH/x.m.Pi0;
     MuL_print = x.var.MuL/x.m.Pi0;
     MC_gen = zeros(1,x.n.NBus);
     MC_gen(x.ind.PFree) = x.cost.MC;
     MB_load = zeros(1,x.n.NBus);
     MB_load(x.ind.Loadbus) = x.cost.MB;
     Cost = x.cost.Obj/x.m.Pi0;

  else
     Pi_k = x.m.Pi(k);
     LambdaP_print = x.var.LambdaP/Pi_k;
     LambdaQ_print = x.var.LambdaQ/Pi_k;
     LambdaI_print = x.var.LambdaI/Pi_k;
     MuH_print = x.var.MuH/Pi_k;
     MuL_print = x.var.MuL/Pi_k;
     MC_gen = zeros(1,x.n.NBus);
     MC_gen(x.ind.PFree) = x.cost.MC;
     MB_load = zeros(1,x.n.NBus);
     MB_load(x.ind.Loadbus) = x.cost.MB;
     Cost = x.cost.Obj/Pi_k;
  end %if

  PG_total = sum(x.var.PG);
  PL_total = sum(x.var.PL);
  Loss = PG_total - PL_total;

%=============================================================================
  disp('============================Variables================================')
  disp(sprintf('Calculated values at the end of iteration %4i are:',x.Iter-1));
  disp([ sprintf('Bus number ')  sprintf(' %6d  ',[1:x.n.NBus])]);
  disp([ sprintf('V          =')  sprintf(' %8.4f',x.var.V)]);
  disp([ sprintf('Angl(rad) =')  sprintf(' %8.4f',x.var.Angl)]);
  disp([ sprintf('Angl(deg) =')  sprintf(' %8.4f',x.var.Angl*180/pi)]);
  disp([ sprintf('PG         =')  sprintf(' %8.4f',x.print.P)]);
  disp([ sprintf('QG         =')  sprintf(' %8.4f',x.print.Q)]);
  disp([ sprintf('PL         =')  sprintf(' %8.4f',PL)]);
  disp([ sprintf('PI         =')  sprintf(' %8.4f',x.var.PI)]);
  disp([ sprintf('LambdaP    =')  sprintf(' %8.4f',full(LambdaP_print))]);
  disp([ sprintf('LambdaQ    =')  sprintf(' %8.4f',full(LambdaQ_print))]);
  disp([ sprintf('MC_gen     =')  sprintf(' %8.4f',MC_gen)]);
  disp([ sprintf('MB_load    =')  sprintf(' %8.4f',MB_load)]);
  fprintf(1,'%3i. Is(%2i,%2i) = %9.4f   LambdaI = %9.4f \n',...
          [(1:x.n.NI)' x.ind.f x.ind.t x.var.Is LambdaI_print]');
  disp([ sprintf('PG_total  =')  sprintf(' %12.4f pu.',PG_total)]);
  disp([ sprintf('PL_total  =')  sprintf(' %12.4f pu.',PL_total)]);
  disp([ sprintf('Loss       =')  sprintf(' %12.4f pu.',Loss)]);
  disp([ sprintf('Cost       =')  sprintf(' %12.4f $/hour',Cost)]);
  disp('=============================================================================')
%=============================================================================
% Print Binding constraints in FP.
```

```
if ~x.flag.ip
  H_type = x.print.Type(x.ind.aH);
  L_type = x.print.Type(x.ind.aL);
  H_name = x.print.Name(x.ind.aH,:);
  L_name = x.print.Name(x.ind.aL,:);
  H_num  = x.print.Number(x.ind.aH,:);
  L_num  = x.print.Number(x.ind.aL,:);

  if ~isempty(H_type)
    IsH_bind = find(H_type==3);       % Look for Is binding at upper limit.
    if ~isempty(IsH_bind)
       IsH_num = H_num(IsH_bind);     % Look up for the num of IsH.
       IsH_f = x.main.ind.f(IsH_num); % This Is is from line ....
       IsH_t = x.main.ind.t(IsH_num); % ... to line ....
    end %if
  else
     IsH_bind = [];
  end %if
  if ~isempty(L_type)
    IsL_bind = find(L_type==3);       % Look for Is binding at lower limit.
    if ~isempty(IsL_bind)
       IsL_num = L_num(IsL_bind);     % Look up for the num of IsL.
       IsL_f = x.main.ind.f(IsL_num); % This Is is from line ....
       IsL_t = x.main.ind.t(IsL_num); % ... to line ....
    end %if
  else
     IsL_bind = [];
  end %if

  disp('=================Binding Constratins in FP==========================')
  disp('Binding constraints on upper limits are: ')
  disp([ sprintf('Upper Limits Number:') sprintf('%9i', x.ind.aH)]);
  disp([ sprintf('Upper Limits       :     ') ...
         sprintf('%s',[H_name num2str(H_num,'%-5i')]')]);
  disp([ sprintf('Mu of Upper Limits =') sprintf('%9.4f', full(MuH_print))]);
  if ~isempty(PsH_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsH_num PsH_f PsH_t]')
  end %if
  disp('--------------------------------------------------------------------')
  disp('Binding constraints on lower limits are: ')
  disp([ sprintf('Lower Limits Number:') sprintf('%9i', x.ind.aL)]);
  disp([ sprintf('Lower Limits       :     ') ...
         sprintf('%s',[L_name num2str(L_num,'%-5i')]')]);
  disp([ sprintf('Mu of Lower Limits =') sprintf('%9.4f', full(MuL_print))]);
  if ~isempty(PsL_bind)
     fprintf(1,'Ps(%2i) is flow on line (%2i,%2i).\n',[PsL_num PsL_f PsL_t]')
  end %if
  disp('====================================================================')

end %if ~x.flag.ip

%=============================================================================
% Print information in IP.

if x.flag.ip
  fprintf('Nu = %8.4g\n',x.var.Nu);

  if x.var.Nu <= x.init.printIP

  disp('=================Conditions in IP==================================')

  ind_aH_bind = find(x.var.MuH > 1e-4);
  ind_aL_bind = find(x.var.MuL > 1e-4);
  aH_bind = x.ind.aH(ind_aH_bind);
  aL_bind = x.ind.aL(ind_aL_bind);

  H_type = x.print.Type(aH_bind,:);
  L_type = x.print.Type(aL_bind,:);
```

```
  H_name = x.print.Name(aH_bind,:);
  L_name = x.print.Name(aL_bind,:);
  H_num  = x.print.Number(aH_bind,:);
  L_num  = x.print.Number(aL_bind,:);

  if ~isempty(H_type)
    IsH_bind = find(H_type==8);       % Look for Is binding at upper limit.
    if ~isempty(IsH_bind)
       IsH_num = H_num(IsH_bind);     % Look up for the num of IsH.
       IsH_f = x.init.ind.f(IsH_num); % This Is is from line ....
       IsH_t = x.init.ind.t(IsH_num); % ... to line ....
    end %if
  else
     IsH_bind = [];
  end %if
  if ~isempty(L_type)
    IsL_bind = find(L_type==8);       % Look for Is binding at lower limit.
    if ~isempty(IsL_bind)
       IsL_num = L_num(IsL_bind);     % Look up for the num of IsL.
       IsL_f = x.init.ind.f(IsL_num); % This Is is from line ....
       IsL_t = x.init.ind.t(IsL_num); % ... to line ....
    end %if
  else
     IsL_bind = [];
  end %if

if ~isempty(aH_bind)   % Print this portion if there is any nonzero MuH.
  disp('Binding constraints on upper limits are: ')
  disp([ sprintf('Upper Limits Number:')  sprintf('%9i',aH_bind)]);
  disp([ sprintf('Upper Limits       :     ')  ...
         sprintf('%s',[H_name num2str(H_num,'%-5i')]')]);
  disp([ sprintf('Mu of Upper Limits    =')  ...
         sprintf('%9.4f',MuH_print(ind_aH_bind))]);
  disp([ sprintf('Slack of Upper Limits =')  ...
         sprintf('%9.4f',x.var.sH(ind_aH_bind))]);
  if ~isempty(IsH_bind)
     fprintf(1,'Is(%2i) is flow on line (%2i,%2i).\n',[IsH_num IsH_f IsH_t]')
  end %if
end %if ~isempty(aH_bind)
if ~isempty(aL_bind)   % Print this portion if there is any nonzero MuL.
  disp('---------------------------------------------------------------------')
  disp('Binding constraints on lower limits are: ')
  disp([ sprintf('Lower Limits Number:')  sprintf('%9i',aL_bind)]);
  disp([ sprintf('Lower Limits       :     ')  ...
         sprintf('%s',[L_name num2str(L_num,'%-5i')]')]);
  disp([ sprintf('Mu of Lower Limits    =')  ...
         sprintf('%9.4f',MuL_print(ind_aL_bind))]);
  disp([ sprintf('Slack of Lower Limits =')  ...
         sprintf('%9.4f',x.var.sL(ind_aL_bind))]);
  if ~isempty(IsL_bind)
     fprintf(1,'Is(%2i) is flow on line (%2i,%2i).\n',[IsL_num IsL_f IsL_t]')
  end %if
end %if ~isempty(aL_bind)

  if x.flag.ip == 2    % for IP with Rho.
     ind_rhoH_bind = find(x.var.rhoH > 1e-4);
     ind_rhoL_bind = find(x.var.rhoL > 1e-4);
     rhoH_bind = x.ind.aH(ind_rhoH_bind);
     rhoL_bind = x.ind.aL(ind_rhoL_bind);

     Hrho_name = x.print.Name(rhoH_bind,:);
     Lrho_name = x.print.Name(rhoL_bind,:);
     Hrho_num  = x.print.Number(rhoH_bind,:);
     Lrho_num  = x.print.Number(rhoL_bind,:);

if ~isempty(rhoH_bind)   % Print this portion if there is any nonzero MurhoH.
    disp('---------------------------------------------------------------------')
    disp('Exceeded constraints on upper limits are: ')
```

```
    disp([ sprintf('Exceeded Upper Limits Number:')  ...
           sprintf('%9i',ind_rhoH_bind)]);
    disp([ sprintf('Exceeded Upper Limits:            ') ...
           sprintf('%s',[Hrho_name num2str(Hrho_num,'%-5i')]')]);
    disp([ sprintf('Mu of Rho on Upper Limits =')  ...
           sprintf('%9.4f',x.var.MurhoH(ind_rhoH_bind))]);
    disp([ sprintf('Rho on Upper Limits       =')  ...
           sprintf('%9.4f',x.var.rhoH(ind_rhoH_bind))]);
end %if ~isempty(rhoH_bind)
if ~isempty(rhoL_bind)   % Print this portion if there is any nonzero MurhoL.
    disp('---------------------------------------------------------------------')
    disp('Exceeded constraints on lower limits are: ')
    disp([ sprintf('Exceeded Lower Limits Number:')  ...
           sprintf('%9i',ind_rhoL_bind)]);
    disp([ sprintf('Exceeded Lower Limits:            ')  ...
           sprintf('%s',[Lrho_name num2str(Lrho_num,'%-5i')]')]);
    disp([ sprintf('Mu of Rho on Lower Limits =')  ...
           sprintf('%9.4f',x.var.MurhoL(ind_rhoL_bind))]);
    disp([ sprintf('Rho on Lower Limits       =')  ...
           sprintf('%9.4f',x.var.rhoL(ind_rhoL_bind))]);
  end %if x.flag.ip ==2
end %if ~isempty(rhoL_bind)
  end %if x.var.Nu
  disp('=====================================================================')
end %if x.flag.ip

%=============================================================================
% Print out pieces of g

if look_g

   if input(' Look at g_var? YES=1; NO=0; ');
      fprintf('                        |g_var| = %8.4g\n', norm_g.g_var);
   end

   if input(' Look at g_lambda? YES=1; NO=0; ');
     fprintf('                       |g_lambda| = %8.4g\n', norm_g.g_lambda);
   end

   if input(' Look at g_Crho? YES=1; NO=0; ');
      fprintf('                        |g_Crho| = %8.4g\n', norm_g.g_Crho);
   end

   if input(' Look at g_slack? YES=1; NO=0; ');
      fprintf('                        |g_slack| = %8.4g\n', norm_g.g_slack);
   end

   if input(' Look at g_comp? YES=1; NO=0; ');
      fprintf('                        |g_comp| = %8.4g\n', norm_g.g_comp);
   end

disp('=====================================================================')
end %if look_g

%=============================================================================
```

APPENDIX B

INPUT DATA FILES

The input data files for the ESCOPF program are in the IEEE Common Data Format (IEEE CDF). This appendix only describes the data that has been used in the ESCOPF program. The full detail of the format is described in [75]. The bus data and line data are shown in Figures B.1, B.4, and B.7. The generator data is shown in Figures B.2, B.5, and B.8. The load data is shown in Figures B.3, B.6, and B.9.

B.3 Bus Data

The followings are the descriptions of the bus data used in the ESCOPF program. Each row of the data format represents the information of one bus in the particular system.

Columns 1-4 **Bus number:** a four digit integer from 1 to 9999

26 **Bus type:** indicated by a single digit integer from 0 to 3, where

0 = Unregulated bus (Load bus)

1 = Hold MVar generation within voltage limits

2 = Hold bus voltage within generator MVar limits

3 = Hold bus voltage and angle Swing bus

Columns 28-33 **Bus voltage:** from solved load flow to be used as initial guess

34-40 **Bus angle:** from solved load flow to be used as initial guess

41-49 **Load MW:** real load resulted from solved load flow

50-58 **Load MVar:** reactive load resulted from solved load flow

59-67 **Gen MW:** real power generation from solved load flow to be used as initial guess

68-75 **Gen MVar:** reactive power generation from solved load flow to be used as initial guess

91-98 **Maximum voltage:** for only bus type 0 (load bus)

99-106 **Minimum voltage:** for only bus type 0 (load bus)

107-114 **Bus capacitor**

115-122 **Bus reactor**

B.4 Line Data

The followings are the descriptions of the transmission line data where each row of the data format represents the information of that particular line.

Columns 1-4 **From bus:** a four digit integer identifying the bus number the line starts

6-9 **To bus:** a four digit integer identifying the bus number the line ends

Columns 19 **Line type:** indicated by a single digit integer from 1 to 4, where

1 = Fixed voltage ratio and/or fixed phase shifter

2 = Fixed phase angle and variable voltage ratio with voltage control

2 = Fixed phase angle and variable voltage ratio with MVar control

4 = Fixed voltage ratio and variable phase shifter with MW control

20-29 **Line resistance (R)**

30-39 **Line reactance (X):** line impedance is computed from $R + jX$

41-49 **Line charging capacitance**

57-61 **Maximum line flow (MVA):** for emergency state

57-61 **Maximum line flow (MVA):** for normal operating state

77-82 **Transformer tap ratio**

84-90 **Phase angle**

91-97 **Minimum limit of variable tap ratio or phase shifter**

98-104 **Maximum limit of variable tap ratio or phase shifter**

134-139 **Probability of contingency[1]:** each transmission line has an outage probability in percent (%) from 0 to 100

This information is added to the CDF file for ESCOPF program.

B.5 Generator Data

The generator data file (GDF) contains the generator data as follows:

Columns 1-5	**Generator bus number**
7-15	**Generator name**
17-23	**Maximum MW:** maximum limit on real power generation
25-29	**Minimum MW:** minimum limit on real power generation
31-38	**Maximum MVar:** maximum limit on reactive power generation
40-45	**Minimum MVar:** minimum limit on reactive power generation
47	**Adjustable real power generation:** indicated by a single digit 0 or 1, where 0 = real power generation is not adjustable 1 = real power generation is adjustable
49	**Adjustable reactive power generation:** indicated by a single digit 0 or 1, where 0 = reactive power generation is not adjustable 1 = reactive power generation is adjustable
51	**Availability:** 0 = the generator is not available 1 = the generator is available
53-75	**Generator cost curve coefficients:** the generator cost curve is written in the form $\alpha + \beta P_{Gi} + \gamma P_{Gi}^2$
53-59	α
61-66	β
68-75	γ

Columns 77-81 **Maximum voltage:** for generator bus

83-87 **Minimum voltage:** for generator bus

89-93 **Maximum MW ramping up**

95-99 **Maximum MW ramping down**

101-106 **Maximum MVar ramping up**

108-113 **Maximum MVar ramping down**

115-129 **Ramping cost curve coefficients:** the ramping cost is written in the form $\beta_R P_{Ri} + \gamma_R P_{Ri}^2$

115-120 β_R

122-129 γ_R

B.6 Load Data

The load data file (LDF) contains information on the load bus.

Columns 1-5 **Load bus number**

7-13 **Load bus name**

15-18 **Load type:** there are three types of customers

1 = residential

2 = industrial

3 = commercial

20-25 **Maximum MW load**

27-32 **Minimum MW load**

Columns 34-40 **Maximum MVar load**

42-48 **Minimum MVar load**

50 **Interruptible load:**

0 = not interruptible

1 = interruptible

52-59 **Maximum MW interruptible:** the maximum amount of load that can be interrupted

61-87 **Customer benefit curve coefficients:** is written in the form

$\alpha_L + \beta_L P_{Li} + \gamma_L P_{Li}^2$

61-68 α_L

70-77 β_L

79-87 γ_L

89-113 **Interruption cost curve coefficients:** is written in the form

$\alpha_I + \beta_I P_{Ii} + \gamma_I P_{Ii}^2$

89-96 α_I

98-104 β_I

106-113 γ_I

115-120 **Load power factor**

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

 08/15/95 OSU DATA              1.000  OSU mods to UC Berkeley 3 bus example
BUS DATA FOLLOWS                              3  ITEMS
   1 Bus 1     HV  1  1  3 1.060    0.0      0.0      0.0     10.0    -5.0     0.0 1.060    150    -100    0.0     0.0        0
   2 Bus 2     HV  1  1  2 1.045    0.0      0.0      0.0      8.0     2.0     0.0 1.045   50.0   -40.0    0.0     0.0        0
   3 Bus 3     HV  1  1  0 1.010    0.0      1.0      1.0      0.0     0.0     0.0 1.010   1.08   0.000    0.0     0.0        0
-999
BRANCH DATA FOLLOWS                           3  ITEMS
   1    2  1  1 1 0  0.00000    1.0000     0.0000     8     8     8    0 0  0.0       0.0 0.0    0.0     0.0     0.0     0.0      0   1  3.1
   1    3  1  1 1 0  0.00000    2.0000     0.0000    30    30    30    0 0  0.0       0.0 0.0    0.0     0.0     0.0     0.0      0   2  0.0
   2    3  1  1 1 0  0.00000    0.3333     0.0000    20    20    20    0 0  0.0       0.0 0.0    0.0     0.0     0.0     0.0      0   3  0.0
-999
LOSS ZONES FOLLOWS                    1 ITEMS
  1 IEEE 14 BUS
-99
END OF DATA
```

Figure B.1: Data for 3-bus case in IEEE CDF format.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
BUNUM GENERNAME MX---MW MN-MW MX---MVR MN-MVR P Q A AL__PHA B-E-TA GAM-M--A V-MAX V-MIN MW-UP MW-DN MVARUP MVARDN BETA_R GAMMAR
    1 GEN1        35.00   5.0      inf  -inf  1 1 1    0.0     0.0    1.0000  1.08 0.950  5.00  8.00    inf     inf     0.0     0.0
    2 GEN2        30.00   5.0      inf  -inf  1 1 1    0.0     0.0    1.6750  1.08 0.950  1.50  7.00    inf     inf     0.0     0.0
```

Figure B.2: Generator data for DC 3-bus case in IEEE CDF format.


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
BUNUM BUSNAME TYPE MX--MW MN--MW MX-MVAR MN-MVAR R MX-MW-IN ALPHA-LD BETA--LD GAMMA--LD ALPHA-IN BETA-IN GAMMA-IN P----F
    3 Indust1    2     50    0.0     inf    -inf 1      100      0.00    33.00    0.0000       0.0   100.0       0.0  0.98
```

Figure B.3: Load data for DC 3-bus case in IEEE CDF format.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

 09/25/93 OSU DATA              100.0  OSU mods to IEEE 14
BUS DATA FOLLOWS                                5  ITEMS
   1 Bus 1     HV  1  1  3 1.060     0.0      0.0      0.0    232.4   -16.9      0.0  1.060    150    -100     0.0     0.0         0
   2 Bus 2     HV  1  1  2 1.045   -4.98     21.7     12.7     40.0    42.4      0.0  1.045     50.0   -40.0     0.0     0.0         0
   3 Bus 3     HV  1  1  2 1.010  -12.72     94.2     19.0      0.0    23.4      0.0  1.010     40.0     0.0     0.0     0.0         0
   4 Bus 4     HV  1  1  0 1.019  -10.33     91.98   -13.64     0.0     0.0      0.0  0.0       1.05     0.95    0.0     0.0         0
   5 Bus 5     HV  1  1  0 1.020   -8.78     51.66    14.42     0.0     0.0      0.0  0.0       1.05     0.95    0.0     0.0         0
-999
BRANCH DATA FOLLOWS                             7  ITEMS
   1    2  1  1 1 0  0.01938    0.05917     0.0528   155   155   120    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  1   1.00
   1    5  1  1 1 0  0.05403    0.22304     0.0492   110   110    90    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  2   1.00
   2    3  1  1 1 0  0.04699    0.19797     0.0438    95    95    75    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  3   1.00
   2    4  1  1 1 0  0.05811    0.17632     0.0374    95    95    75    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  4   1.00
   2    5  1  1 1 0  0.05695    0.17388     0.0340    95    95    75    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  5   1.00
   3    4  1  1 1 0  0.06701    0.17103     0.0346    95    95    75    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  6   1.00
   4    5  1  1 1 0  0.01335    0.04211     0.0128    95    95    75    0 0  0.0       0.0 0.0    0.0      0.0     0.0     0.0      0  7   1.00
-999
LOSS ZONES FOLLOWS                      1 ITEMS
  1 IEEE 14 BUS
-99
INTERCHANGE DATA FOLLOWS                    1 ITEMS
 1    2 Bus 2     HV    0.0  999.99  IEEE14  IEEE 14 Bus Test Case
-9
TIE LINES FOLLOWS                       0 ITEMS
-999
END OF DATA
```

Figure B.4: Data for 5-bus case in IEEE CDF format.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
BUNUM GENERNAME MX---MW MN-MW MX---MVR MN-MVR P Q A AL__PHA B-E-TA GAM-M--A V-MAX V-MIN MW-UP MW-DN MVARUP MVARDN BETA_R GAMMAR
    1 GEN1       250.00  45.0   150.00  -100  1 1 1    0.0   11.0   .000820  1.08 0.950  50.0  50.0    inf     inf     0.0 .00000
    2 GEN2       150.00  15.0    50.00   -40  1 1 1    0.0   12.0   .000776  1.08 0.950  35.0  35.0    inf     inf     0.0 .00000
    3 SynCon 1     0.00   0.0    40.00   -40  0 1 1    0.0    0.0   .000000  1.08 0.950   inf   inf    inf     inf     0.0 .00000
```

Figure B.5: Generator data for 5-bus case in IEEE CDF format.


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
BUNUM BUSNAME TYPE MX--MW MN--MW MX-MVAR MN-MVAR R MX-MW-IN ALPHA-LD BETA--LD GAMMA--LD ALPHA-IN BETA-IN GAMMA-IN P----F
    2 Residen    1    inf    0.0     inf    -inf 1      100      0.00    21.43   -0.2147      0.0   100.0      0.0  0.98
    3 Indust1    2    inf    0.0     inf    -inf 1      100      0.00    23.00   -0.0531      0.0   100.0      0.0  0.98
    4 Indust2    2    inf    0.0     inf    -inf 1      100      0.00    22.48   -0.0531      0.0   100.0      0.0  0.98
    5 Commerc    3    inf    0.0     inf    -inf 1      100      0.00    22.11   -0.0930      0.0   100.0      0.0  0.98
```

Figure B.6: Load data for 5-bus case in IEEE CDF format.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

 07/20/94 OSU mods to UW case  100.0  1962 W IEEE14 W/LINE LIMITS, BUS KV
BUS DATA FOLLOWS                            14 ITEMS
   1 Bus 1     HV  1  1  3 1.0600    .00      .00      .00   232.39  -16.90  230.00 1.0600  150.00 -100.00   .0000   .0000     0    1
   2 Bus 2     HV  1  1  2 1.0450  -4.98    21.70    12.70    40.00   42.41  230.00 1.0450   50.00  -40.00   .0000   .0000     0    2
   3 Bus 3     HV  1  1  2 1.0100 -12.72    94.20    19.00      .00   23.39  230.00 1.0100   40.00     .00   .0000   .0000     0    3
   4 Bus 4     HV  1  1  0 1.0186 -10.32    47.80    -3.90      .00     .00  230.00  .0000    1.05    0.950   .0000   .0000     0    4
   5 Bus 5     HV  1  1  0 1.0203  -8.78     7.60     1.60      .00     .00  230.00  .0000    1.05    0.950   .0000   .0000     0    5
   6 Bus 6     LV  1  1  2 1.0700 -14.22    11.20     7.50      .00   12.24   69.00 1.0700   24.00   -6.00   .0000   .0000     0    6
   7 Bus 7     ZV  1  1  0 1.0620 -13.37      .00      .00      .00     .00  161.00  .0000    1.250   0.750   .0000   .0000     0    7
   8 Bus 8     TV  1  1  2 1.0900 -13.37      .00      .00      .00   17.36  138.00 1.0900   24.00   -6.00   .0000   .0000     0    8
   9 Bus 9     LV  1  1  0 1.0563 -14.95    29.50    16.60      .00     .00   69.00  .0000    1.050   0.950   .0000   .1900     0    9
  10 Bus 10    LV  1  1  0 1.0513 -15.10     9.00     5.80      .00     .00   69.00  .0000    1.050   0.950   .0000   .0000     0   10
  11 Bus 11    LV  1  1  0 1.0571 -14.80     3.50     1.80      .00     .00   69.00  .0000    1.050   0.900   .0000   .0000     0   11
  12 Bus 12    LV  1  1  0 1.0552 -15.08     6.10     1.60      .00     .00   69.00  .0000    1.050   0.950   .0000   .0000     0   12
  13 Bus 13    LV  1  1  0 1.0504 -15.16    13.50     5.80      .00     .00   69.00  .0000    1.050   0.950   .0000   .0000     0   13
  14 Bus 14    LV  1  1  0 1.0358 -16.04    14.90     5.00      .00     .00   69.00  .0000    1.050   0.950   .0000   .0000     0   14
-999
BRANCH DATA FOLLOWS                         20 ITEMS
   1    2  1  1 1 0   .019380   .059170   .05280  120.  155.  120.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  1  1.00
   1    5  1  1 1 0   .054030   .223040   .04920   90.  110.   90.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  2  1.00
   2    3  1  1 1 0   .046990   .197970   .04380   75.   95.   75.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  3  1.00
   2    4  1  1 1 0   .058110   .176320   .03740   75.   95.   75.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  4  1.00
   2    5  1  1 1 0   .056950   .173880   .03400   50.   95.   75.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  5  1.00
   3    4  1  1 1 0   .067010   .171030   .03460   50.   95.   75.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  6  1.00
   4    5  1  1 1 0   .013350   .042110   .01280   90.  110.   90.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  7  1.00
   6   11  1  1 1 0   .094980   .198900   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  8  0.00
   6   12  1  1 1 0   .122910   .255810   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0  9  0.00
   6   13  1  1 1 0   .066150   .130270   .00000   20.   30.   20.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 10  0.00
   9   10  1  1 1 0   .031810   .084500   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 11  0.00
   9   14  1  1 1 0   .127110   .270380   .00000   15.   25.   15.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 12  0.00
  10   11  1  1 1 0   .082050   .192070   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 13  0.00
  12   13  1  1 1 0   .220920   .199880   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 14  0.00
  13   14  1  1 1 0   .170930   .348020   .00000   10.   20.   10.   0 0   .0000     .00  .0000  .0000 .00000   .0000  .0000  0 15  0.00
   4    7  1  1 1 1   .000000   .209120   .00000   35.   45.   35.   0 0   .9780     .00  .0000  .0000 .00000   .0000  .0000  0 16  0.00
   4    9  1  1 1 1   .000000   .556180   .00000   20.   30.   20.   0 0   .9690     .00  .0000  .0000 .00000   .0000  .0000  0 17  0.00
   5    6  1  1 1 1   .000000   .252020   .00000   50.   60.   50.   0 0   .9320     .00  .0000  .0000 .00000   .0000  .0000  0 18  0.00
   7    8  1  1 1 1   .000000   .176150   .00000   20.   30.   20.   0 0  1.0000     .00  .0000  .0000 .00000   .0000  .0000  0 19  0.00
   7    9  1  1 1 1   .000000   .110010   .00000   35.   45.   35.   0 0  1.0000     .00  .0000  .0000 .00000   .0000  .0000  0 20  0.00
-999
```

Figure B.7: Data for IEEE 14-bus case in IEEE CDF format.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
BUNUM GENERNAME MX--GEN MNGEN MX---MVR MN-MVR P Q A AL--PHA B-E-TA GAM---MA V-MAX V-MIN MW-UP MW-DN MVARUP MVARDN BETA-R GAMMA__R
    1 GEN 1      250.00  45.0   150.00  -100  1 1 1 692.32  11.0   .000820  1.080 0.950    50    50    inf    inf     0.0 .00000
    2 GEN 2      150.00  15.0    50.00   -40  1 1 1 692.32  12.0   .000776  1.080 0.950    35    35    inf    inf     0.0 .00000
    3 SynCon 1     0.00  0.00    40.00   -40  0 1 1   0.00   0.0   .000000  1.080 0.950   inf   inf    inf    inf     0.0 .00000
    6 SynCon 2     0.00  0.00   104.00  -100  0 1 1   0.00   0.0   .000000  1.080 0.950   inf   inf    inf    inf     0.0 .00000
    8 SynCon 3     0.00  0.00   104.00  -100  0 1 1   0.00   0.0   .000000  1.080 0.950   inf   inf    inf    inf     0.0 .00000
```

Figure B.8: Generator data for IEEE 14-bus case in IEEE CDF format.


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
BUNUM BUSNAME TYPE MX--MW MN--MW MX-MVAR MN-MVAR R MX-MW-IN ALPHA-LD BETA--LD GAMMA--LD ALPHA-IN BETA-IN GAMMA-IN P----F
    2 Residen    1    inf     0.0     inf    -inf 1      100     0.00    21.43    -0.2146      0.0   100.0      0.0  0.98
    3 Indust1    2    inf     0.0     inf    -inf 1      100     0.00    22.98    -0.0530      0.0   100.0      0.0  0.98
    4 Indust2    2    inf     0.0     inf    -inf 1      100     0.00    22.45    -0.1021      0.0   100.0      0.0  0.98
    5 Commerc    3    inf     0.0     inf    -inf 1      100     0.00    22.07    -0.6314      0.0   100.0      0.0  0.98
    6 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.11    -0.4292      0.0   100.0      0.0  0.98
    9 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.41    -0.1652      0.0   100.0      0.0  0.98
   10 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.50    -0.5434      0.0   100.0      0.0  0.98
   11 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.39    -1.3904      0.0   100.0      0.0  0.98
   12 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.51    -0.8022      0.0   100.0      0.0  0.98
   13 Residen    1    inf     0.0     inf    -inf 1      100     0.00    22.63    -0.3644      0.0   100.0      0.0  0.98
   14 Residen    1    inf     0.0     inf    -inf 1      100     0.00    23.02    -0.3359      0.0   100.0      0.0  0.98
```

Figure B.9: Load data for IEEE 14-bus case in IEEE CDF format.

APPENDIX C

RESULTS FROM ESCOPF PROGRAM IN MATLAB

C.7 Standard OPF of DC 3-bus case

```
>> ESCOPF
DC Case name: dc3
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration   27 are:
Bus number       1        2        3
Angl(rad) =   0.0000  -8.0000 -13.9994
Angl(deg) =   0.0000 -458.3662 -802.1068
PG        =  14.9997  10.0000   0.0000
PL        =   0.0000   0.0000  24.9997
PI        =   0.0000   0.0000   0.0000
LambdaP   =  29.9994  33.5000  33.0000
MC_gen    =  29.9994  33.5000   0.0000
MB_load   =   0.0000   0.0000  33.0000
  1. Ps( 1, 2) =  8.00000   LambdaPs =  5.00094
  2. Ps( 1, 3) =  6.99970   LambdaPs =  0.00000
  3. Ps( 2, 3) = 18.00001   LambdaPs =  0.00000
PG_total  =      24.9997 $/hour
PL_total  =      24.9997 $/hour
Loss      =       0.0000 $/hour
Cost      =    -432.4991 $/hour
========================================================================
Nu = 5.005e-16
==================Conditions in IP======================================
Binding constraints on upper limits are:
Upper Limits Number:       4
Upper Limits       :     Ps  1
Mu of Upper Limits    =  5.0009
Slack of Upper Limits =  0.0000
Ps( 1) is flow on line ( 1, 2).
========================================================================
```

```
Expected Security Cost  =    -432.4991 $/hour
```

C.8 ESCOPF of DC 3-bus case

```
>> ESCOPF
DC Case name: dc3
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables================================
Calculated values at the end of iteration   31 are:
Bus number       1        2        3
Angl(rad) =   0.0000  -8.0000 -13.9589
Angl(deg) =   0.0000 -458.3662 -799.7845
PG        =  14.9794   9.8784   0.0000
PL        =   0.0000   0.0000  24.8578
PI        =   0.0000   0.0000   0.0000
LambdaP   =  29.9589  31.0500  30.8941
MC_gen    =  29.9589  33.0927   0.0000
MB_load   =   0.0000   0.0000  33.0000
  1. Ps( 1, 2) =  8.00000   LambdaPs =  1.55874
  2. Ps( 1, 3) =  6.97944   LambdaPs =  0.00000
  3. Ps( 2, 3) = 17.87841   LambdaPs =  0.00000
PG_total  =      24.8578 $/hour
PL_total  =      24.8578 $/hour
Loss      =       0.0000 $/hour
Cost      =    -432.4739 $/hour
=========================================================================
Nu = 5.005e-16
=================Conditions in IP=====================================
Binding constraints on upper limits are:
Upper Limits Number:       4
Upper Limits       :     Ps  1
Mu of Upper Limits    =  1.5587
Slack of Upper Limits =  0.0000
Ps( 1) is flow on line ( 1, 2).
=========================================================================
Print statement for SUBPROBLEM   1 which is line ( 1, 2).
============================Variables================================
Calculated values at the end of iteration   31 are:
Bus number       1        2        3
Angl(rad) =   0.0000 -28.0331 -31.1304
Angl(deg) =   0.0000 -1606.1808 -1783.6396
PG        =  15.5652   9.2927   0.0000
PL        =   0.0000   0.0000  24.8578
PI        =   0.0000   0.0000   0.0000
LambdaP   =  31.1304  31.1304  31.1304
MC_gen    =  31.1304  31.1304   0.0000
MB_load   =   0.0000   0.0000  33.0000
  1. Ps( 1, 3) = 15.56519   LambdaPs =  0.00000
  2. Ps( 2, 3) =  9.29265   LambdaPs =  0.00000
```

```
PG_total  =      24.8578 $/hour
PL_total  =      24.8578 $/hour
Loss      =       0.0000 $/hour
Cost      =    -433.3917 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:       8
Upper Limits       :     PL  3
Mu of Upper Limits    =101.8696
Slack of Upper Limits =  0.0000
================================================================================
Print statement for SUBPROBLEM   2 which is line ( 1, 3).
============================Variables=====================================
Calculated values at the end of iteration   31 are:
Bus number       1        2        3
Angl(rad) =   0.0000  -8.0000 -14.4588
Angl(deg) =   0.0000 -458.3662 -828.4295
PG        =   8.0000  11.3784   0.0000
PL        =   0.0000   0.0000  19.3784
PI        =   0.0000   0.0000   5.4794
LambdaP   =  16.0000 133.0000 133.0000
MC_gen    =  16.0000  38.1177   0.0000
MB_load   =   0.0000   0.0000  33.0000
  1. Ps( 1, 2) =  8.00000   LambdaPs = 117.00000
  2. Ps( 2, 3) = 19.37841   LambdaPs =  0.00000
PG_total  =      19.3784 $/hour
PL_total  =      19.3784 $/hour
Loss      =       0.0000 $/hour
Cost      =     189.3154 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:       4       7
Upper Limits       :     Ps  1   PG  2
Mu of Upper Limits    =117.0000 94.8823
Slack of Upper Limits =  0.0000  0.0000
Ps( 1) is flow on line ( 1, 2).
================================================================================
Print statement for SUBPROBLEM   3 which is line ( 2, 3).
============================Variables=====================================
Calculated values at the end of iteration   31 are:
Bus number       1        2        3
Angl(rad) =   0.0000   8.0000 -49.7157
Angl(deg) =   0.0000 458.3662 -2848.4991
PG        =  16.8578   8.0000   0.0000
PL        =   0.0000   0.0000  24.8578
PI        =   0.0000   0.0000   0.0000
LambdaP   =  33.7157  26.8000  33.7157
MC_gen    =  33.7157  26.8000   0.0000
MB_load   =   0.0000   0.0000  33.0000
  1. Ps( 1, 2) = -8.00000   LambdaPs = -6.91569
  2. Ps( 1, 3) = 24.85784   LambdaPs =  0.00000
PG_total  =      24.8578 $/hour
PL_total  =      24.8578 $/hour
Loss      =       0.0000 $/hour
Cost      =    -428.9219 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:       8
Upper Limits       :     PL  3
Mu of Upper Limits    = 99.2843
Slack of Upper Limits =  0.0000
--------------------------------------------------------------------------------
```

```
Binding constraints on lower limits are:
Lower Limits Number:       4
Lower Limits       :     Ps  1
Mu of Lower Limits    =  6.9157
Slack of Lower Limits =  0.0000
Ps( 1) is flow on line ( 1, 2).
================================================================================
Expected Security Cost  =    -419.9955 $/hour
```

C.9 Standard OPF of DC 5-bus case

```
>> ESCOPF
DC Case name: ieee5
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration   29 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0788  -0.2363  -0.1934  -0.1654
Angl(deg) =   0.0000  -4.5125 -13.5402 -11.0809  -9.4773
PG        =   1.8964   0.7996   0.0000   0.0000   0.0000
PL        =   0.0000   0.2167   0.9668   0.9713   0.5413
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3110  12.1241  12.7322  12.1652  12.0422
MC_gen    =  11.3110  12.1241   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1241  12.7322  12.1652  12.0422
  1. Ps( 1, 2) =  1.20000   LambdaPs =  1.01514
  2. Ps( 2, 3) =  0.75000   LambdaPs =  1.20987
  3. Ps( 2, 4) =  0.58434   LambdaPs =  0.00000
  4. Ps( 3, 4) = -0.21683   LambdaPs = -0.00000
  5. Ps( 1, 5) =  0.69644   LambdaPs =  0.00000
  6. Ps( 2, 5) =  0.44859   LambdaPs =  0.00000
  7. Ps( 4, 5) = -0.60375   LambdaPs = -0.00000
PG_total  =       2.6961 $/hour
PL_total  =       2.6961 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1617.5964 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP==================================
Binding constraints on upper limits are:
Upper Limits Number:       7       8
Upper Limits       :     Ps  1   Ps  2
Mu of Upper Limits    =  1.0151  1.2099
Slack of Upper Limits =  0.0000  0.0000
Ps( 1) is flow on line ( 1, 2).
Ps( 2) is flow on line ( 2, 3).
================================================================================
Expected Security Cost  =   -1617.5964 $/hour
```

C.10 ESCOPF of DC 5-bus case

```
>> ESCOPF
DC Case name: ieee5
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0559  -0.2117  -0.1705  -0.1454
Angl(deg) =   0.0000  -3.2036 -12.1313  -9.7686  -8.3292
PG        =   1.4640   1.1500   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2401  11.2401  11.2401  11.2401  11.2401
MC_gen    =  11.2401  12.1785   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  0.85193   LambdaPs =  0.00000
  2. Ps( 2, 3) =  0.74169   LambdaPs =  0.00000
  3. Ps( 2, 4) =  0.58403   LambdaPs =  0.00000
  4. Ps( 3, 4) = -0.20831   LambdaPs = -0.00000
  5. Ps( 1, 5) =  0.61207   LambdaPs =  0.00000
  6. Ps( 2, 5) =  0.46313   LambdaPs =  0.00000
  7. Ps( 4, 5) = -0.54192   LambdaPs = -0.00000
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1577.0060 $/hour
=========================================================================
Nu = 5.005e-16
==================Conditions in IP=======================================
=========================================================================
Print statement for SUBPROBLEM   1 which is line ( 1, 2).
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.2182  -0.3563  -0.2984  -0.2613
Angl(deg) =   0.0000 -12.4992 -20.4140 -17.0968 -14.9690
PG        =   1.1000   1.5000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2103   0.9500   0.9176   0.5221
PI        =   0.0000   0.0028   0.0000   0.0000   0.0112
LambdaP   =  11.1804 112.3998 112.3998 112.3998 112.3998
MC_gen    =  11.1804  12.2328   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.3998  12.9110  12.7346  12.3998
  1. Ps( 2, 3) =  0.65754   LambdaPs =  0.00000
  2. Ps( 2, 4) =  0.40901   LambdaPs =  0.00000
  3. Ps( 3, 4) = -0.29246   LambdaPs = -0.00000
  4. Ps( 1, 5) =  1.10000   LambdaPs = 101.21936
  5. Ps( 2, 5) =  0.22316   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.80110   LambdaPs = -0.00000
PG_total  =       2.6000 $/hour
PL_total  =       2.6000 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1400.6347 $/hour
=========================================================================
```

```
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:       2      10      14      16      17
Upper Limits       :     PG  2   Ps  5   PG  2   PL  3   PL  4
Mu of Upper Limits    = 10.4173101.2194 89.7497  0.5112  0.3348
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000  0.0000
Ps( 5) is flow on line ( 1, 5).
=====================================================================
Print statement for SUBPROBLEM   2 which is line ( 2, 3).
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0491  -0.4172  -0.2291  -0.1851
Angl(deg) =   0.0000  -2.8124 -23.9026 -13.1274 -10.6042
PG        =   1.5271   1.0869   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2505  12.1687  37.1195  19.2210   7.9275
MC_gen    =  11.2505  12.1687   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  0.74789   LambdaPs =  0.00000
  2. Ps( 2, 4) =  0.91765   LambdaPs =  0.00000
  3. Ps( 3, 4) = -0.95000   LambdaPs = -17.89846
  4. Ps( 1, 5) =  0.77925   LambdaPs =  0.00000
  5. Ps( 2, 5) =  0.70403   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.95000   LambdaPs = -12.95987
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =      -0.0000 $/hour
Cost      =   -1582.8676 $/hour
=====================================================================
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:      15      16      17      18
Upper Limits       :     PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    =100.1117 75.7915 93.5136104.2635
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       9      12
Lower Limits       :     Ps  4   Ps  7
Mu of Lower Limits    = 17.8985 12.9599
Slack of Lower Limits =  0.0000  0.0000
Ps( 4) is flow on line ( 3, 4).
Ps( 7) is flow on line ( 4, 5).
=====================================================================
Print statement for SUBPROBLEM   3 which is line ( 2, 4).
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0349  -0.2277  -0.2213  -0.1773
Angl(deg) =   0.0000  -2.0015 -13.0472 -12.6802 -10.1570
PG        =   1.2786   1.3354   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2097  12.2072  20.4115  28.1423   7.5997
MC_gen    =  11.2097  12.2072   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  0.53226   LambdaPs =  0.00000
  2. Ps( 2, 3) =  0.91765   LambdaPs =  0.00000
  3. Ps( 3, 4) = -0.03235   LambdaPs = -0.00000
  4. Ps( 1, 5) =  0.74639   LambdaPs =  0.00000
  5. Ps( 2, 5) =  0.73689   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.95000   LambdaPs = -22.35289
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
```

```
Loss      =       0.0000 $/hour
Cost      =   -1559.0642 $/hour
=============================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:      15      16      17      18
Upper Limits       :     PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    =100.0732 92.4995 84.5923104.5913
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000
-----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:      12
Lower Limits       :     Ps  7
Mu of Lower Limits    = 22.3529
Slack of Lower Limits =  0.0000
Ps( 7) is flow on line ( 4, 5).
=============================================================================
Print statement for SUBPROBLEM   4 which is line ( 3, 4).
============================Variables=====================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0781  -0.2777  -0.1692  -0.1482
Angl(deg) =   0.0000  -4.4753 -15.9104  -9.6942  -8.4900
PG        =   1.8140   0.8000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2975  11.2975  29.5930  11.2975  11.2975
MC_gen    =  11.2975  12.1242   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  1.19011   LambdaPs =  0.00000
  2. Ps( 2, 3) =  0.95000   LambdaPs = 18.29553
  3. Ps( 2, 4) =  0.46428   LambdaPs =  0.00000
  4. Ps( 1, 5) =  0.62389   LambdaPs =  0.00000
  5. Ps( 2, 5) =  0.36275   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.45336   LambdaPs = -0.00000
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1607.8943 $/hour
=============================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:       8      15      16      17      18
Upper Limits       :     Ps  2   PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    = 18.2955100.9829 83.3180101.4371100.8936
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000  0.0000
Ps( 2) is flow on line ( 2, 3).
-----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:      14
Lower Limits       :     PG  2
Mu of Lower Limits    =  0.8267
Slack of Lower Limits =  0.0000
=============================================================================
Print statement for SUBPROBLEM   5 which is line ( 1, 5).
============================Variables=====================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.1017  -0.2792  -0.2584  -0.2480
Angl(deg) =   0.0000  -5.8286 -15.9984 -14.8062 -14.2109
PG        =   1.5500   1.0640   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2542  12.1651  12.1651  12.1651  12.1651
MC_gen    =  11.2542  12.1651   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
```

```
  1. Ps( 1, 2) =  1.55000   LambdaPs =  0.91093
  2. Ps( 2, 3) =  0.84488   LambdaPs =  0.00000
  3. Ps( 2, 4) =  0.79866   LambdaPs =  0.00000
  4. Ps( 3, 4) = -0.10512   LambdaPs = -0.00000
  5. Ps( 2, 5) =  0.75738   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.22410   LambdaPs = -0.00000
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1584.9580 $/hour
=====================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:       7      15      16      17      18
Upper Limits       :     Ps  1   PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    =  0.9109100.1153100.7459100.5695100.0259
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000  0.0000
Ps( 1) is flow on line ( 1, 2).
=====================================================================
Print statement for SUBPROBLEM   6 which is line ( 2, 5).
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0645  -0.2369  -0.2113  -0.1975
Angl(deg) =   0.0000  -3.6943 -13.5741 -12.1086 -11.3163
PG        =   1.8140   0.8000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2975  11.2975  11.2975  11.2975  11.2975
MC_gen    =  11.2975  12.1242   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  0.98242   LambdaPs =  0.00000
  2. Ps( 2, 3) =  0.82079   LambdaPs =  0.00000
  3. Ps( 2, 4) =  0.74855   LambdaPs =  0.00000
  4. Ps( 3, 4) = -0.12921   LambdaPs = -0.00000
  5. Ps( 1, 5) =  0.83158   LambdaPs =  0.00000
  6. Ps( 4, 5) = -0.29830   LambdaPs = -0.00000
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =      -0.0000 $/hour
Cost      =   -1607.8943 $/hour
=====================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:      15      16      17      18
Upper Limits       :     PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    =100.9829101.6135101.4371100.8936
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:      14
Lower Limits       :     PG  2
Mu of Lower Limits    =  0.8267
Slack of Lower Limits =  0.0000
=====================================================================
Print statement for SUBPROBLEM   7 which is line ( 4, 5).
============================Variables================================
Calculated values at the end of iteration   41 are:
Bus number       1        2        3        4        5
Angl(rad) =   0.0000  -0.0897  -0.2825  -0.2761  -0.1063
Angl(deg) =   0.0000  -5.1388 -16.1845 -15.8175  -6.0888
PG        =   1.8140   0.8000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2131   0.9500   0.9176   0.5333
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2975  11.2975  22.7846  33.6087  11.2975
MC_gen    =  11.2975  12.1242   0.0000   0.0000   0.0000
```

```
MB_load   =   0.0000  12.2804  12.9110  12.7346  12.1910
  1. Ps( 1, 2) =  1.36656   LambdaPs =  0.00000
  2. Ps( 2, 3) =  0.91765   LambdaPs =  0.00000
  3. Ps( 2, 4) =  0.95000   LambdaPs = 33.03848
  4. Ps( 3, 4) = -0.03235   LambdaPs = -0.00000
  5. Ps( 1, 5) =  0.44744   LambdaPs =  0.00000
  6. Ps( 2, 5) =  0.08584   LambdaPs =  0.00000
PG_total  =       2.6140 $/hour
PL_total  =       2.6140 $/hour
Loss      =       0.0000 $/hour
Cost      =   -1607.8943 $/hour
==========================================================================
Nu = 5.005e-16
=================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:       9      15      16      17      18
Upper Limits       :     Ps  3   PL  2   PL  3   PL  4   PL  5
Mu of Upper Limits    = 33.0385100.9829 90.1264 79.1259100.8936
Slack of Upper Limits =  0.0000  0.0000  0.0000  0.0000  0.0000
Ps( 3) is flow on line ( 2, 4).
--------------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:      14
Lower Limits       :     PG  2
Mu of Lower Limits    =  0.8267
Slack of Lower Limits =  0.0000
==========================================================================
Expected Security Cost  =   -1576.1276 $/hour
```

C.11 Standard OPF of AC 5-bus case

```
>> ESCOPF
Case name: ieee5
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration   39 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0651   1.0334   1.0250   1.0323
Angl(rad) =   0.0000  -0.0681  -0.2010  -0.1570  -0.1338
Angl(deg) =   0.0000  -3.9030 -11.5174  -8.9966  -7.6660
PG        =   1.9818   0.7160   0.0000  -0.0000   0.0000
QG        =  -0.0569   0.3289   0.3459   0.0000   0.0000
PL        =   0.0000   0.2170   0.9406   0.9160   0.5148
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3250  12.1111  13.0106  12.7202  12.5047
LambdaQ   =  -0.0000   0.0000   0.0000   0.1547   0.1522
MC_gen    =  11.3250  12.1111   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1111  13.0106  12.7516  12.5356
  1. Is( 2, 1) =    1.1935   LambdaI =    0.0000
```

```
  2. Is( 5, 1) =    0.6503   LambdaI =    0.0000
  3. Is( 1, 2) =    1.2000   LambdaI =    0.3670
  4. Is( 3, 2) =    0.7009   LambdaI =    0.0000
  5. Is( 4, 2) =    0.5459   LambdaI =    0.0000
  6. Is( 5, 2) =    0.4188   LambdaI =    0.0000
  7. Is( 2, 3) =    0.7013   LambdaI =    0.0000
  8. Is( 4, 3) =    0.2601   LambdaI =    0.0000
  9. Is( 2, 4) =    0.5425   LambdaI =    0.0000
 10. Is( 3, 4) =    0.2413   LambdaI =    0.0000
 11. Is( 5, 4) =    0.5654   LambdaI =    0.0000
 12. Is( 1, 5) =    0.6457   LambdaI =    0.0000
 13. Is( 2, 5) =    0.4143   LambdaI =    0.0000
 14. Is( 4, 5) =    0.5653   LambdaI =    0.0000
PG_total  =       2.6978 $/hour
PL_total  =       2.5884 $/hour
Loss      =       0.1093 $/hour
Cost      =   -1487.6028 $/hour
========================================================================
Nu = 5.005e-16
==================Conditions in IP=====================================
Binding constraints on upper limits are:
Upper Limits Number:        1       21
Upper Limits       :     V   1    Is  3
Mu of Upper Limits    =   3.0831   0.3670
Slack of Upper Limits =   0.0000   0.0000
Is( 3) is flow on line ( 1, 2).
========================================================================
Expected Security Cost  =   -1487.6028 $/hour
```

C.12 ESCOPF of AC 5-bus case

```
>> ESCOPF
Case name: ieee5
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0744   1.0430   1.0337   1.0402
Angl(rad) =   0.0000  -0.0493  -0.1757  -0.1376  -0.1168
Angl(deg) =   0.0000  -2.8275 -10.0643  -7.8839  -6.6920
PG        =   1.5174   1.0818   0.0000  -0.0000   0.0000
QG        =  -0.1513   0.3623   0.3293  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2489  11.5891  12.4348  12.2318  12.0657
LambdaQ   =  -0.0000   0.0000   0.0000   0.1319   0.1249
MC_gen    =  11.2489  12.1679   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    0.8523   LambdaI =    0.0000
```

```
  2. Is( 5, 1) =    0.5672   LambdaI =    0.0000
  3. Is( 1, 2) =    0.8642   LambdaI =    0.0000
  4. Is( 3, 2) =    0.6734   LambdaI =    0.0000
  5. Is( 4, 2) =    0.5479   LambdaI =    0.0000
  6. Is( 5, 2) =    0.4340   LambdaI =    0.0000
  7. Is( 2, 3) =    0.6735   LambdaI =    0.0000
  8. Is( 4, 3) =    0.2313   LambdaI =    0.0000
  9. Is( 2, 4) =    0.5442   LambdaI =    0.0000
 10. Is( 3, 4) =    0.2109   LambdaI =    0.0000
 11. Is( 5, 4) =    0.5102   LambdaI =    0.0000
 12. Is( 1, 5) =    0.5634   LambdaI =    0.0000
 13. Is( 2, 5) =    0.4294   LambdaI =    0.0000
 14. Is( 4, 5) =    0.5101   LambdaI =    0.0000
PG_total  =       2.5993 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.0877 $/hour
Cost      =   -1466.9101 $/hour
======================================================================
Nu = 5.005e-16
==================Conditions in IP====================================
Binding constraints on upper limits are:
Upper Limits Number:        1
Upper Limits       :     V   1
Mu of Upper Limits    =   2.0546
Slack of Upper Limits =   0.0000
======================================================================
Print statement for SUBPROBLEM   1 which is line ( 1, 2).
============================Variables=================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0800   1.0488   1.0377   1.0425
Angl(rad) =   0.0000  -0.2147  -0.3231  -0.2689  -0.2359
Angl(deg) =   0.0000 -12.2988 -18.5149 -15.4044 -13.5166
PG        =   1.1880   1.4318   0.0000  -0.0000   0.0000
QG        =   0.0048   0.3387   0.3375  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.1948  67.4020  71.7663  69.8643  68.4595
LambdaQ   =  -0.0000   0.0000   0.0000   1.0797   1.2017
MC_gen    =  11.1948  12.2222   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 5, 1) =    1.0954   LambdaI =    0.0000
  2. Is( 3, 2) =    0.5875   LambdaI =    0.0000
  3. Is( 4, 2) =    0.3898   LambdaI =    0.0000
  4. Is( 5, 2) =    0.2508   LambdaI =    0.0000
  5. Is( 2, 3) =    0.5860   LambdaI =    0.0000
  6. Is( 4, 3) =    0.3237   LambdaI =    0.0000
  7. Is( 2, 4) =    0.3775   LambdaI =    0.0000
  8. Is( 3, 4) =    0.3044   LambdaI =    0.0000
  9. Is( 5, 4) =    0.7843   LambdaI =    0.0000
 10. Is( 1, 5) =    1.1000   LambdaI =   53.1177
 11. Is( 2, 5) =    0.2273   LambdaI =    0.0000
 12. Is( 4, 5) =    0.7821   LambdaI =    0.0000
PG_total  =       2.6198 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1083 $/hour
Cost      =   -1409.7955 $/hour
======================================================================
Nu = 5.005e-16
==================Conditions in IP====================================
Binding constraints on upper limits are:
Upper Limits Number:        1        2       28       32       33       34
                           35       36
Upper Limits       :     V   1    V   2    Is  12   PG  2    PL  2    PL  3
                         PL  4    PL  5
Mu of Upper Limits    =  66.5908   2.3110  53.1177  55.1798  44.7680  41.7904
                         42.8268  44.0022
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

```
                        0.0000   0.0000
Is(12) is flow on line ( 1, 5).
================================================================================
Print statement for SUBPROBLEM   2 which is line ( 2, 3).
============================Variables===================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0753   0.9697   1.0048   1.0188
Angl(rad) =   0.0000  -0.0447  -0.3644  -0.1913  -0.1527
Angl(deg) =   0.0000  -2.5598 -20.8778 -10.9607  -8.7498
PG        =   1.6125   1.0876   0.0000  -0.0000   0.0000
QG        =  -0.0615   0.5000   0.4000  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2644  12.1688  62.8941  15.5543  14.3777
LambdaQ   =  -0.0000   1.8434   0.6951   5.4365   4.2324
MC_gen    =  11.2644  12.1688   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    0.7705   LambdaI =    0.0000
  2. Is( 5, 1) =    0.7483   LambdaI =    0.0000
  3. Is( 1, 2) =    0.7829   LambdaI =    0.0000
  4. Is( 4, 2) =    0.9050   LambdaI =    0.0000
  5. Is( 5, 2) =    0.6926   LambdaI =    0.0000
  6. Is( 4, 3) =    0.9500   LambdaI =   39.5847
  7. Is( 2, 4) =    0.9005   LambdaI =    0.0000
  8. Is( 3, 4) =    0.9442   LambdaI =    0.0000
  9. Is( 5, 4) =    0.9384   LambdaI =    0.0000
 10. Is( 1, 5) =    0.7417   LambdaI =    0.0000
 11. Is( 2, 5) =    0.6874   LambdaI =    0.0000
 12. Is( 4, 5) =    0.9388   LambdaI =    0.0000
PG_total  =       2.7001 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1885 $/hour
Cost      =   -1352.9307 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP=================================
Binding constraints on upper limits are:
Upper Limits Number:        1       13       14       24       33       34
                           35       36
Upper Limits       :     V   1    QG  2    QG  3    Is  8    PL  2    PL  3
                         PL  4    PL  5
Mu of Upper Limits    =  55.5998   1.8434   0.6951  39.5847  99.6269  50.5216
                         96.2520  97.4686
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
Is( 8) is flow on line ( 4, 3).
================================================================================
Print statement for SUBPROBLEM   3 which is line ( 2, 4).
============================Variables===================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0740   1.0375   1.0101   1.0223
Angl(rad) =   0.0000  -0.0585  -0.2151  -0.1986  -0.1602
Angl(deg) =   0.0000  -3.3527 -12.3229 -11.3773  -9.1790
PG        =   1.9020   0.7318  -0.0000  -0.0000   0.0000
QG        =  -0.1265   0.4216   0.4000  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3119  11.7240  12.7441  12.7607  12.4514
LambdaQ   =  -0.0000   0.0000   0.0205   0.2545   0.2348
MC_gen    =  11.3119  12.1136   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    1.0099   LambdaI =    0.0000
  2. Is( 5, 1) =    0.7754   LambdaI =    0.0000
  3. Is( 1, 2) =    1.0225   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8292   LambdaI =    0.0000
  5. Is( 5, 2) =    0.6488   LambdaI =    0.0000
```

```
  6. Is( 2, 3) =    0.8299   LambdaI =    0.0000
  7. Is( 4, 3) =    0.1925   LambdaI =    0.0000
  8. Is( 3, 4) =    0.1577   LambdaI =    0.0000
  9. Is( 5, 4) =    0.9249   LambdaI =    0.0000
 10. Is( 1, 5) =    0.7706   LambdaI =    0.0000
 11. Is( 2, 5) =    0.6440   LambdaI =    0.0000
 12. Is( 4, 5) =    0.9248   LambdaI =    0.0000
PG_total  =       2.6338 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1222 $/hour
Cost      =   -1458.0991 $/hour
=============================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:        1       14       33       34       35       36
Upper Limits       :     V   1    QG  3    PL  2    PL  3    PL  4    PL  5
Mu of Upper Limits    =   3.0071   0.0205 100.4460 100.8085 100.0979 100.2066
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
-----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       32
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.3895
Slack of Lower Limits =   0.0000
=============================================================================
Print statement for SUBPROBLEM   4 which is line ( 3, 4).
============================Variables=====================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0698   1.0434   1.0260   1.0345
Angl(rad) =   0.0000  -0.0677  -0.2324  -0.1359  -0.1189
Angl(deg) =   0.0000  -3.8811 -13.3158  -7.7862  -6.8130
PG        =   1.8816   0.7318  -0.0000   0.0000  -0.0000
QG        =  -0.1368   0.4460   0.3056  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3086  11.7844  12.7522  12.3390  12.1815
LambdaQ   =  -0.0000   0.0000   0.0000   0.1683   0.1472
MC_gen    =  11.3086  12.1136   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    1.1749   LambdaI =    0.0000
  2. Is( 5, 1) =    0.5840   LambdaI =    0.0000
  3. Is( 1, 2) =    1.1850   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8605   LambdaI =    0.0000
  5. Is( 4, 2) =    0.4554   LambdaI =    0.0000
  6. Is( 5, 2) =    0.3567   LambdaI =    0.0000
  7. Is( 2, 3) =    0.8642   LambdaI =    0.0000
  8. Is( 2, 4) =    0.4464   LambdaI =    0.0000
  9. Is( 5, 4) =    0.4388   LambdaI =    0.0000
 10. Is( 1, 5) =    0.5783   LambdaI =    0.0000
 11. Is( 2, 5) =    0.3473   LambdaI =    0.0000
 12. Is( 4, 5) =    0.4406   LambdaI =    0.0000
PG_total  =       2.6134 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1019 $/hour
Cost      =   -1481.1280 $/hour
=============================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:        1       33       34       35       36
Upper Limits       :     V   1    PL  2    PL  3    PL  4    PL  5
Mu of Upper Limits    =   2.4082 100.3856 100.8046 100.5371 100.4943
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000
-----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       32
```

```
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.3292
Slack of Lower Limits =   0.0000
================================================================================
Print statement for SUBPROBLEM   5 which is line ( 1, 5).
============================Variables===================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0618   1.0298   1.0068   1.0079
Angl(rad) =   0.0000  -0.0884  -0.2389  -0.2144  -0.2047
Angl(deg) =   0.0000  -5.0636 -13.6902 -12.2819 -11.7278
PG        =   1.6655   0.9878  -0.0000  -0.0000   0.0000
QG        =  -0.1681   0.5000   0.4000  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2731  12.1533  13.2516  13.2333  13.1658
LambdaQ   =  -0.0000   0.0703   0.0624   0.2735   0.2800
MC_gen    =  11.2731  12.1533   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    1.5429   LambdaI =    0.0000
  2. Is( 1, 2) =    1.5500   LambdaI =    0.1763
  3. Is( 3, 2) =    0.7867   LambdaI =    0.0000
  4. Is( 4, 2) =    0.7616   LambdaI =    0.0000
  5. Is( 5, 2) =    0.7211   LambdaI =    0.0000
  6. Is( 2, 3) =    0.7882   LambdaI =    0.0000
  7. Is( 4, 3) =    0.2006   LambdaI =    0.0000
  8. Is( 2, 4) =    0.7585   LambdaI =    0.0000
  9. Is( 3, 4) =    0.1690   LambdaI =    0.0000
 10. Is( 5, 4) =    0.2232   LambdaI =    0.0000
 11. Is( 2, 5) =    0.7175   LambdaI =    0.0000
 12. Is( 4, 5) =    0.2207   LambdaI =    0.0000
PG_total  =       2.6533 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1418 $/hour
Cost      =   -1414.4894 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP================================
Binding constraints on upper limits are:
Upper Limits Number:        1       13       14       20       33       34
                           35       36
Upper Limits       :     V  1    QG  2    QG  3    Is  3    PL  2    PL  3
                         PL  4    PL  5
Mu of Upper Limits    =   3.8860   0.0703   0.0624   0.1763 100.0024 100.2925
                         99.6214  99.4830
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
Is( 3) is flow on line ( 1, 2).
================================================================================
Print statement for SUBPROBLEM   6 which is line ( 2, 5).
============================Variables===================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0800   1.0727   1.0401   1.0196   1.0211
Angl(rad) =   0.0000  -0.0575  -0.2002  -0.1727  -0.1604
Angl(deg) =   0.0000  -3.2966 -11.4724  -9.8948  -9.1896
PG        =   1.8914   0.7318   0.0000  -0.0000   0.0000
QG        =  -0.0943   0.3657   0.3796  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3102  11.7177  12.6499  12.5510  12.4327
LambdaQ   =  -0.0000   0.0000   0.0000   0.2022   0.2197
MC_gen    =  11.3102  12.1136   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    0.9953   LambdaI =    0.0000
  2. Is( 5, 1) =    0.7776   LambdaI =    0.0000
  3. Is( 1, 2) =    1.0066   LambdaI =    0.0000
  4. Is( 3, 2) =    0.7556   LambdaI =    0.0000
```

```
  5. Is( 4, 2) =    0.7097   LambdaI =    0.0000
  6. Is( 2, 3) =    0.7565   LambdaI =    0.0000
  7. Is( 4, 3) =    0.2054   LambdaI =    0.0000
  8. Is( 2, 4) =    0.7059   LambdaI =    0.0000
  9. Is( 3, 4) =    0.1755   LambdaI =    0.0000
 10. Is( 5, 4) =    0.2875   LambdaI =    0.0000
 11. Is( 1, 5) =    0.7725   LambdaI =    0.0000
 12. Is( 4, 5) =    0.2851   LambdaI =    0.0000
PG_total  =       2.6233 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1117 $/hour
Cost      =   -1470.0150 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP==================================
Binding constraints on upper limits are:
Upper Limits Number:        1       33       34       35       36
Upper Limits       :     V   1    PL  2    PL  3    PL  4    PL  5
Mu of Upper Limits    =   2.7057 100.4524 100.9069 100.3182 100.2283
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000
--------------------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       32
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.3959
Slack of Lower Limits =   0.0000
================================================================================
Print statement for SUBPROBLEM   7 which is line ( 4, 5).
============================Variables=====================================
Calculated values at the end of iteration   43 are:
Bus number       1        2        3        4        5
V         =   1.0797   1.0601   1.0239   0.9974   1.0500
Angl(rad) =   0.0000  -0.0768  -0.2424  -0.2303  -0.0855
Angl(deg) =   0.0000  -4.4013 -13.8875 -13.1948  -4.8971
PG        =   1.9105   0.7318   0.0000  -0.0000  -0.0000
QG        =  -0.0696   0.3421   0.4000  -0.0000  -0.0000
PL        =   0.0000   0.2156   0.8892   0.9011   0.5056
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3133  11.8757  13.0211  13.1228  11.7968
LambdaQ   =  -0.0000   0.0000   0.0251   0.2358  -0.2352
MC_gen    =  11.3133  12.1136   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1700  13.5568  12.9103  12.7057
  1. Is( 2, 1) =    1.3537   LambdaI =    0.0000
  2. Is( 5, 1) =    0.4187   LambdaI =    0.0000
  3. Is( 1, 2) =    1.3583   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8631   LambdaI =    0.0000
  5. Is( 4, 2) =    0.9137   LambdaI =    0.0000
  6. Is( 5, 2) =    0.0846   LambdaI =    0.0000
  7. Is( 2, 3) =    0.8643   LambdaI =    0.0000
  8. Is( 4, 3) =    0.1760   LambdaI =    0.0000
  9. Is( 2, 4) =    0.9114   LambdaI =    0.0000
 10. Is( 3, 4) =    0.1411   LambdaI =    0.0000
 11. Is( 1, 5) =    0.4147   LambdaI =    0.0000
 12. Is( 2, 5) =    0.0672   LambdaI =    0.0000
PG_total  =       2.6423 $/hour
PL_total  =       2.5116 $/hour
Loss      =       0.1308 $/hour
Cost      =   -1448.4235 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP==================================
Binding constraints on upper limits are:
Upper Limits Number:        5       14       33       34       35       36
Upper Limits       :     V   5    QG  3    PL  2    PL  3    PL  4    PL  5
Mu of Upper Limits    =   3.1172   0.0251 100.2943 100.5306  99.7396 100.9567
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
--------------------------------------------------------------------------------
Binding constraints on lower limits are:
```

```
Lower Limits Number:        32
Lower Limits        :     PG  2
Mu of Lower Limits    =   0.2379
Slack of Lower Limits =   0.0000
========================================================================
Expected Security Cost =   -1464.5752 $/hour
```

C.13 Standard OPF of DC IEEE 14-bus case

```
>> ESCOPF
DC Case name: ieee14
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables================================
Calculated values at the end of iteration   32 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0788  -0.2363  -0.1939  -0.1651  -0.2769  -0.2587
             -0.2587  -0.2929  -0.2993  -0.2929  -0.3013  -0.3070  -0.3293
Angl(deg) =   0.0000  -4.5125 -13.5402 -11.1069  -9.4612 -15.8642 -14.8243
            -14.8243 -16.7799 -17.1500 -16.7798 -17.2649 -17.5881 -18.8695
PG        =   1.8953   0.8006   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2168   0.9645   0.5034   0.0793   0.1168   0.0000
                0.0000   0.3108   0.0954   0.0370   0.0649   0.1445   0.1622
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3108  12.1243  12.7559  12.1697  12.0501  12.0877  12.1499
               12.1499  12.1395  12.1307  12.1100  12.0909  12.0955  12.1204
MC_gen    =  11.3108  12.1243   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.1243  12.7559  12.1697  12.0501  12.0877   0.0000
                0.0000  12.1395  12.1307  12.1100  12.0909  12.0955  12.1204
  1. Ps( 1, 2) =    1.2000      LambdaPs =    1.0177
  2. Ps( 2, 3) =    0.7500      LambdaPs =    1.2537
  3. Ps( 2, 4) =    0.5866      LambdaPs =    0.0000
  4. Ps( 3, 4) =   -0.2145      LambdaPs =   -0.0000
  5. Ps( 1, 5) =    0.6953      LambdaPs =    0.0000
  6. Ps( 2, 5) =    0.4471      LambdaPs =    0.0000
  7. Ps( 4, 5) =   -0.6196      LambdaPs =   -0.0000
  8. Ps( 5, 6) =    0.4434      LambdaPs =    0.0000
  9. Ps( 4, 7) =    0.3103      LambdaPs =    0.0000
 10. Ps( 7, 8) =    0.0000      LambdaPs =   -0.0000
 11. Ps( 4, 9) =    0.1780      LambdaPs =    0.0000
 12. Ps( 7, 9) =    0.3103      LambdaPs =    0.0000
 13. Ps( 9,10) =    0.0670      LambdaPs =    0.0000
 14. Ps( 6,11) =    0.0654      LambdaPs =    0.0000
 15. Ps(10,11) =   -0.0285      LambdaPs =   -0.0000
 16. Ps( 6,12) =    0.0776      LambdaPs =    0.0000
 17. Ps( 6,13) =    0.1836      LambdaPs =    0.0000
```

```
 18. Ps(12,13) =    0.0127      LambdaPs =    0.0000
 19. Ps( 9,14) =    0.1105      LambdaPs =    0.0000
 20. Ps(13,14) =    0.0518      LambdaPs =    0.0000
PG_total  =       2.6959 pu.
PL_total  =       2.6959 pu.
Loss      =      -0.0000 pu.
Cost      =    -243.4211 $/hour
=====================================================================
Nu = 5.005e-16
============================Constraints==============================
Binding constraints on upper limits are:
Upper Limits Number:       14       15
Upper Limits       :     Ps  1    Ps  2
Mu of Upper Limits    =   1.0177   1.2537
Slack of Upper Limits =   0.0000   0.0000
Ps( 1) is flow on line ( 1, 2).
Ps( 2) is flow on line ( 2, 3).
=====================================================================
Expected Security Cost  =    -243.4211 $/hour
```

C.14 ESCOPF of DC IEEE 14-bus case

```
>> ESCOPF
DC Case name: ieee14
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=============================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0579  -0.2153  -0.1756  -0.1486  -0.2582  -0.2398
             -0.2398  -0.2735  -0.2800  -0.2738  -0.2822  -0.2878  -0.3096
Angl(deg) =   0.0000  -3.3170 -12.3364 -10.0600  -8.5130 -14.7944 -13.7379
            -13.7379 -15.6726 -16.0419 -15.6862 -16.1706 -16.4876 -17.7414
PG        =   1.5077   1.1500   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2473  11.2473  11.2473  11.2473  11.2473  11.2473  11.2473
             11.2473  11.2473  11.2473  11.2473  11.2473  11.2473  11.2473
MC_gen    =  11.2473  12.1785   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
              0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    0.8821      LambdaPs =    0.0000
  2. Ps( 2, 3) =    0.7493      LambdaPs =    0.0000
  3. Ps( 2, 4) =    0.5999      LambdaPs =    0.0000
  4. Ps( 3, 4) =   -0.2007      LambdaPs =   -0.0000
```

```
  5. Ps( 1, 5) =    0.6256      LambdaPs =    0.0000
  6. Ps( 2, 5) =    0.4695      LambdaPs =    0.0000
  7. Ps( 4, 5) =   -0.5825      LambdaPs =   -0.0000
  8. Ps( 5, 6) =    0.4350      LambdaPs =    0.0000
  9. Ps( 4, 7) =    0.3070      LambdaPs =    0.0000
 10. Ps( 7, 8) =    0.0000      LambdaPs =   -0.0000
 11. Ps( 4, 9) =    0.1761      LambdaPs =    0.0000
 12. Ps( 7, 9) =    0.3070      LambdaPs =    0.0000
 13. Ps( 9,10) =    0.0668      LambdaPs =    0.0000
 14. Ps( 6,11) =    0.0637      LambdaPs =    0.0000
 15. Ps(10,11) =   -0.0273      LambdaPs =   -0.0000
 16. Ps( 6,12) =    0.0763      LambdaPs =    0.0000
 17. Ps( 6,13) =    0.1804      LambdaPs =    0.0000
 18. Ps(12,13) =    0.0125      LambdaPs =    0.0000
 19. Ps( 9,14) =    0.1094      LambdaPs =    0.0000
 20. Ps(13,14) =    0.0507      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -208.5145 $/hour
======================================================================
Nu = 5.005e-16
============================Constraints===============================
======================================================================
Print statement for SUBPROBLEM   1 which is line ( 1, 2).
============================Variables=================================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.2181  -0.3566  -0.2990  -0.2613  -0.3709  -0.3591
             -0.3591  -0.3907  -0.3961  -0.3881  -0.3944  -0.4002  -0.4239
Angl(deg) =   0.0000 -12.4971 -20.4298 -17.1295 -14.9690 -21.2507 -20.5755
            -20.5755 -22.3883 -22.6962 -22.2369 -22.5997 -22.9322 -24.2853
PG        =   1.1000   1.5000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2055   0.9500   0.4819   0.0749   0.1107   0.0000
              0.0000   0.2966   0.0910   0.0352   0.0617   0.1375   0.1550
PI        =   0.0000   0.0079   0.0000   0.0166   0.0027   0.0040   0.0000
              0.0000   0.0103   0.0031   0.0012   0.0021   0.0047   0.0051
LambdaP   =  11.1804 112.6094 112.6094 112.6094 112.6094 112.6094 112.6094
             112.6094 112.6094 112.6094 112.6094 112.6094 112.6094 112.6094
MC_gen    =  11.1804  12.2328   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.6094  12.9100  12.6094  12.6094  12.6094   0.0000
              0.0000  12.6094  12.6094  12.6094  12.6094  12.6094  12.6094
  1. Ps( 2, 3) =    0.6590      LambdaPs =    0.0000
  2. Ps( 2, 4) =    0.4121      LambdaPs =    0.0000
  3. Ps( 3, 4) =   -0.2910      LambdaPs =   -0.0000
  4. Ps( 1, 5) =    1.1000      LambdaPs =  101.4290
  5. Ps( 2, 5) =    0.2234      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -0.8134      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4350      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.2876      LambdaPs =    0.0000
  9. Ps( 7, 8) =    0.0000      LambdaPs =    0.0000
 10. Ps( 4, 9) =    0.1650      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.2876      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0557      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0705      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0353      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0748      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1791      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0131      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.1003      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0547      LambdaPs =    0.0000
PG_total  =       2.6000 pu.
PL_total  =       2.6000 pu.
Loss      =      -0.0000 pu.
Cost      =     409.9982 $/hour
```

```
================================================================
Nu = 5.005e-16
============================Constraints=========================
Binding constraints on upper limits are:
Upper Limits Number:        2       17       34       36
Upper Limits       :     PG  2    Ps  5    PG  2    PL  3
Mu of Upper Limits    =   9.6754 101.4290  90.7011   0.3006
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000
Ps( 5) is flow on line ( 1, 5).
================================================================
Print statement for SUBPROBLEM   2 which is line ( 2, 3).
============================Variables===========================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0667  -0.4340  -0.2459  -0.1998  -0.3155  -0.3070
             -0.3070  -0.3391  -0.3441  -0.3346  -0.3400  -0.3463  -0.3721
Angl(deg) =   0.0000  -3.8216 -24.8660 -14.0908 -11.4500 -18.0748 -17.5875
            -17.5875 -19.4270 -19.7158 -19.1705 -19.4806 -19.8393 -21.3213
PG        =   1.8577   0.8000   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
                0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3047  11.3047  43.1789  11.3047  11.3047  11.3047  11.3047
               11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
MC_gen    =  11.3047  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
                0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
                0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.0163      LambdaPs =    0.0000
  2. Ps( 2, 4) =    0.9136      LambdaPs =    0.0000
  3. Ps( 3, 4) =   -0.9500      LambdaPs =  -31.8742
  4. Ps( 1, 5) =    0.8414      LambdaPs =    0.0000
  5. Ps( 2, 5) =    0.6893      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -0.9943      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4588      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.2918      LambdaPs =    0.0000
  9. Ps( 7, 8) =    0.0000      LambdaPs =    0.0000
 10. Ps( 4, 9) =    0.1675      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.2918      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0522      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0783      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0419      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0779      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1879      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0141      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.1001      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0599      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -239.1522 $/hour
================================================================
Nu = 5.005e-16
============================Constraints=========================
Binding constraints on upper limits are:
Upper Limits Number:       35       36       37       38       39       40
                           41       42       43       44       45
Upper Limits       :     PL  2    PL  3    PL  4    PL  5    PL  6    PL  9
                         PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    = 100.9649  69.7311 100.9649 100.9649 100.9649 100.9649
                        100.9649 100.9649 100.9649 100.9649 100.9649
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000
----------------------------------------------------------------
Binding constraints on lower limits are:
```

```
Lower Limits Number:       16       34
Lower Limits       :     Ps  4    PG  2
Mu of Lower Limits    =  31.8742   0.8195
Slack of Lower Limits =   0.0000   0.0000
Ps( 4) is flow on line ( 3, 4).
=====================================================================
Print statement for SUBPROBLEM   3 which is line ( 2, 4).
============================Variables================================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0665  -0.2564  -0.2472  -0.2007  -0.3165  -0.3082
             -0.3082  -0.3403  -0.3453  -0.3357  -0.3410  -0.3473  -0.3733
Angl(deg) =   0.0000  -3.8075 -14.6883 -14.1660 -11.5010 -18.1334 -17.6587
            -17.6587 -19.4961 -19.7830 -19.2336 -19.5398 -19.8994 -21.3865
PG        =   1.8577   0.8000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
             11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
MC_gen    =  11.3047  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
              0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.0125      LambdaPs =    0.0000
  2. Ps( 2, 3) =    0.9039      LambdaPs =    0.0000
  3. Ps( 3, 4) =   -0.0461      LambdaPs =   -0.0000
  4. Ps( 1, 5) =    0.8452      LambdaPs =    0.0000
  5. Ps( 2, 5) =    0.6951      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -1.0034      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4593      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.2915      LambdaPs =    0.0000
  9. Ps( 7, 8) =   -0.0000      LambdaPs =   -0.0000
 10. Ps( 4, 9) =    0.1673      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.2915      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0519      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0786      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0422      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0780      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1881      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0141      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.0999      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0601      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =       0.0000 pu.
Cost      =    -239.1522 $/hour
=====================================================================
Nu = 5.005e-16
============================Constraints==============================
Binding constraints on upper limits are:
Upper Limits Number:       35       36       37       38       39       40
                           41       42       43       44       45
Upper Limits       :     PL  2    PL  3    PL  4    PL  5    PL  6    PL  9
                         PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    = 100.9649 101.6053 100.9649 100.9649 100.9649 100.9649
                        100.9649 100.9649 100.9649 100.9649 100.9649
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       34
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.8195
Slack of Lower Limits =   0.0000
```

```
=====================================================================
Print statement for SUBPROBLEM   4 which is line ( 3, 4).
============================Variables================================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0800  -0.2796  -0.1750  -0.1517  -0.2602  -0.2398
             -0.2398  -0.2739  -0.2806  -0.2750  -0.2841  -0.2895  -0.3106
Angl(deg) =   0.0000  -4.5835 -16.0187 -10.0262  -8.6926 -14.9069 -13.7393
            -13.7393 -15.6927 -16.0777 -15.7590 -16.2774 -16.5863 -17.7955
PG        =   1.8577   0.8000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3047  11.3047  43.1309  11.3047  11.3047  11.3047  11.3047
             11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
MC_gen    =  11.3047  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
              0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.2189      LambdaPs =    0.0000
  2. Ps( 2, 3) =    0.9500      LambdaPs =   31.8262
  3. Ps( 2, 4) =    0.4842      LambdaPs =    0.0000
  4. Ps( 1, 5) =    0.6388      LambdaPs =    0.0000
  5. Ps( 2, 5) =    0.3713      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -0.5021      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4304      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.3099      LambdaPs =    0.0000
  9. Ps( 7, 8) =    0.0000      LambdaPs =    0.0000
 10. Ps( 4, 9) =    0.1778      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.3099      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0696      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0609      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0245      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0760      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1789      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0121      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.1112      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0489      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -239.1522 $/hour
=====================================================================
Nu = 5.005e-16
============================Constraints==============================
Binding constraints on upper limits are:
Upper Limits Number:       15       35       36       37       38       39
                           40       41       42       43       44       45
Upper Limits       :     Ps  2    PL  2    PL  3    PL  4    PL  5    PL  6
                         PL  9    PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    =  31.8262 100.9649  69.7791 100.9649 100.9649 100.9649
                        100.9649 100.9649 100.9649 100.9649 100.9649 100.9649
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
Ps( 2) is flow on line ( 2, 3).
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       34
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.8195
Slack of Lower Limits =   0.0000
=====================================================================
Print statement for SUBPROBLEM   5 which is line ( 1, 5).
============================Variables================================
Calculated values at the end of iteration   42 are:
```

```
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.1017  -0.2815  -0.2627  -0.2501  -0.3552  -0.3293
             -0.3293  -0.3643  -0.3718  -0.3681  -0.3788  -0.3838  -0.4027
Angl(deg) =   0.0000  -5.8286 -16.1262 -15.0543 -14.3286 -20.3520 -18.8681
            -18.8681 -20.8744 -21.3041 -21.0908 -21.7060 -21.9918 -23.0741
PG        =   1.5500   1.1077   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2542  12.1719  12.1719  12.1719  12.1719  12.1719  12.1719
             12.1719  12.1719  12.1719  12.1719  12.1719  12.1719  12.1719
MC_gen    =  11.2542  12.1719   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
              0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.5500      LambdaPs =    0.9177
  2. Ps( 2, 3) =    0.8555      LambdaPs =    0.0000
  3. Ps( 2, 4) =    0.8207      LambdaPs =    0.0000
  4. Ps( 3, 4) =   -0.0945      LambdaPs =   -0.0000
  5. Ps( 2, 5) =    0.7680      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -0.2733      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4171      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.3183      LambdaPs =    0.0000
  9. Ps( 7, 8) =    0.0000      LambdaPs =   -0.0000
 10. Ps( 4, 9) =    0.1826      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.3183      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0777      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0528      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0164      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0751      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1747      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0112      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.1163      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0437      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -212.4275 $/hour
================================================================
Nu = 5.005e-16
============================Constraints=========================
Binding constraints on upper limits are:
Upper Limits Number:      14       35       36       37       38       39
                          40       41       42       43       44       45
Upper Limits       :    Ps  1    PL  2    PL  3    PL  4    PL  5    PL  6
                        PL  9    PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    =  0.9177 100.0977 100.7381 100.0977 100.0977 100.0977
                       100.0977 100.0977 100.0977 100.0977 100.0977 100.0977
Slack of Upper Limits =  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                         0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
Ps( 1) is flow on line ( 1, 2).
================================================================
Print statement for SUBPROBLEM   6 which is line ( 2, 5).
============================Variables===========================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0664  -0.2408  -0.2171  -0.2010  -0.3072  -0.2831
             -0.2831  -0.3178  -0.3250  -0.3207  -0.3309  -0.3360  -0.3556
Angl(deg) =   0.0000  -3.8038 -13.7960 -12.4364 -11.5144 -17.5995 -16.2177
            -16.2177 -18.2069 -18.6222 -18.3748 -18.9588 -19.2520 -20.3753
PG        =   1.8577   0.8000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
```

```
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
             11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
MC_gen    =  11.3047  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
              0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.0115      LambdaPs =    0.0000
  2. Ps( 2, 3) =    0.8301      LambdaPs =    0.0000
  3. Ps( 2, 4) =    0.7680      LambdaPs =    0.0000
  4. Ps( 3, 4) =   -0.1199      LambdaPs =   -0.0000
  5. Ps( 1, 5) =    0.8461      LambdaPs =    0.0000
  6. Ps( 4, 5) =   -0.3471      LambdaPs =   -0.0000
  7. Ps( 5, 6) =    0.4214      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.3156      LambdaPs =    0.0000
  9. Ps( 7, 8) =   -0.0000      LambdaPs =   -0.0000
 10. Ps( 4, 9) =    0.1811      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.3156      LambdaPs =    0.0000
 12. Ps( 9,10) =    0.0751      LambdaPs =    0.0000
 13. Ps( 6,11) =    0.0554      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.0190      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0754      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.1760      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0115      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.1146      LambdaPs =    0.0000
 19. Ps(13,14) =    0.0454      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -239.1522 $/hour
======================================================================
Nu = 5.005e-16
============================Constraints===============================
Binding constraints on upper limits are:
Upper Limits Number:       35       36       37       38       39       40
                           41       42       43       44       45
Upper Limits       :     PL  2    PL  3    PL  4    PL  5    PL  6    PL  9
                         PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    = 100.9649 101.6053 100.9649 100.9649 100.9649 100.9649
                        100.9649 100.9649 100.9649 100.9649 100.9649
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000
----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       34
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.8195
Slack of Lower Limits =   0.0000
======================================================================
Print statement for SUBPROBLEM   7 which is line ( 4, 5).
============================Variables=================================
Calculated values at the end of iteration   42 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
Angl(rad) =   0.0000  -0.0889  -0.2761  -0.2643  -0.1193  -0.2660  -0.3090
             -0.3090  -0.3325  -0.3302  -0.3036  -0.2932  -0.3032  -0.3498
Angl(deg) =   0.0000  -5.0964 -15.8175 -15.1446  -6.8367 -15.2413 -17.7030
            -17.7030 -19.0489 -18.9204 -17.3931 -16.8001 -17.3747 -20.0393
PG        =   1.8577   0.8000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2134   0.9500   0.4986   0.0776   0.1146   0.0000
              0.0000   0.3069   0.0941   0.0364   0.0638   0.1422   0.1600
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
             11.3047  11.3047  11.3047  11.3047  11.3047  11.3047  11.3047
MC_gen    =  11.3047  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
```

```
             0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =  0.0000  12.2696  12.9100  12.2696  12.2696  12.2696   0.0000
             0.0000  12.2696  12.2696  12.2696  12.2696  12.2696  12.2696
  1. Ps( 1, 2) =    1.3553      LambdaPs =    0.0000
  2. Ps( 2, 3) =    0.8907      LambdaPs =    0.0000
  3. Ps( 2, 4) =    0.8939      LambdaPs =    0.0000
  4. Ps( 3, 4) =   -0.0593      LambdaPs =   -0.0000
  5. Ps( 1, 5) =    0.5024      LambdaPs =    0.0000
  6. Ps( 2, 5) =    0.1573      LambdaPs =    0.0000
  7. Ps( 5, 6) =    0.5820      LambdaPs =    0.0000
  8. Ps( 4, 7) =    0.2135      LambdaPs =    0.0000
  9. Ps( 7, 8) =   -0.0000      LambdaPs =    0.0000
 10. Ps( 4, 9) =    0.1225      LambdaPs =    0.0000
 11. Ps( 7, 9) =    0.2135      LambdaPs =    0.0000
 12. Ps( 9,10) =   -0.0232      LambdaPs =   -0.0000
 13. Ps( 6,11) =    0.1538      LambdaPs =    0.0000
 14. Ps(10,11) =   -0.1174      LambdaPs =   -0.0000
 15. Ps( 6,12) =    0.0864      LambdaPs =    0.0000
 16. Ps( 6,13) =    0.2272      LambdaPs =    0.0000
 17. Ps(12,13) =    0.0226      LambdaPs =    0.0000
 18. Ps( 9,14) =    0.0524      LambdaPs =    0.0000
 19. Ps(13,14) =    0.1077      LambdaPs =    0.0000
PG_total  =       2.6577 pu.
PL_total  =       2.6577 pu.
Loss      =      -0.0000 pu.
Cost      =    -239.1522 $/hour
=====================================================================
Nu = 5.005e-16
============================Constraints==============================
Binding constraints on upper limits are:
Upper Limits Number:       35       36       37       38       39       40
                           41       42       43       44       45
Upper Limits       :     PL  2    PL  3    PL  4    PL  5    PL  6    PL  9
                         PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    = 100.9649 101.6053 100.9649 100.9649 100.9649 100.9649
                        100.9649 100.9649 100.9649 100.9649 100.9649
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       34
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.8195
Slack of Lower Limits =   0.0000
=====================================================================
Expected Security Cost  =    -203.9004 $/hour
```

C.15 Standard OPF of AC IEEE 14-bus case

```
>> ESCOPF
Case name: ieee14
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
```

```
Program converges
****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration  129 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V          =   1.0800   1.0670   1.0380   1.0387   1.0500   1.0352   1.0137
               1.0269   1.0064   1.0071   1.0184   1.0219   1.0171   0.9964
Angl(rad)  =   0.0000  -0.0684  -0.2033  -0.1660  -0.1426  -0.2545  -0.2366
              -0.2366  -0.2740  -0.2763  -0.2679  -0.2705  -0.2730  -0.2917
Angl(deg)  =   0.0000  -3.9166 -11.6503  -9.5131  -8.1687 -14.5810 -13.5543
             -13.5543 -15.6989 -15.8294 -15.3516 -15.5003 -15.6411 -16.7121
PG         =   2.0123   0.7994   0.0000  -0.0000   0.0000   0.0000   0.0000
               0.0000  -0.0000  -0.0000  -0.0000  -0.0000  -0.0000   0.0000
QG         =  -0.1774   0.1671   0.3023   0.0000   0.0000   0.3974   0.0000
               0.0766  -0.0000  -0.0000  -0.0000  -0.0000  -0.0000  -0.0000
PL         =   0.0000   0.2168   0.9393   0.4739   0.0753   0.1041   0.0000
               0.0000   0.2612   0.0804   0.0319   0.0567   0.1249   0.1348
PI         =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
               0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP    =  11.3300  12.1241  13.0239  12.7583  12.5520  13.1701  12.3196
              12.3196  13.7257  13.7077  13.4871  13.4007  13.5039  13.9105
LambdaQ    =  -0.0000   0.0000   0.0000   0.0726   0.0469   0.0000   0.0000
               0.0000   0.2722   0.2443   0.1373   0.0670   0.1036   0.2672
MC_gen     =  11.3300  12.1241   0.0000   0.0000   0.0000   0.0000   0.0000
               0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load    =   0.0000  12.1241  13.0239  12.7731  12.5616  13.1701   0.0000
               0.0000  13.7809  13.7573  13.5150  13.4143  13.5250  13.9648
  1. Is( 2, 1) =    1.1920   LambdaI =    0.0000
  2. Is( 5, 1) =    0.6713   LambdaI =    0.0000
  3. Is( 1, 2) =    1.2000   LambdaI =    0.3807
  4. Is( 3, 2) =    0.7102   LambdaI =    0.0000
  5. Is( 4, 2) =    0.5727   LambdaI =    0.0000
  6. Is( 5, 2) =    0.4371   LambdaI =    0.0000
  7. Is( 2, 3) =    0.7116   LambdaI =    0.0000
  8. Is( 4, 3) =    0.2175   LambdaI =    0.0000
  9. Is( 2, 4) =    0.5747   LambdaI =    0.0000
 10. Is( 3, 4) =    0.2050   LambdaI =    0.0000
 11. Is( 5, 4) =    0.6098   LambdaI =    0.0000
 12. Is( 7, 4) =    0.3424   LambdaI =    0.0000
 13. Is( 9, 4) =    0.1953   LambdaI =    0.0000
 14. Is( 1, 5) =    0.6736   LambdaI =    0.0000
 15. Is( 2, 5) =    0.4408   LambdaI =    0.0000
 16. Is( 4, 5) =    0.6115   LambdaI =    0.0000
 17. Is( 6, 5) =    0.5000   LambdaI =    0.2739
 18. Is( 5, 6) =    0.5000   LambdaI =    0.2739
 19. Is(11, 6) =    0.0987   LambdaI =    0.0000
 20. Is(12, 6) =    0.0747   LambdaI =    0.0000
 21. Is(13, 6) =    0.1793   LambdaI =    0.0000
 22. Is( 4, 7) =    0.3424   LambdaI =    0.0000
 23. Is( 8, 7) =    0.0746   LambdaI =    0.0000
 24. Is( 9, 7) =    0.3500   LambdaI =    0.8386
 25. Is( 7, 8) =    0.0746   LambdaI =    0.0000
 26. Is( 4, 9) =    0.1953   LambdaI =    0.0000
 27. Is( 7, 9) =    0.3500   LambdaI =    0.8386
 28. Is(10, 9) =    0.0264   LambdaI =    0.0000
 29. Is(14, 9) =    0.0681   LambdaI =    0.0000
 30. Is( 9,10) =    0.0264   LambdaI =    0.0000
 31. Is(11,10) =    0.0677   LambdaI =    0.0000
 32. Is( 6,11) =    0.0987   LambdaI =    0.0000
 33. Is(10,11) =    0.0677   LambdaI =    0.0000
 34. Is( 6,12) =    0.0747   LambdaI =    0.0000
 35. Is(13,12) =    0.0180   LambdaI =    0.0000
 36. Is( 6,13) =    0.1793   LambdaI =    0.0000
 37. Is(12,13) =    0.0180   LambdaI =    0.0000
 38. Is(14,13) =    0.0722   LambdaI =    0.0000
 39. Is( 9,14) =    0.0681   LambdaI =    0.0000
```

```
 40. Is(13,14) =    0.0722   LambdaI =    0.0000
PG_total  =       2.8117 pu.
PL_total  =       2.4994 pu.
Loss      =       0.3123 pu.
Cost      =     139.6527 $/hour
==========================================================================
Nu = 5.005e-16
==================Conditions in IP=====================================
Binding constraints on upper limits are:
Upper Limits Number:        1        5       48       62       63       69
                           72
Upper Limits       :     V   1    V   5    Is  3    Is  17   Is  18   Is  24
                         Is  27
Mu of Upper Limits    =   3.5577   0.5243   0.3807   0.2739   0.2739   0.8386
                          0.8386
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000
Is( 3) is flow on line ( 1, 2).
Is(17) is flow on line ( 6, 5).
Is(18) is flow on line ( 5, 6).
Is(24) is flow on line ( 9, 7).
Is(27) is flow on line ( 7, 9).
==========================================================================
Expected Security Cost  =     139.6527 $/hour
```

C.16 ESCOPF of AC IEEE 14-bus case

```
>> ESCOPF
Case name: ieee14
DATA HAS BEEN READ IN
 FP press 0; IP press 1; IPwRho press 2 (default=  2):
 Start showing print statement when Nu is this small (default= 1e-06):
 Enter factor of Nu (default=   0.1):
 Enter the last value of Nu (default= 1e-15):
 Enter sigma (default=   1):
 Enter penalty cost for exceeding limits (default=  1000):
 Enter MAX number of iterations for OPF (default= 100):

*****************
Program converges
*****************
Print statement for MAIN
============================Variables=================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V         =   1.0800   1.0692   1.0354   1.0381   1.0500   1.0337   1.0090
              1.0166   1.0030   1.0040   1.0161   1.0202   1.0154   0.9937
Angl(rad) =   0.0000  -0.0483  -0.1783  -0.1462  -0.1252  -0.2368  -0.2172
             -0.2172  -0.2551  -0.2576  -0.2497  -0.2528  -0.2552  -0.2734
Angl(deg) =   0.0000  -2.7697 -10.2185  -8.3767  -7.1736 -13.5685 -12.4473
            -12.4473 -14.6136 -14.7586 -14.3087 -14.4853 -14.6198 -15.6623
PG        =   1.5589   1.1500   0.0000  -0.0000   0.0000  -0.0000   0.0000
             -0.0000  -0.0000   0.0000   0.0000  -0.0000   0.0000   0.0000
QG        =  -0.1100   0.1170   0.2462   0.0000   0.0000   0.3982   0.0000
              0.0438  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
              0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2557  11.5830  12.4372  12.2336  12.0720  12.2168  11.9354
```

```
                11.9354   12.7240   12.7095   12.5137   12.4315   12.5243   12.8965
LambdaQ     =  -0.0000    0.0000    0.0000    0.0068   -0.0608    0.0000    0.0000
                0.0000    0.1447    0.1396    0.0872    0.0540    0.0812    0.1806
MC_gen      =  11.2557   12.1785    0.0000    0.0000    0.0000    0.0000    0.0000
                0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
MB_load     =   0.0000   12.5520   13.4921   13.2603   13.0524   13.1962    0.0000
                0.0000   13.7447   13.7327   13.5217   13.4376   13.5416   13.9489
  1. Is( 2, 1) =    0.8488   LambdaI =    0.0000
  2. Is( 5, 1) =    0.5936   LambdaI =    0.0000
  3. Is( 1, 2) =    0.8551   LambdaI =    0.0000
  4. Is( 3, 2) =    0.6911   LambdaI =    0.0000
  5. Is( 4, 2) =    0.5787   LambdaI =    0.0000
  6. Is( 5, 2) =    0.4553   LambdaI =    0.0000
  7. Is( 2, 3) =    0.6907   LambdaI =    0.0000
  8. Is( 4, 3) =    0.1877   LambdaI =    0.0000
  9. Is( 2, 4) =    0.5797   LambdaI =    0.0000
 10. Is( 3, 4) =    0.1775   LambdaI =    0.0000
 11. Is( 5, 4) =    0.5631   LambdaI =    0.0000
 12. Is( 7, 4) =    0.3451   LambdaI =    0.0000
 13. Is( 9, 4) =    0.1966   LambdaI =    0.0000
 14. Is( 1, 5) =    0.5945   LambdaI =    0.0000
 15. Is( 2, 5) =    0.4584   LambdaI =    0.0000
 16. Is( 4, 5) =    0.5656   LambdaI =    0.0000
 17. Is( 6, 5) =    0.4960   LambdaI =    0.0000
 18. Is( 5, 6) =    0.4960   LambdaI =    0.0000
 19. Is(11, 6) =    0.1000   LambdaI =    0.0081
 20. Is(12, 6) =    0.0749   LambdaI =    0.0000
 21. Is(13, 6) =    0.1798   LambdaI =    0.0000
 22. Is( 4, 7) =    0.3451   LambdaI =    0.0000
 23. Is( 8, 7) =    0.0431   LambdaI =    0.0000
 24. Is( 9, 7) =    0.3500   LambdaI =    0.4794
 25. Is( 7, 8) =    0.0431   LambdaI =    0.0000
 26. Is( 4, 9) =    0.1966   LambdaI =    0.0000
 27. Is( 7, 9) =    0.3500   LambdaI =    0.4794
 28. Is(10, 9) =    0.0301   LambdaI =    0.0000
 29. Is(14, 9) =    0.0687   LambdaI =    0.0000
 30. Is( 9,10) =    0.0301   LambdaI =    0.0000
 31. Is(11,10) =    0.0694   LambdaI =    0.0000
 32. Is( 6,11) =    0.1000   LambdaI =    0.0081
 33. Is(10,11) =    0.0694   LambdaI =    0.0000
 34. Is( 6,12) =    0.0749   LambdaI =    0.0000
 35. Is(13,12) =    0.0182   LambdaI =    0.0000
 36. Is( 6,13) =    0.1798   LambdaI =    0.0000
 37. Is(12,13) =    0.0182   LambdaI =    0.0000
 38. Is(14,13) =    0.0732   LambdaI =    0.0000
 39. Is( 9,14) =    0.0687   LambdaI =    0.0000
 40. Is(13,14) =    0.0732   LambdaI =    0.0000
PG_total  =       2.7089 pu.
PL_total  =       2.4183 pu.
Loss      =       0.2905 pu.
Cost      =     159.3057 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP==================================
Binding constraints on upper limits are:
Upper Limits Number:        1        5       64       69       72       77
Upper Limits       :     V   1    V   5    Is  19   Is  24   Is  27   Is  32
Mu of Upper Limits    =   0.8309   1.7146   0.0081   0.4794   0.4794   0.0081
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
Is(19) is flow on line (11, 6).
Is(24) is flow on line ( 9, 7).
Is(27) is flow on line ( 7, 9).
Is(32) is flow on line ( 6,11).
================================================================================
Print statement for SUBPROBLEM   1 which is line ( 1, 2).
============================Variables===================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
```

```
                    8         9        10        11        12        13        14
V           =    1.0800    1.0644    1.0332    1.0415    1.0500    1.0336    1.0366
                 1.0795    1.0212    1.0190    1.0237    1.0217    1.0181    1.0055
Angl(rad)   =    0.0000   -0.2001   -0.3107   -0.2684   -0.2358   -0.3479   -0.3383
                -0.3383   -0.3750   -0.3760   -0.3648   -0.3643   -0.3676   -0.3898
Angl(deg)   =    0.0000  -11.4653  -17.8000  -15.3801  -13.5129  -19.9316  -19.3850
               -19.3850  -21.4850  -21.5456  -20.9000  -20.8746  -21.0636  -22.3337
PG          =    1.1876    1.5000    0.0000   -0.0000    0.0000    0.0000   -0.0000
                 0.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000
QG          =   -0.0304    0.0561    0.2076    0.0000    0.0000    0.3274   -0.0000
                 0.2631   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000
PL          =    0.0000    0.2068    0.8545    0.4500    0.0714    0.1038    0.0000
                 0.0000    0.2623    0.0807    0.0319    0.0565    0.1247    0.1350
PI          =    0.0000    0.0000    0.0406    0.0000    0.0000    0.0000    0.0000
                 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
LambdaP     =   11.1948  106.9718  113.9224  111.2565  109.0185  108.7752  111.3661
               111.3661  111.4267  111.5866  110.5714  110.5389  111.2297  113.5483
LambdaQ     =   -0.0000    0.0000    0.0000   -0.0611   -0.1703    0.0000    0.0000
                 0.0000    0.0258    0.1544    0.1656    0.3916    0.4965    0.7371
MC_gen      =   11.1948   12.2328    0.0000    0.0000    0.0000    0.0000    0.0000
                 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
MB_load     =    0.0000   12.5520   13.9224   13.2603   13.0524   13.1962    0.0000
                 0.0000   13.7447   13.7327   13.5217   13.4376   13.5416   13.9489
  1. Is( 5, 1) =    1.0939   LambdaI =    0.0000
  2. Is( 3, 2) =    0.5897   LambdaI =    0.0000
  3. Is( 4, 2) =    0.4063   LambdaI =    0.0000
  4. Is( 5, 2) =    0.2223   LambdaI =    0.0000
  5. Is( 2, 3) =    0.5883   LambdaI =    0.0000
  6. Is( 4, 3) =    0.2463   LambdaI =    0.0000
  7. Is( 2, 4) =    0.4067   LambdaI =    0.0000
  8. Is( 3, 4) =    0.2396   LambdaI =    0.0000
  9. Is( 5, 4) =    0.7956   LambdaI =    0.0000
 10. Is( 7, 4) =    0.3540   LambdaI =    0.0000
 11. Is( 9, 4) =    0.1956   LambdaI =    0.0000
 12. Is( 1, 5) =    1.1000   LambdaI =   92.8051
 13. Is( 2, 5) =    0.2206   LambdaI =    0.0000
 14. Is( 4, 5) =    0.7948   LambdaI =    0.0000
 15. Is( 6, 5) =    0.4973   LambdaI =    0.0000
 16. Is( 5, 6) =    0.4973   LambdaI =    0.0000
 17. Is(11, 6) =    0.0908   LambdaI =    0.0000
 18. Is(12, 6) =    0.0728   LambdaI =    0.0000
 19. Is(13, 6) =    0.1747   LambdaI =    0.0000
 20. Is( 4, 7) =    0.3540   LambdaI =    0.0000
 21. Is( 8, 7) =    0.2437   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3700   LambdaI =    0.0000
 23. Is( 7, 8) =    0.2437   LambdaI =    0.0000
 24. Is( 4, 9) =    0.1956   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3700   LambdaI =    0.0000
 26. Is(10, 9) =    0.0271   LambdaI =    0.0000
 27. Is(14, 9) =    0.0728   LambdaI =    0.0000
 28. Is( 9,10) =    0.0271   LambdaI =    0.0000
 29. Is(11,10) =    0.0594   LambdaI =    0.0000
 30. Is( 6,11) =    0.0908   LambdaI =    0.0000
 31. Is(10,11) =    0.0594   LambdaI =    0.0000
 32. Is( 6,12) =    0.0728   LambdaI =    0.0000
 33. Is(13,12) =    0.0166   LambdaI =    0.0000
 34. Is( 6,13) =    0.1747   LambdaI =    0.0000
 35. Is(12,13) =    0.0166   LambdaI =    0.0000
 36. Is(14,13) =    0.0663   LambdaI =    0.0000
 37. Is( 9,14) =    0.0728   LambdaI =    0.0000
 38. Is(13,14) =    0.0663   LambdaI =    0.0000
PG_total    =       2.6876 pu.
PL_total    =       2.3777 pu.
Loss        =       0.3099 pu.
Cost        =     631.3073 $/hour
======================================================================
Nu = 5.005e-16
==================Conditions in IP====================================
```

```
Binding constraints on upper limits are:
Upper Limits Number:        1        5       29       57       85       86
                           88       89       90       91       92       93
                           94       95       96
Upper Limits       :     V   1    V   5    PG  2    Is  14   PG  2    PL  2
                         PL  4    PL  5    PL  6    PL  9    PL  10   PL  11
                         PL  12   PL  13   PL  14
Mu of Upper Limits    = 116.8633   0.0803  37.5783  92.8051  57.1607   5.5803
                          2.0162   4.0685   4.4210   2.3128   2.1148   2.9167
                          2.8192   2.2111   0.2510
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000
Is(14) is flow on line ( 1, 5).
================================================================================
Print statement for SUBPROBLEM   2 which is line ( 2, 3).
============================Variables===================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V          =   1.0800   1.0786   0.9941   1.0360   1.0500   1.0504   1.0370
               1.0800   1.0244   1.0246   1.0348   1.0377   1.0331   1.0140
Angl(rad)  =   0.0000  -0.0626  -0.3769  -0.2155  -0.1764  -0.2918  -0.2833
              -0.2833  -0.3186  -0.3194  -0.3082  -0.3079  -0.3109  -0.3329
Angl(deg)  =   0.0000  -3.5885 -21.5925 -12.3456 -10.1075 -16.7161 -16.2317
             -16.2317 -18.2525 -18.3022 -17.6564 -17.6409 -17.8135 -19.0727
PG         =   2.0261   0.8000   0.0000  -0.0000   0.0000  -0.0000  -0.0000
               0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000  -0.0000
QG         =  -0.3578   0.3390   0.3499   0.0000   0.0000   0.4443  -0.0000
               0.2638  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL         =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
               0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI         =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
               0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP    =  11.3323  11.7495  14.6916  12.9458  12.5844  12.5867  12.9546
              12.9546  12.9592  12.9661  12.8207  12.7891  12.8741  13.1759
LambdaQ    =  -0.0000   0.0000   0.0000   0.0746   0.0141   0.0000   0.0524
               0.0000   0.0749   0.0798   0.0531   0.0463   0.0683   0.1313
MC_gen     =  11.3323  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
               0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load    =   0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
               0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    1.0769   LambdaI =    0.0000
  2. Is( 5, 1) =    0.8240   LambdaI =    0.0000
  3. Is( 1, 2) =    1.0935   LambdaI =    0.0000
  4. Is( 4, 2) =    0.8966   LambdaI =    0.0000
  5. Is( 5, 2) =    0.6773   LambdaI =    0.0000
  6. Is( 4, 3) =    0.9204   LambdaI =    0.0000
  7. Is( 2, 4) =    0.8990   LambdaI =    0.0000
  8. Is( 3, 4) =    0.9161   LambdaI =    0.0000
  9. Is( 5, 4) =    0.9746   LambdaI =    0.0000
 10. Is( 7, 4) =    0.3512   LambdaI =    0.0000
 11. Is( 9, 4) =    0.1915   LambdaI =    0.0000
 12. Is( 1, 5) =    0.8281   LambdaI =    0.0000
 13. Is( 2, 5) =    0.6803   LambdaI =    0.0000
 14. Is( 4, 5) =    0.9749   LambdaI =    0.0000
 15. Is( 6, 5) =    0.5442   LambdaI =    0.0000
 16. Is( 5, 6) =    0.5442   LambdaI =    0.0000
 17. Is(11, 6) =    0.1051   LambdaI =    0.0000
 18. Is(12, 6) =    0.0743   LambdaI =    0.0000
 19. Is(13, 6) =    0.1807   LambdaI =    0.0000
 20. Is( 4, 7) =    0.3512   LambdaI =    0.0000
 21. Is( 8, 7) =    0.2443   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3495   LambdaI =    0.0000
 23. Is( 7, 8) =    0.2443   LambdaI =    0.0000
 24. Is( 4, 9) =    0.1915   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3495   LambdaI =    0.0000
 26. Is(10, 9) =    0.0100   LambdaI =    0.0000
```

```
 27. Is(14, 9) =    0.0601   LambdaI =    0.0000
 28. Is( 9,10) =    0.0100   LambdaI =    0.0000
 29. Is(11,10) =    0.0737   LambdaI =    0.0000
 30. Is( 6,11) =    0.1051   LambdaI =    0.0000
 31. Is(10,11) =    0.0737   LambdaI =    0.0000
 32. Is( 6,12) =    0.0743   LambdaI =    0.0000
 33. Is(13,12) =    0.0187   LambdaI =    0.0000
 34. Is( 6,13) =    0.1807   LambdaI =    0.0000
 35. Is(12,13) =    0.0187   LambdaI =    0.0000
 36. Is(14,13) =    0.0761   LambdaI =    0.0000
 37. Is( 9,14) =    0.0601   LambdaI =    0.0000
 38. Is(13,14) =    0.0761   LambdaI =    0.0000
PG_total  =       2.8261 pu.
PL_total  =       2.4183 pu.
Loss      =       0.4078 pu.
Cost      =     261.6795 $/hour
======================================================================
Nu = 5.005e-16
==================Conditions in IP====================================
Binding constraints on upper limits are:
Upper Limits Number:        1        5        8       86       87       88
                           89       90       91       92       93       94
                           95       96
Upper Limits       :     V  1    V  5    V  8    PL  2    PL  3    PL  4
                         PL  5    PL  6    PL  9    PL  10   PL  11   PL  12
                         PL  13   PL  14
Mu of Upper Limits    =   3.5604   1.3868   0.3084 100.8025  98.8005 100.2993
                        100.4651 100.6095 100.7703 100.7504 100.6902 100.6392
                        100.6536 100.7463
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
----------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       85
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.3746
Slack of Lower Limits =   0.0000
======================================================================
Print statement for SUBPROBLEM   3 which is line ( 2, 4).
============================Variables=================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V         =   1.0800   1.0741   1.0423   1.0389   1.0500   1.0385   1.0363
              1.0800   1.0216   1.0202   1.0266   1.0263   1.0223   1.0076
Angl(rad) =   0.0000  -0.0581  -0.2184  -0.2135  -0.1739  -0.2887  -0.2820
             -0.2820  -0.3178  -0.3184  -0.3063  -0.3052  -0.3085  -0.3317
Angl(deg) =   0.0000  -3.3273 -12.5110 -12.2340  -9.9614 -16.5389 -16.1570
            -16.1570 -18.2083 -18.2443 -17.5494 -17.4853 -17.6777 -19.0040
PG        =   1.9510   0.8000   0.0000  -0.0000   0.0000  -0.0000   0.0000
              0.0000  -0.0000  -0.0000   0.0000   0.0000  -0.0000   0.0000
QG        =  -0.2599   0.2162   0.2809   0.0000   0.0000   0.3608  -0.0000
              0.2678  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
              0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3200  11.7231  12.7402  12.7666  12.4625  12.4422  12.7771
             12.7771  12.7832  12.7954  12.6638  12.6442  12.7270  13.0112
LambdaQ   =  -0.0000   0.0000   0.0000  -0.0018  -0.0058   0.0000   0.0069
              0.0000   0.0153   0.0288   0.0256   0.0447   0.0590   0.0926
MC_gen    =  11.3200  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
              0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    1.0023   LambdaI =    0.0000
  2. Is( 5, 1) =    0.8124   LambdaI =    0.0000
```

```
  3. Is( 1, 2) =    1.0148   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8445   LambdaI =    0.0000
  5. Is( 5, 2) =    0.6814   LambdaI =    0.0000
  6. Is( 2, 3) =    0.8467   LambdaI =    0.0000
  7. Is( 4, 3) =    0.0490   LambdaI =    0.0000
  8. Is( 3, 4) =    0.0209   LambdaI =    0.0000
  9. Is( 5, 4) =    0.9710   LambdaI =    0.0000
 10. Is( 7, 4) =    0.3496   LambdaI =    0.0000
 11. Is( 9, 4) =    0.1919   LambdaI =    0.0000
 12. Is( 1, 5) =    0.8164   LambdaI =    0.0000
 13. Is( 2, 5) =    0.6858   LambdaI =    0.0000
 14. Is( 4, 5) =    0.9704   LambdaI =    0.0000
 15. Is( 6, 5) =    0.5168   LambdaI =    0.0000
 16. Is( 5, 6) =    0.5168   LambdaI =    0.0000
 17. Is(11, 6) =    0.0985   LambdaI =    0.0000
 18. Is(12, 6) =    0.0738   LambdaI =    0.0000
 19. Is(13, 6) =    0.1785   LambdaI =    0.0000
 20. Is( 4, 7) =    0.3496   LambdaI =    0.0000
 21. Is( 8, 7) =    0.2480   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3605   LambdaI =    0.0000
 23. Is( 7, 8) =    0.2480   LambdaI =    0.0000
 24. Is( 4, 9) =    0.1919   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3605   LambdaI =    0.0000
 26. Is(10, 9) =    0.0174   LambdaI =    0.0000
 27. Is(14, 9) =    0.0665   LambdaI =    0.0000
 28. Is( 9,10) =    0.0174   LambdaI =    0.0000
 29. Is(11,10) =    0.0670   LambdaI =    0.0000
 30. Is( 6,11) =    0.0985   LambdaI =    0.0000
 31. Is(10,11) =    0.0670   LambdaI =    0.0000
 32. Is( 6,12) =    0.0738   LambdaI =    0.0000
 33. Is(13,12) =    0.0177   LambdaI =    0.0000
 34. Is( 6,13) =    0.1785   LambdaI =    0.0000
 35. Is(12,13) =    0.0177   LambdaI =    0.0000
 36. Is(14,13) =    0.0715   LambdaI =    0.0000
 37. Is( 9,14) =    0.0665   LambdaI =    0.0000
 38. Is(13,14) =    0.0715   LambdaI =    0.0000
PG_total  =       2.7510 pu.
PL_total  =       2.4183 pu.
Loss      =       0.3327 pu.
Cost      =     176.6256 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP==================================
Binding constraints on upper limits are:
Upper Limits Number:        1        5        8       86       87       88
                           89       90       91       92       93       94
                           95       96
Upper Limits       :     V   1    V   5    V   8    PL  2    PL  3    PL  4
                         PL  5    PL  6    PL  9    PL  10   PL  11   PL  12
                         PL  13   PL  14
Mu of Upper Limits    =   2.6696   0.4143   0.0403 100.8289 100.7519 100.4940
                        100.5911 100.7540 100.9583 100.9315 100.8528 100.7843
                        100.8026 100.9189
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
--------------------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       85
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.4010
Slack of Lower Limits =   0.0000
================================================================================
Print statement for SUBPROBLEM   4 which is line ( 3, 4).
============================Variables=====================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
```

```
V         =   1.0800   1.0674   1.0409   1.0402   1.0500   1.0184   1.0250
              1.0580   1.0105   1.0076   1.0105   1.0066   1.0034   0.9929
Angl(rad) =   0.0000  -0.0674  -0.2340  -0.1477  -0.1285  -0.2366  -0.2206
             -0.2206  -0.2592  -0.2613  -0.2519  -0.2531  -0.2562  -0.2764
Angl(deg) =   0.0000  -3.8620 -13.4086  -8.4623  -7.3638 -13.5573 -12.6411
            -12.6411 -14.8528 -14.9694 -14.4301 -14.5042 -14.6813 -15.8371
PG        =   1.9244   0.8000  -0.0000  -0.0000   0.0000   0.0000   0.0000
             -0.0000  -0.0000   0.0000  -0.0000  -0.0000  -0.0000  -0.0000
QG        =  -0.1730   0.1776   0.3085   0.0000   0.0000   0.2535   0.0000
              0.1985  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
              0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3156  11.7859  12.7659  12.3670  12.1900  12.1559  12.3827
             12.3827  12.3909  12.4197  12.3310  12.3554  12.4271  12.6583
LambdaQ   =  -0.0000   0.0000   0.0000  -0.0009  -0.0293   0.0000   0.0000
              0.0000  -0.0005   0.0140   0.0161   0.0450   0.0557   0.0817
MC_gen    =  11.3156  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
              0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    1.1748   LambdaI =    0.0000
  2. Is( 5, 1) =    0.6084   LambdaI =    0.0000
  3. Is( 1, 2) =    1.1830   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8685   LambdaI =    0.0000
  5. Is( 4, 2) =    0.4781   LambdaI =    0.0000
  6. Is( 5, 2) =    0.3651   LambdaI =    0.0000
  7. Is( 2, 3) =    0.8723   LambdaI =    0.0000
  8. Is( 2, 4) =    0.4784   LambdaI =    0.0000
  9. Is( 5, 4) =    0.5036   LambdaI =    0.0000
 10. Is( 7, 4) =    0.3579   LambdaI =    0.0000
 11. Is( 9, 4) =    0.2023   LambdaI =    0.0000
 12. Is( 1, 5) =    0.6096   LambdaI =    0.0000
 13. Is( 2, 5) =    0.3671   LambdaI =    0.0000
 14. Is( 4, 5) =    0.5055   LambdaI =    0.0000
 15. Is( 6, 5) =    0.4562   LambdaI =    0.0000
 16. Is( 5, 6) =    0.4562   LambdaI =    0.0000
 17. Is(11, 6) =    0.0788   LambdaI =    0.0000
 18. Is(12, 6) =    0.0721   LambdaI =    0.0000
 19. Is(13, 6) =    0.1702   LambdaI =    0.0000
 20. Is( 4, 7) =    0.3579   LambdaI =    0.0000
 21. Is( 8, 7) =    0.1876   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3805   LambdaI =    0.0000
 23. Is( 7, 8) =    0.1876   LambdaI =    0.0000
 24. Is( 4, 9) =    0.2023   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3805   LambdaI =    0.0000
 26. Is(10, 9) =    0.0399   LambdaI =    0.0000
 27. Is(14, 9) =    0.0824   LambdaI =    0.0000
 28. Is( 9,10) =    0.0399   LambdaI =    0.0000
 29. Is(11,10) =    0.0475   LambdaI =    0.0000
 30. Is( 6,11) =    0.0788   LambdaI =    0.0000
 31. Is(10,11) =    0.0475   LambdaI =    0.0000
 32. Is( 6,12) =    0.0721   LambdaI =    0.0000
 33. Is(13,12) =    0.0150   LambdaI =    0.0000
 34. Is( 6,13) =    0.1702   LambdaI =    0.0000
 35. Is(12,13) =    0.0150   LambdaI =    0.0000
 36. Is(14,13) =    0.0585   LambdaI =    0.0000
 37. Is( 9,14) =    0.0824   LambdaI =    0.0000
 38. Is(13,14) =    0.0585   LambdaI =    0.0000
PG_total  =       2.7244 pu.
PL_total  =       2.4183 pu.
Loss      =       0.3061 pu.
Cost      =     146.5188 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP================================
Binding constraints on upper limits are:
```

```
Upper Limits Number:        1        5       86       87       88       89
                           90       91       92       93       94       95
                           96
Upper Limits       :     V   1    V   5    PL  2    PL  3    PL  4    PL  5
                         PL  6    PL  9    PL  10   PL  11   PL  12   PL  13
                         PL  14
Mu of Upper Limits    =   1.7350   0.8238 100.7662 100.7262 100.8934 100.8683
                        101.0403 101.3538 101.3101 101.1874 101.0731 101.1032
                        101.2741
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000
---------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       85
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.3383
Slack of Lower Limits =   0.0000
=====================================================================
Print statement for SUBPROBLEM   5 which is line ( 1, 5).
============================Variables================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V         =   1.0800   1.0694   1.0382   1.0362   1.0418   1.0232   1.0333
              1.0800   1.0169   1.0137   1.0160   1.0116   1.0085   0.9988
Angl(rad) =   0.0000  -0.0890  -0.2407  -0.2270  -0.2165  -0.3218  -0.3016
             -0.3016  -0.3407  -0.3434  -0.3354  -0.3379  -0.3407  -0.3590
Angl(deg) =   0.0000  -5.0995 -13.7922 -13.0049 -12.4067 -18.4375 -17.2820
            -17.2820 -19.5229 -19.6730 -19.2197 -19.3605 -19.5229 -20.5695
PG        =   1.6471   1.1195  -0.0000   0.0000  -0.0000  -0.0000  -0.0000
              0.0000  -0.0000   0.0000  -0.0000   0.0000  -0.0000   0.0000
QG        =  -0.2991   0.3227   0.2771   0.0000   0.0000   0.3030   0.0000
              0.2864  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
              0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.2701  12.1737  13.2503  13.2554  13.1676  13.1122  13.2824
             13.2824  13.2967  13.3379  13.2698  13.3215  13.3914  13.6048
LambdaQ   =  -0.0000   0.0000   0.0000  -0.0007  -0.0213   0.0000   0.0025
              0.0000   0.0042   0.0188   0.0188   0.0477   0.0597   0.0897
MC_gen    =  11.2701  12.1737   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
              0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    1.5384   LambdaI =    0.0000
  2. Is( 1, 2) =    1.5500   LambdaI =    0.2235
  3. Is( 3, 2) =    0.7974   LambdaI =    0.0000
  4. Is( 4, 2) =    0.7990   LambdaI =    0.0000
  5. Is( 5, 2) =    0.7475   LambdaI =    0.0000
  6. Is( 2, 3) =    0.7992   LambdaI =    0.0000
  7. Is( 4, 3) =    0.0886   LambdaI =    0.0000
  8. Is( 2, 4) =    0.8027   LambdaI =    0.0000
  9. Is( 3, 4) =    0.0711   LambdaI =    0.0000
 10. Is( 5, 4) =    0.2757   LambdaI =    0.0000
 11. Is( 7, 4) =    0.3774   LambdaI =    0.0000
 12. Is( 9, 4) =    0.2078   LambdaI =    0.0000
 13. Is( 2, 5) =    0.7516   LambdaI =    0.0000
 14. Is( 4, 5) =    0.2780   LambdaI =    0.0000
 15. Is( 6, 5) =    0.4648   LambdaI =    0.0000
 16. Is( 5, 6) =    0.4648   LambdaI =    0.0000
 17. Is(11, 6) =    0.0711   LambdaI =    0.0000
 18. Is(12, 6) =    0.0708   LambdaI =    0.0000
 19. Is(13, 6) =    0.1656   LambdaI =    0.0000
 20. Is( 4, 7) =    0.3774   LambdaI =    0.0000
 21. Is( 8, 7) =    0.2652   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3938   LambdaI =    0.0000
```

```
 23. Is( 7, 8) =    0.2652   LambdaI =    0.0000
 24. Is( 4, 9) =    0.2078   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3938   LambdaI =    0.0000
 26. Is(10, 9) =    0.0457   LambdaI =    0.0000
 27. Is(14, 9) =    0.0862   LambdaI =    0.0000
 28. Is( 9,10) =    0.0457   LambdaI =    0.0000
 29. Is(11,10) =    0.0400   LambdaI =    0.0000
 30. Is( 6,11) =    0.0711   LambdaI =    0.0000
 31. Is(10,11) =    0.0400   LambdaI =    0.0000
 32. Is( 6,12) =    0.0708   LambdaI =    0.0000
 33. Is(13,12) =    0.0140   LambdaI =    0.0000
 34. Is( 6,13) =    0.1656   LambdaI =    0.0000
 35. Is(12,13) =    0.0140   LambdaI =    0.0000
 36. Is(14,13) =    0.0535   LambdaI =    0.0000
 37. Is( 9,14) =    0.0862   LambdaI =    0.0000
 38. Is(13,14) =    0.0535   LambdaI =    0.0000
PG_total  =       2.7666 pu.
PL_total  =       2.4183 pu.
Loss      =       0.3482 pu.
Cost      =     221.5087 $/hour
================================================================================
Nu = 5.005e-16
==================Conditions in IP================================
Binding constraints on upper limits are:
Upper Limits Number:        1        8       47       86       87       88
                           89       90       91       92       93       94
                           95       96
Upper Limits       :     V   1    V   8    Is  3    PL  2    PL  3    PL  4
                         PL  5    PL  6    PL  9    PL  10   PL  11   PL  12
                         PL  13   PL  14
Mu of Upper Limits    =   3.9575   0.0147   0.2235 100.3783 100.2419 100.0050
                         99.8891 100.0840 100.4471 100.3910 100.2481 100.1065
                        100.1381 100.3259
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
Is( 3) is flow on line ( 1, 2).
================================================================================
Print statement for SUBPROBLEM   6 which is line ( 2, 5).
============================Variables===================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V         =   1.0800   1.0734   1.0426   1.0430   1.0500   1.0316   1.0375
              1.0800   1.0223   1.0197   1.0232   1.0199   1.0167   1.0055
Angl(rad) =   0.0000  -0.0574  -0.2032  -0.1861  -0.1734  -0.2785  -0.2594
             -0.2594  -0.2979  -0.3003  -0.2922  -0.2945  -0.2973  -0.3156
Angl(deg) =   0.0000  -3.2895 -11.6423 -10.6644  -9.9346 -15.9592 -14.8633
            -14.8633 -17.0660 -17.2064 -16.7430 -16.8720 -17.0318 -18.0835
PG        =   1.9396   0.8000   0.0000  -0.0000   0.0000  -0.0000   0.0000
             -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000  -0.0000
QG        =  -0.2459   0.2208   0.2621   0.0000   0.0000   0.3168   0.0000
              0.2606  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =   0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
              0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   =  11.3181  11.7199  12.6582  12.5761  12.4402  12.4034  12.5950
             12.5950  12.6051  12.6389  12.5633  12.6001  12.6679  12.8822
LambdaQ   =  -0.0000   0.0000   0.0000   0.0040  -0.0123   0.0000   0.0068
              0.0000   0.0117   0.0244   0.0216   0.0445   0.0571   0.0890
MC_gen    =  11.3181  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
              0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =   0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
              0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    0.9920   LambdaI =    0.0000
  2. Is( 5, 1) =    0.8103   LambdaI =    0.0000
  3. Is( 1, 2) =    1.0039   LambdaI =    0.0000
```

```
 4. Is( 3, 2) =    0.7701   LambdaI =    0.0000
 5. Is( 4, 2) =    0.7482   LambdaI =    0.0000
 6. Is( 2, 3) =    0.7718   LambdaI =    0.0000
 7. Is( 4, 3) =    0.1043   LambdaI =    0.0000
 8. Is( 2, 4) =    0.7521   LambdaI =    0.0000
 9. Is( 3, 4) =    0.0922   LambdaI =    0.0000
10. Is( 5, 4) =    0.3392   LambdaI =    0.0000
11. Is( 7, 4) =    0.3699   LambdaI =    0.0000
12. Is( 9, 4) =    0.2052   LambdaI =    0.0000
13. Is( 1, 5) =    0.8143   LambdaI =    0.0000
14. Is( 4, 5) =    0.3415   LambdaI =    0.0000
15. Is( 6, 5) =    0.4689   LambdaI =    0.0000
16. Is( 5, 6) =    0.4689   LambdaI =    0.0000
17. Is(11, 6) =    0.0744   LambdaI =    0.0000
18. Is(12, 6) =    0.0709   LambdaI =    0.0000
19. Is(13, 6) =    0.1666   LambdaI =    0.0000
20. Is( 4, 7) =    0.3699   LambdaI =    0.0000
21. Is( 8, 7) =    0.2413   LambdaI =    0.0000
22. Is( 9, 7) =    0.3855   LambdaI =    0.0000
23. Is( 7, 8) =    0.2413   LambdaI =    0.0000
24. Is( 4, 9) =    0.2052   LambdaI =    0.0000
25. Is( 7, 9) =    0.3855   LambdaI =    0.0000
26. Is(10, 9) =    0.0401   LambdaI =    0.0000
27. Is(14, 9) =    0.0824   LambdaI =    0.0000
28. Is( 9,10) =    0.0401   LambdaI =    0.0000
29. Is(11,10) =    0.0430   LambdaI =    0.0000
30. Is( 6,11) =    0.0744   LambdaI =    0.0000
31. Is(10,11) =    0.0430   LambdaI =    0.0000
32. Is( 6,12) =    0.0709   LambdaI =    0.0000
33. Is(13,12) =    0.0145   LambdaI =    0.0000
34. Is( 6,13) =    0.1666   LambdaI =    0.0000
35. Is(12,13) =    0.0145   LambdaI =    0.0000
36. Is(14,13) =    0.0558   LambdaI =    0.0000
37. Is( 9,14) =    0.0824   LambdaI =    0.0000
38. Is(13,14) =    0.0558   LambdaI =    0.0000
PG_total  =       2.7396 pu.
PL_total  =       2.4183 pu.
Loss      =       0.3213 pu.
Cost      =     163.7770 $/hour
=============================================================================
Nu = 5.005e-16
==================Conditions in IP===================================
Binding constraints on upper limits are:
Upper Limits Number:        1        5        8       86       87       88
                           89       90       91       92       93       94
                           95       96
Upper Limits       :     V   1    V   5    V   8    PL  2    PL  3    PL  4
                         PL  5    PL  6    PL  9    PL  10   PL  11   PL  12
                         PL  13   PL  14
Mu of Upper Limits    =   2.5230   0.2749   0.0401 100.8322 100.8339 100.6833
                        100.6147 100.7928 101.1371 101.0889 100.9541 100.8285
                        100.8621 101.0486
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000
-----------------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       85
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.4043
Slack of Lower Limits =   0.0000
=============================================================================
Print statement for SUBPROBLEM   7 which is line ( 4, 5).
============================Variables====================================
Calculated values at the end of iteration   96 are:
Bus number       1        2        3        4        5        6        7
                 8        9       10       11       12       13       14
V         =   1.0761   1.0551   1.0182   1.0046   1.0500   0.9919   0.9942
```

```
             1.0356   0.9757   0.9733   0.9791   0.9795   0.9746   0.9598
Angl(rad) =  0.0000  -0.0753  -0.2399  -0.2293  -0.0990  -0.2471  -0.2868
            -0.2868  -0.3173  -0.3113  -0.2825  -0.2679  -0.2743  -0.3183
Angl(deg) =  0.0000  -4.3135 -13.7453 -13.1396  -5.6708 -14.1602 -16.4346
           -16.4346 -18.1792 -17.8366 -16.1874 -15.3498 -15.7190 -18.2395
PG        =  1.9397   0.8000   0.0000  -0.0000   0.0000  -0.0000   0.0000
            -0.0000  -0.0000   0.0000  -0.0000   0.0000  -0.0000   0.0000
QG        = -0.0642   0.1886   0.3230   0.0000   0.0000   0.1382  -0.0000
             0.2437  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
PL        =  0.0000   0.2068   0.8951   0.4500   0.0714   0.1038   0.0000
             0.0000   0.2623   0.0807   0.0319   0.0565   0.1247   0.1350
PI        =  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
             0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
LambdaP   = 11.3181  11.8571  12.9529  12.9873  11.9227  11.9800  12.9439
            12.9439  12.9177  12.8407  12.4712  12.2253  12.3728  12.9608
LambdaQ   = -0.0000   0.0000   0.0000   0.0707  -0.2227   0.0000   0.0000
             0.0000  -0.0353  -0.0099   0.0125   0.0562   0.0620   0.0746
MC_gen    = 11.3181  12.1242   0.0000   0.0000   0.0000   0.0000   0.0000
             0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
MB_load   =  0.0000  12.5520  13.4921  13.2603  13.0524  13.1962   0.0000
             0.0000  13.7447  13.7327  13.5217  13.4376  13.5416  13.9489
  1. Is( 2, 1) =    1.3291   LambdaI =    0.0000
  2. Is( 5, 1) =    0.4717   LambdaI =    0.0000
  3. Is( 1, 2) =    1.3325   LambdaI =    0.0000
  4. Is( 3, 2) =    0.8549   LambdaI =    0.0000
  5. Is( 4, 2) =    0.8942   LambdaI =    0.0000
  6. Is( 5, 2) =    0.1379   LambdaI =    0.0000
  7. Is( 2, 3) =    0.8558   LambdaI =    0.0000
  8. Is( 4, 3) =    0.1110   LambdaI =    0.0000
  9. Is( 2, 4) =    0.8946   LambdaI =    0.0000
 10. Is( 3, 4) =    0.0775   LambdaI =    0.0000
 11. Is( 7, 4) =    0.2774   LambdaI =    0.0000
 12. Is( 9, 4) =    0.1541   LambdaI =    0.0000
 13. Is( 1, 5) =    0.4714   LambdaI =    0.0000
 14. Is( 2, 5) =    0.1420   LambdaI =    0.0000
 15. Is( 6, 5) =    0.5811   LambdaI =    0.0000
 16. Is( 5, 6) =    0.5811   LambdaI =    0.0000
 17. Is(11, 6) =    0.1685   LambdaI =    0.0000
 18. Is(12, 6) =    0.0844   LambdaI =    0.0000
 19. Is(13, 6) =    0.2181   LambdaI =    0.0000
 20. Is( 4, 7) =    0.2774   LambdaI =    0.0000
 21. Is( 8, 7) =    0.2353   LambdaI =    0.0000
 22. Is( 9, 7) =    0.3200   LambdaI =    0.0000
 23. Is( 7, 8) =    0.2353   LambdaI =    0.0000
 24. Is( 4, 9) =    0.1541   LambdaI =    0.0000
 25. Is( 7, 9) =    0.3200   LambdaI =    0.0000
 26. Is(10, 9) =    0.0697   LambdaI =    0.0000
 27. Is(14, 9) =    0.0533   LambdaI =    0.0000
 28. Is( 9,10) =    0.0697   LambdaI =    0.0000
 29. Is(11,10) =    0.1373   LambdaI =    0.0000
 30. Is( 6,11) =    0.1685   LambdaI =    0.0000
 31. Is(10,11) =    0.1373   LambdaI =    0.0000
 32. Is( 6,12) =    0.0844   LambdaI =    0.0000
 33. Is(13,12) =    0.0267   LambdaI =    0.0000
 34. Is( 6,13) =    0.2181   LambdaI =    0.0000
 35. Is(12,13) =    0.0267   LambdaI =    0.0000
 36. Is(14,13) =    0.1162   LambdaI =    0.0000
 37. Is( 9,14) =    0.0533   LambdaI =    0.0000
 38. Is(13,14) =    0.1162   LambdaI =    0.0000
PG_total  =       2.7397 pu.
PL_total  =       2.4183 pu.
Loss      =       0.3214 pu.
Cost      =     163.8254 $/hour
==========================================================================
Nu = 5.005e-16
==================Conditions in IP=======================================
Binding constraints on upper limits are:
Upper Limits Number:        5       86       87       88       89       90
```

```
                            91        92        93        94        95        96
Upper Limits       :     V  5    PL  2    PL  3    PL  4    PL  5    PL  6
                         PL  9    PL  10   PL  11   PL  12   PL  13   PL  14
Mu of Upper Limits    =   3.2296 100.6949 100.5392 100.2586 101.1749 101.2162
                        100.8341 100.8940 101.0480 101.2009 101.1562 100.9730
Slack of Upper Limits =   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                          0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
-------------------------------------------------------------------------
Binding constraints on lower limits are:
Lower Limits Number:       85
Lower Limits       :     PG  2
Mu of Lower Limits    =   0.2670
Slack of Lower Limits =   0.0000
=========================================================================
Expected Security Cost  =     165.8068 $/hour
```

VITA

Parnjit Damrongkulkamjorn

Candidate for the Degree of

Doctor of Philosophy

Thesis: OPTIMAL POWER FLOW WITH EXPECTED SECURITY COSTS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Bangkok, Thailand, on May 6, 1969, the daughter of Priyapas and Waewphan Damrongkulkamjorn.

Education: Graduated from St. Joseph Convent School, Bangkok, Thailand in March 1987; received Bachelor of Engineering degree in Electrical Engineering from Kasetsart University, Bangkok, Thailand, in May 1991, and Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma, in May 1993. Completed requirements for the Doctor of Philosophy degree with a major in Electrical Engineering at Oklahoma State University in May 1999.

Experience: Employed by Oklahoma State University, Department of Electrical and Computer Engineering, as a research assistant from August 1993 to December 1998.