Security-Constrained Unit Commitment Programming Project

May 9th, 2005

ECE 556

Power Market Operations

-Kathleen E. Williams

Abstract – This paper discusses the security-constrained unit commitment programming project guidelines, formulations, high-level and low-level programming, and procedures to complete a secure and economical system model. Low level programming was completed for the unit commitment portion. In lieu of security program, a high-level process flow will show security based constraints due to the author's own constraints discussed in the comments section of the report.

Table of Contents

Introduction

Security-Constrained Unit Commitment (SCUC) consists of two components, system security and economic dispatch. The objective of the problem is focused solely on the economic dispatch of generators with bidding segments, no load costs, starting costs, and other costs incurred during operation. Least cost is desired according to the system operators. Simple economics dictates that generators can supply (X) MW of power at location *xyz* and deliver the power through the transmission lines *L123*. However, this is not such a simple problem of who can "bid the lowest price" and win the contract in a short period of time. Location, transmission design, and the generator's operating efficiencies are conglomerated together to form a complex set of questions that the system operators must answer using analysis before awarding contracts to independent power producers. Thus, much planning and analysis is needed to determine the minimal cost that satisfies system security.

The power market operations course, ECE 556, at the Illinois Institute of Technology has set guidelines to help make cost-effect and secure dispatch decisions for diverse system models. The final project for the course outlines a good practice problem of determining the least-cost dispatch for three generators, 5 transmission lines, phase shifters, and bidding segments. The problem definition will describe the project overview with given information and final expectations. The game plan will then be drawn to show how the author tackled the challenge. Program components are discussed to categorize the project portions into manageable sub-programming projects. Finally the process flow shows a high-level procedure for the life of the program.

Problem Definition

The final project instructions dictated that a program be written to automatically formulate, compute, and solve the security-constrained unit commitment model.

The testing system, a 6-bus system, depicted in Figure 1, has 3 units, 5 transmission lines, and 2 phase shifting transformers. System load and reserve requirements over the 24-hour horizon are shown in Table 1. Individual unit constraints are shown in Table 2. Unit bidding information is shown in Table 3. It is assumed that bidding for all the hours are the same. Variable startup cost information is shown in Table 4. Transmission line data are shown in Table 5. Phase shifting transformer data are shown in Table 6. Load distribution factors are shown in Table 7.

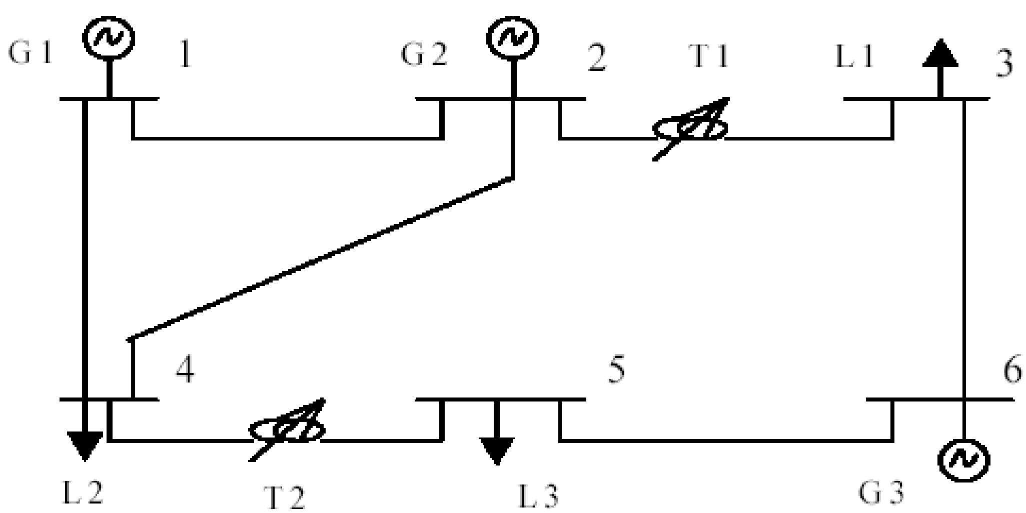

Figure 1. Single-line diagram for the 6-bus system

Table 1

Hour	Load (MW)	Spinning Reserve (MW)	Operating Reserve (MW)
1	175.19	2.63	12.26
2	165.15	2.48	11.56
3	158.67	2.38	11.1
4	154.73	2.32	10.83
5	155.06	2.33	10.85
6	160.48	2.4	11.23
7	173.39	2.6	12.14
8	177.6	2.85	13.33
9	186.81	3.09	14.39
10	206.96	3.26	15.2
11	228.61	3.43	16
12	236.1	3.54	16.52
13	242.18	3.63	16.95
14	243.6	3.66	17.05
15	248.86	3.73	17.42
16	255.79	3.84	17.91
17	256	3.84	17.92
18	246.74	3.7	17.27
19	245.97	3.69	17.22
20	237.35	3.56	16.62
21	237.31	3.56	16.62
22	232.67	3.41	15.9
23	195.93	3.02	14.07
24	195.6	2.95	13.78

Table 2a

Unit Name	Minimum Capacity (MW)	Maximum Capacity (MW)	No-load Cost ($)	Startup Cost ($)	Shutdown Cost ($)	Minimum Up Time (Hours)	Minimum Down Time (Hours)
G1	100	220	200	100	50	4	4
G2	10	100	150	200	40	3	2
G3	10	20	50	0	0	1	1

Table 2b

Unit Name	Ramp Up Rate (MW/Hour)	Ramp Down Rate (MW/Hour)	MSR (MW/min)	QSC (MW)	Initial Status	Initial Hour	Initial MW
G1	40	50	2	15	ON	4	140
G2	30	35	1.5	10	ON	3	20
G3	20	20	0.5	10	ON	2	10

Table 3

Unit Name	Bidding Segments	MW	Price ($/MWh)
G1	1	40	20
G1	2	60	25
G1	3	80	24
G1	4	40	23
G2	1	30	24
G2	2	50	26
G2	3	20	28
G3	1	10	30
G3	2	20	32

Table 4

Unit Name	Hot Start (Hours)	Hot Start ($)	Warm Start (Hours)	Warm Start ($)	Cold Start (Hours)	Cold Start ($)
G1	4	100	8	150	12	180
G2	2	200	4	300	6	360
G3	1	0	2	50	3	60

Table 5

Line No.	From Bus	To Bus	R (pu)	X (pu)	Flow Limit (MW)
1	1	2	0.0050	0.170	200
2	1	4	0.0030	0.258	100
3	2	4	0.0070	0.197	100
4	5	6	0.0020	0.140	100
5	3	6	0.0005	0.018	100

Table 6

Phase-shifting Transformer No.	From Bus	To Bus	R (pu)	X (pu)	Angle Max	Angle Min
T1	2	3	0	0.037	30	-30
T2	4	5	0	0.037	15	-15

Table 7

Load Name	Bus No.	Percentage of System Load
L1	3	20%
L2	4	40%
L3	5	40%

The consideration of phase-shifters and variable start-up costs was optional in the final project problem definition.

Game Plan

The approach on the project should have been at least a month before the due date to allow for plenty of time to tackle bugs in the program and formulations. The author's approach was to tackle each sub problem of the unit commitment separately and combine the constraints and objective functions together into one master matrix to be processed by the MIP program. The project began determining a cost formulation for the test case generators. It was clear that the price of the generator would influence the formulation and the programming of the objective function and constraints. Formulation was done for every test case generator and a "variable-matrix" was formed to represent all the indicator variables, continuous variables, etc that the individual price and quantities formulated. Only the first few hours were used to test the program and generate constraints like the cost function for the generators. The author learned to develop algorithms to generate common patterns for NT (number of hours) hours. The start-up and shut-down indicators were formulated for the test case generators at any hour. Sample data was manually entered into excel from the formulations for the first test case generator and for the first few hours. The author then developed, after many long hours and failed attempts, loops to generate the correct matrices for hours 1:NT. The objective matrix, f, and the constraints, A, and Ae, were then exported from MATLAB using –ascii commands and then imported into excel. The output from the program was then compared against the expected output for the first few hours. "If"-else" statements, generated in excel, highlighted the programming output cells that deviated from the expected results. If the routines worked, they were added to the master program and the additional constraints were added to the master constraint matrix in the program. This process of write formulations, manually enter in first few hours test case generator 1, develop algorithm, write code, test code, export output, and lastly compare output to formulations continued for every subcomponent of the unit commitment.

Program Components

Unit Commitment

Cost Functions

For each test generator, there are bidding prices for each segment of MW generated. Segment prices can be either non-convex or convex. The prices are used in the objective function and it is desired to minimize these costs. The constraints for the formulations are also included.

Start-up and Shut-down indicators

These indicators represent the status of the generator, whether it is starting up or shutting down. Indicators are used for NT hours (24 hours in the case) and are useful in other formulations.

Start-up and Shut-down costs

Additional costs associated with the above start-up and shut-down indicators. It is the objective to minimize these costs throughout the system and throughout NT hours.

Capacities

Each test case generator has a minimum capacity it is willing to generate as being committed and a maximum capacity. These min and max limits are formulated as constraints for each generator for NT hours.

Reserves

The 10-minute spinning reserve of a unit is the unloaded synchronized generation that can ramp up in 10 minutes. The spinning reserve of a unit cannot exceed the difference between its maximum capacity and current generation. It is also limited by the 10-minute maximum sustained rate. Operating reserve is the unloaded synchronized/unsynchronized generating capacity that can ramp up in 10 minutes. When a unit is in operation, its operating reserve is the same as spinning reserve. When a unit is down, its operating reserve is the same as its quick start capability. These limits are represented as constraints with their respective indicator variables.

Ramping Constraints

From one hour to the next, a unit cannot increase its output above a maximum increment, which is called the ramping up limit. Similarly, a unit cannot decrease its output above a maximum decrement, which is called the ramping down limit. These limits are represented as constraints with indicator variables.

Minimum uptime/downtime constraints

Minimum up time constraint implies that a unit must stay in operation for a certain number of hours before it can be shut down; minimum down time constraint implies that a unit must remain down for a certain number of hours before it can be brought online. Indicator variables can be used to represent these limits in the constraints.

System load balance and reserve requirements

These are constraints that the generators must meet the set load D, spinning reserve requirements, and operating reserve requirements. These constraints are also represented using indicator variables.

Variable Start-Up Costs

These costs were not considered in the project, however, they are time sensitive and depend on previous hour status.

Security

DC Power Flow

DC power flow is used a quick solution to the test case system. The bus, load, and line data is entered into the program as arrays. The impedance matrices can be determined.

Power Transfer Distribution Factors

Based on the system bus, line, and load data, PTDF's can be formulated for each bus 1-N, N-N, and then represented as constraints using indicator variables. Line flow and phase shifter limits must also be included.

Process Flow

Flow diagrams will illustrate some high-level processes for the program. The first diagram, Figure 2, shows the game plan process for each subcomponent.

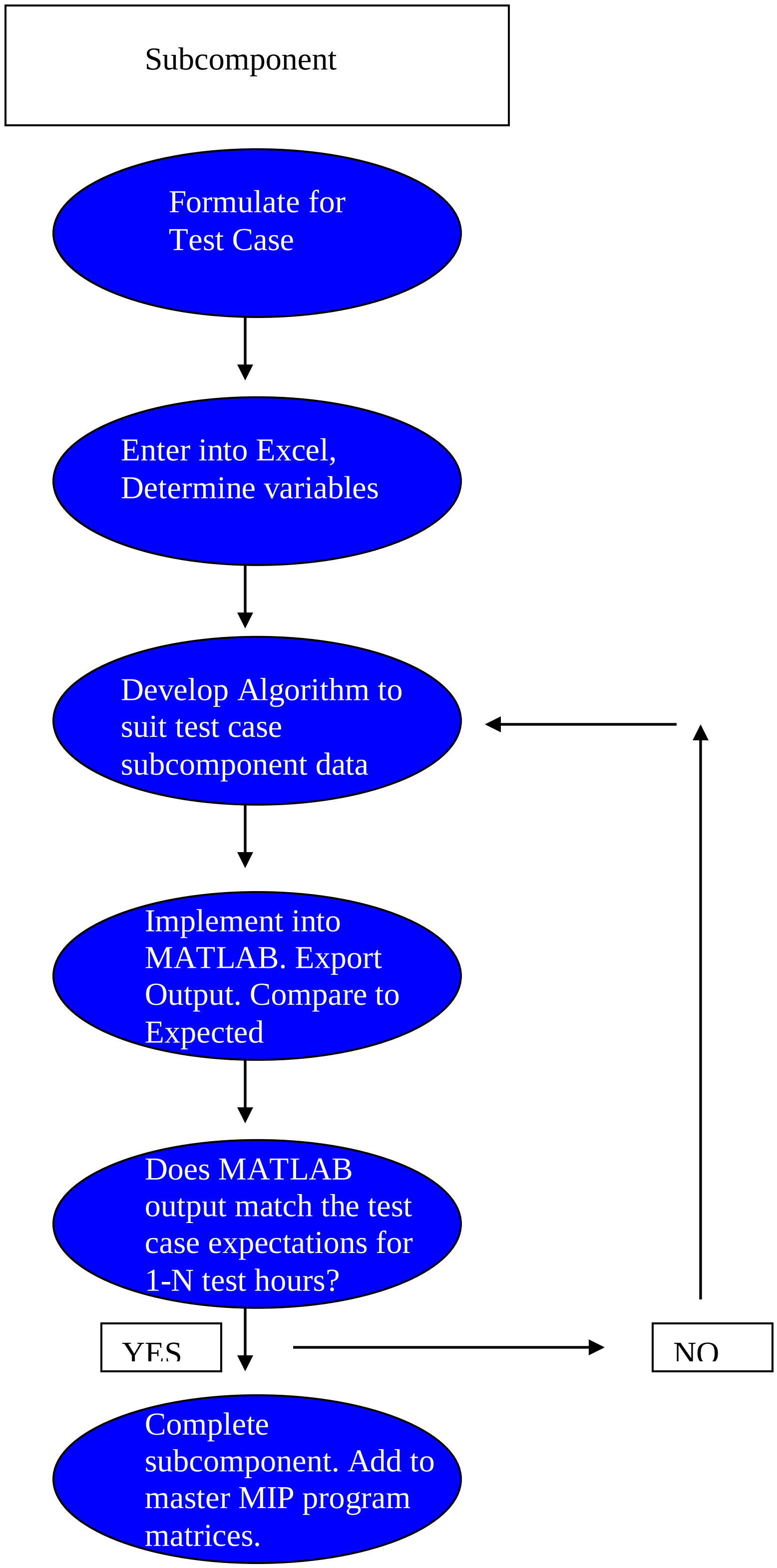

Figure 3 shows a description of the two parts of the program project. Only the unit commitment portion was completed except for the minimum uptime/down time component.

Unit Commitment

For 1:N Generators()

Costs represented in the objective function ($/MWh, start-up, shut-down, no-load)
Price variable constraints, bidding segment constraints
Minimum capacity, maximum capacities and other constant variables
Upper bounds, lower bounds, variable array (integer/continuous). Every subcomponent for the unit commitment has the constraints represented in matrix form for A, Ae, b, be, xint, lb,ub, and f.

Summary – When the function completes, each generator will have a MIP objective function, equality and inequality arrays, upper bounds, lower bounds, and x-array. Since the generator constants do not change for each hour, there will be only one MIP for each generator usable for hours 1:N. These will be master reference arrays for calculating other components of the program. Each subcomponent for the unit commitment is represented inside the generator function because the generator influences the unit commitment. Each subcomponent also generates its own f, A, Ae, b, be, xint, lb, ub, and f matrices. When the subcomponent procedure is complete, the results are added to the master matrices representing the unit commitment MIP.

Inputs:
(noload,startupcost,shutdowncost,price,quantity,NT,PMIN,PMAX,QSC,MSR,RU,RD,D,SR,OR)
Outputs:
(f,Aeq,beq,Aeinq,beinq,lb,ub,x)

Security

For 1:N Buses()

Imports bus and lines data. Computes admittance matrix and calculates power transfer distribution factors. Limits on the line flows are entered as constraints along with phase shift limits.

Summary – The PTDF are calculated for N buses for K lines. The constraints hold PTDF information and flow limits

Inputs:
(line, frombus, tobus, r, x, flowlimit, pstrans, frombus, tobus, r, x , anglemax , anglemin, load, busno, percent)
Outputs:
(f,Aeq,beq,Aeinq,beinq,lb,ub,x)

II. Unit Commitment

This section of the final project for programming the SCUC will focus only on the unit commitment portion of the problem. The formulation section lists all relevant equations needed to form the unit commitment. The algorithms section discusses some techniques in generating the constraints or implementing the formulation into MATLAB. The appendix lists code referenced for unit commitment.

Formulation

Cost Functions

For a convex function the formulation is as follows:

Convex function

$$c_{it} = C_{i0} u_{it} + \sum_{m} IC_{im} px_{it,m}$$

$$p_{it} = \sum_{m} px_{it,m}$$

$$0 \le px_{it,m} \le MW_{im}$$

For a non-convex function, the formulation is as follows:

Non-convex function

$$c_{it} = C_{i0} u_{it} + \sum_{m} IC_{im} px_{it,m}$$

$$p_{it} = \sum_{m} px_{it,m}$$

$$MW_{i1}\delta_{it,1} \le px_{it,1} \le MW_{i1} u_{it}, \text{ m=1 (the first piece)}$$

$$MW_{im}\delta_{it,m} \le px_{it,m} \le MW_{im}\delta_{it,m-1},\ 2 \le m \le M-1$$

$$0 \le px_{it,m} \le MW_{im}\delta_{it,m-1}, \text{ m=M (the last piece)}$$

Table 3 below represent the bidding segments and prices for the *test case* in the assignment. Table 2a represents the test case unit capacities and no-load costs. G1 is non-convex, G2 is convex, G3 is convex

For G1 at any hour

$c_{1,t} = 200u_{1,t} + 20px_{1,t,1} + 25px_{1,t,2} + 24px_{1,t,3} + 23px_{1,t,4}$

$p_{1,t} = px_{1,t,1} + px_{1,t,2} + px_{1,t,3} + px_{1,t,4}$

$40\delta_{1,t,1} \leq px_{1,t,1} \leq 40u_{1,t}$

$60\delta_{1,t,2} \leq px_{1,t,2} \leq 60\delta_{1,t,2}$

$80\delta_{1,t,3} \leq px_{1,t,3} \leq 80\delta_{1,t,3}$

$0 \leq px_{1,t,4} \leq 40\delta_{1,t,4}$

Objective Function Array

f =

u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
200	0	20	25	24	23	0	0	0	0	0	100	50	0	0

Constraints Matrix

		u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
=	0	0	-1	1	1	1	1	0	0	0	0	0	0	0	0	0
E	0	0	0	-1	0	0	0	0	40	0	0	0	0	0	0	0
E	0	-40	0	1	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	-1	0	0	0	0	60	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	-60	0	0	0	0	0	0
E	0	0	0	0	0	-1	0	0	0	0	80	0	0	0	0	0
E	0	0	0	0	0	1	0	0	0	0	-80	0	0	0	0	0
E	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	1	0	0	0	0	-40	0	0	0	0

Bounds Arrays

	u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
lb	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ub	1	inf	inf	inf	inf	inf	1	1	1	1	1	1	1	1	1
x	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1

For G2 at any hour
$c_{2,}t = 150u_{2,t} + 24px_{2,t,1} + 26px_{2,t,2} + 28px_{2,t,3}$
$p_{2,t} = px_{2,t,1} + px_{2,t,2} + px_{2,t,3}$
$0 \leq px_{2,t,1} \leq 30$
$0 \leq px_{2,t,2} \leq 50$
$0 \leq px_{2,t,3} \leq 20$

objective function array

f =

	$c_{1,}t$	$u_{1,t}$	$p_{1,t}$	$px_{1,t,1}$	$px_{1,t,2}$	$px_{1,t,3}$	$px_{1,t,4}$	$\delta_{1,t,}$	$\delta_{1,t,1}$	$\delta_{1,t,2}$	$\delta_{1,t,3}$	$\delta_{1,t,4}$
=	-1	150	0	24	26	28	-	0	0	0	0	0

Constraints matrix

		$c_{1,}t$	$u_{1,t}$	$p_{1,t}$	$px_{1,t,1}$	$px_{1,t,2}$	$px_{1,t,3}$	$px_{1,t,4}$	$\delta_{1,t,}$	$\delta_{1,t,1}$	$\delta_{1,t,2}$	$\delta_{1,t,3}$	$\delta_{1,t,4}$
=	0	0	0	-1	1	1	1	-	-	-	-	-	-
E	0	0	0	0	-1	0	0	-	-	-	-	-	-
E	0	0	0	0	0	-1	0	-	-	-	-	-	-
E	0	0	0	0	0	0	-1	-	-	-	-	-	-
E	30	0	0	0	1	0	0	-	-	-	-	-	-
E	50	0	0	0	0	1	0	-	-	-	-	-	-
E	20	0	0	0	0	0	1	-	-	-	-	-	-

		$c_{1,}t$	$u_{1,t}$	$p_{1,t}$	$px_{1,t,1}$	$px_{1,t,2}$	$px_{1,t,3}$	$px_{1,t,4}$	$\delta_{1,t,}$	$\delta_{1,t,1}$	$\delta_{1,t,2}$	$\delta_{1,t,3}$	$\delta_{1,t,4}$
lb		0	0	0	0	0	0	-	-	-	-	-	-
ub		Inf	1	inf	inf	inf	inf	-	-	-	-	-	-
x		0	1	0	0	0	0	-	-	-	-	-	-

For G3 at any hour
$c_{3,}t = 50u_{3,t} + 30px_{3,t,1} + 32px_{3,t,2}$
$p_{3,t} = px_{3,t,1} + px_{3,t,2}$
$0 \leq px_{2,t,1} \leq 10$
$0 \leq px_{2,t,2} \leq 20$

Start-up and shut-down indicators

$$y_{it} - z_{it} = u_{it} - u_{i,t-1}$$

$$y_{it} + z_{it} \leq 1$$

For G1 at any hour

$y_{1,t} - z_{1,t} - u_{1,t} + u_{1,(t-1)} = 0$
$y_{1,t} + z_{1,t} \leq 1$

For G2 at any hour

$y_{2,t} - z_{2,t} - u_{2,t} + u_{2,(t-1)} = 0$
$y_{2,t} + z_{2,t} \leq 1$

For GN at any hour

$y_{N,t} - z_{N,t} - u_{N,t} + u_{N,(t-1)} = 0$
$y_{N,t} + z_{N,t} \leq 1$

Example for G1 for Hours 1 through 4

objective function array

f =

	$u_{1,1}$	$y_{1,1}$	$z_{1,1}$	$u_{1,2}$	$y_{1,2}$	$z_{1,2}$	$u_{1,3}$	$y_{1,3}$	$z_{1,3}$	$u_{1,4}$	$y_{1,4}$	$z_{1,4}$
=	0	0	0	0	0	0	0	0	0	0	0	0

Constraints matrix

		$u_{1,1}$	$y_{1,1}$	$z_{1,1}$	$u_{1,2}$	$y_{1,2}$	$z_{1,2}$	$u_{1,3}$	$y_{1,3}$	$z_{1,3}$	$u_{1,4}$	$y_{1,4}$	$z_{1,4}$
=	0	-1	1	-1	0	0	0	0	0	0	0	0	0
=	0	1	0	0	-1	1	-1	0	0	0	0	0	0
=	0	0	0	0	1	0	0	-1	1	-1	0	0	0
=	0	0	0	0	0	0	0	1	0	0	-1	1	-1
E	1	0	1	1	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	1	1	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	1	1	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	1	1

For Hour 1

		u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
=	0	-1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
=	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

For Hour 2

		u1,2	p1,2	px1,1,2	px1,1,2	px1,1,2	px1,1,2	δ1,2	δ1,1,2	δ1,2,2	δ1,2,3	δ1,2,4	y1,2	z1,2	sr1,2	or1,2
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	-1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
=	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

For Hour 3

		u1,3	p1,3	px1,1,3	px1,1,3	px1,1,3	px1,1,3	δ1,3	δ1,3,1	δ1,3,2	δ1,3,3	δ1,3,4	y1,3	z1,3	sr1,3	or1,3
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	-1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
=	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bounds Arrays

		$u_{1,1}$	$y_{1,1}$	$z_{1,1}$	$u_{1,2}$	$y_{1,2}$	$z_{1,2}$	$u_{1,3}$	$y_{1,3}$	$z_{1,3}$	$u_{1,4}$	$y_{1,4}$	$z_{1,4}$
lb		0	0	0	0	0	0	0	0	0	0	0	0
ub		1	1	1	1	1	1	1	1	1	1	1	1
x		1	1	1	1	1	1	1	1	1	1	1	1

Notes: The arrays will be the same pattern for each generator but the combined array will have variables representative of each generator for each hour

For a N Generator system with NT hours, there will be (3)X(N)X(NT) variables and (2)X(N)X(NT) rows. For the test case, 3 generators and 24 hours, there will be 216 variables to represent the start-up and shut-down indicators and 144 rows with constraints.

Combined objective function

f = [G1variables Hour1 G1 variables Hour 2 …. G1 variables Hour NT G2variables Hour1 G2 variables Hour 2 …. G2 variables Hour NT …. GNvariables Hour1 GN variables Hour 2 …. GN variables Hour NT]

Similar procedure as above can be applied to combine the equality and inequality constraints and upper, lower bounds, and integer array.

Start-up and Shut-down Costs

$$csu_{it} = ST_i y_{it}$$

$$csd_{it} = SD_i z_{it}$$

For the test case:

Table 2a

Unit Name	Minimum Capacity (MW)	Maximum Capacity (MW)	No-load Cost ($)	Startup Cost ($)	Shutdown Cost ($)
G1	100	220	200	100	50
G2	10	100	150	200	40
G3	10	20	50	0	0

For G1

f =

	$u_{1,1}$	$y_{1,1}$	$z_{1,1}$	$u_{1,2}$	$y_{1,2}$	$z_{1,2}$	$u_{1,3}$	$y_{1,3}$	$z_{1,3}$	$u_{1,4}$	$y_{1,4}$	$z_{1,4}$
=	0	100	50	0	100	50	0	100	50	0	100	50

Similar process is used for G2 and G3 variables in the objective function

Capacities

$$PMIN_i u_{it} \leq p_{it} \leq PMAX_i u_{it}$$

For any hour

G1:
$100u_{1,t} \leq px_{1,t} \leq 220u_{1,t}$

G2

$10u_{2,t} \leq px_{2,t} \leq 100u_{2,t}$

G3

$10u_{2,t} \leq px_{1,t} \leq 20u_{2,t}$

For the first 4 hours for G1

Constraints matrix

		u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
0	0	-220	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	100	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

		$u_{1,1}$	$px_{1,1}$	$u_{1,2}$	$px_{1,2}$	$u_{1,3}$	$px_{1,3}$	$u_{1,4}$	$px_{1,4}$
E	0	-220	1	0	0	0	0	0	0
E	0	100	-1	0	0	0	0	0	0
E	0	0	0	-220	1	0	0	0	0
E	0	0	0	100	-1	0	0	0	0
E	0	0	0	0	0	-220	1	0	0
E	0	0	0	0	0	100	-1	0	0
E	0	0	0	0	0	0	0	-220	1
E	0	0	0	0	0	0	0	100	-1

		$u_{1,1}$	$px_{1,1}$	$u_{1,2}$	$px_{1,2}$	$u_{1,3}$	$px_{1,3}$	$u_{1,4}$	$px_{1,4}$
lb		0	0	0	0	0	0	0	0
ub		1	inf	1	inf	1	inf	1	inf
x		1	0	1	0	1	0	1	0

Note the objective function should remain unchanged for this subpart of the MILP problem

Reserves

Spinning Reserve

$$p_{it} + sr_{it} \leq PMAX_i$$

$$0 \leq sr_{it} \leq u_{it}(10MSR_i)$$

Operating Reserve

$$or_{it} = sr_{it} + (1 - u_{it})QSC_i$$

For any hour the test case can be modeled as:

For G1:
$p_{1,t} + sr_{1,t} \leq 220$
$0 \leq sr_{1,t,} \leq u_{1,t}(10*2)$
$or_{1,t} = sr_{1,t} + (1 - u_{1,t})15$

For G2:
$p_{2,t} + sr_{2,t} \leq 100$
$0 \leq sr_{2,t,} \leq u_{2,t}(10*1.5)$
$or_{2,t} = sr_{2,t} + (1 - u_{2,t})10$

For G3:
$p_{3,t} + sr_{3,t} \leq 20$
$0 \leq sr_{3,t,} \leq u_{3,t}(10*0.5)$
$or_{3,t} = sr_{3,t} + (1 - u_{3,t})10$

Example for G1 for Hours 1 through 3

constraints

		$u_{1,1}$	$p_{1,1}$	$sr_{1,1}$	$or_{1,1}$	$u_{1,2}$	$p_{1,2}$	$sr_{1,2}$	$or_{1,2}$	$u_{1,3}$	$p_{1,3}$	$sr_{1,3}$	$or_{1,3}$
=	15	15	0	-1	1	0	0	0	0	0	0	0	0
E	220	0	1	1	0	0	0	0	0	0	0	0	0
E	0	-10*2	0	1	0	0	0	0	0	0	0	0	0
E	0	0	0	-1	0	0	0	0	0	0	0	0	0
=	15	0	0	0	0	15	0	-1	1	0	0	0	0
E	220	0	0	0	0	0	1	1	0	0	0	0	0
E	0	0	0	0	0	-10*2	0	1	0	0	0	0	0
E	0	0	0	0	0	0	0	-1	0	0	0	0	0
=	15	0	0	0	0	0	0	0	0	15	0	-1	1
E	220	0	0	0	0	0	0	0	0	0	1	1	0

E	0	0	0	0	0	0	0	0	0	-10*2	0	1	0
E	0	0	0	0	0	0	0	0	0	0	0	-1	0

		u1,1	p1,1	px1,1,1	px1,1,2	px1,1,3	px1,1,4	δ1,1	δ1,1,1	δ1,1,2	δ1,1,3	δ1,1,4	y1,1	z1,1	sr1,1	or1,1
0	0	-220	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	100	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ramping Constraints

$$p_{it} - p_{i,t-1} \le y_{it} PMIN_i + (1 - y_{it}) RU_i$$

$$p_{i,t-1} - p_{it} \le z_{it} PMIN_i + (1 - z_{it}) RD_i$$

For the test case

G1 at any hour
$p_{1,t} - p_{1,t-1} \le y_{1,t}100 + (1-y_{1,t})40$
$p_{1,t-1} - p_{1,t} \le z_{1,t}100 + (1-z_{1,t})50$

G2 at any hour
$p_{2,t} - p_{2,t-1} \le y_{2,t}10 + (1-y_{2,t})30$
$p_{2,t-1} - p_{2,t} \le z_{2,t}10 + (1-z_{2,t})25$

G3 at any hour
$p_{3,t} - p_{3,t-1} \le y_{3,t}10 + (1-y_{3,t})20$
$p_{3,t-1} - p_{3,t} \le z_{3,t}10 + (1-z_{3,t})20$

Example for G1 for Hours 1 through 4

constraints

		$p_{1,1}$	$y_{1,1}$	$z_{1,1}$	$p_{1,2}$	$y_{1,2}$	$z_{1,2}$	$p_{1,3}$	$y_{1,3}$	$z_{1,3}$
E	40	1	-(100-40)	0	0	0	0	0	0	0
E	50	-1	0	-(100-50)	0	0	0	0	0	0
E	40	-1	0	0	1	-(100-40)	0	0	0	0

E	50	1	0	0	-1	0	-(100-50)	0	0	0
E	40	0	0	0	-1	0	0	1	-(100-40)	0
E	50	0	0	0	1	0	0	-1	0	-(100-50)
E	40	0	0	0	0	0	0	-1	0	0
E	50	0	0	0	0	0	0	1	0	0

Minimum Up/Down Time

Minimum Up Time

$$UT_i = \max\{0, \min[NT, (MU_i - TU_{i0})U_{i0}]\}$$

$$\sum_{t=1}^{UT_i}(1 - u_{it}) = 0$$

$$\sum_{m=t}^{t+MU_i-1} u_{im} \geq MU_i y_{it} \quad \forall t = UT_i + 1, \ldots, NT - MU_i + 1$$

$$\sum_{m=t}^{T}(u_{im} - y_{it}) \geq 0 \quad \forall t = NT - MU_i + 2, \ldots, NT$$

Minimum Down Time

-

$$DT_i = \max\{0, \min[NT, (MD_i - TD_{i0})(1 - U_{i0})]\}$$

$$\sum_{t=1}^{DT_i} u_{it} = 0$$

$$\sum_{m=t}^{t+MD_i-1}(1 - u_{im}) \geq MD_i z_{it} \quad \forall t = DT_i + 1, \ldots, NT - MD_i + 1$$

$$\sum_{m=t}^{T}(1 - u_{im} - z_{it}) \geq 0 \quad \forall t = NT - MD_i + 2, \ldots, NT$$

System Load Balance and Reserve Requirements

Load balance is modeled as

$$\sum_i p_{it} = D_t$$

Spinning reserve requirement is modeled as

$$\sum_i sr_{it} \geq SR_t$$

Operating reserve requirement is modeled as

$$\sum_i or_{it} \geq OR_t$$

Example for G1 and G2 for Hours 1 through 2

objective function

constraints

		$p_{1,1}$	$sr_{1,1}$	$or_{1,1}$	$p_{1,2}$	$sr_{1,2}$	$or_{1,2}$	$p_{2,1}$	$sr_{2,1}$	$or_{,21}$	$p_{2,2}$	$sr_{2,2}$	$or_{2,2}$
=	-D1	1	0	0	0	0	0	1	0	0	0	0	0
=	-D2	0	0	0	1	0	0	0	0	0	1	0	0
E	-SR1	0	-1	0	0	0	0	0	-1	0	0	0	0
E	-SR2	0	0	0	0	-1	0	0	0	0	0	-1	0
E	-OR1	0	0	-1	0	0	0	0	0	-1	0	0	0
E	-OR2	0	0	0	0	0	-1	0	0	0	0	0	-1

Algorithms

Determine Function Type

```
For 1:length(price)
  Is value(price) < previous value
    Yes → non-convex function
    No → convex function
```

Start matrix, generate other hours

Many of the subcomponents of the unit commitment program had a common algorithm. First there was the constraint matrix for the first hour. Subsequent matrices were dependent upon the values from hour 1. Also, these other matrices were formed using dynamic references. For example, if the matrix from hour 4 was formed, it was based off the matrix from hour 3. The matrix from hour 2 was formed off the basis from hour 1. Thus only the first hour need be programmed well and the following hours have values that stem from the previous hour but are shifted either vertically or horizontally, but most of the time vertically in the matrix. The amount of shift is static and determined by looking at patterns for the test case matrices shown in the formulation section of the report. Once a common shift pattern is know, it is programmed and then tested.

```
Is this matrix coordinate within the first hour?

Yes
  Is this cell coordinate (x,y) == K ?
    Yes
      Set Value
    No
      Is this cell coordinate (k,l) == M ?
        Yes
          Set Value
        No
         ……………………………………
No
  Set value of (x,y) to value of (x - shift,y - shift)
```

III. Security-Constraints

Formulation

Define

$$B' = \begin{bmatrix} -(b_{12}+b_{13}) & b_{12} & b_{13} \\ b_{21} & -(b_{21}+b_{23}) & b_{23} \\ b_{31} & b_{32} & -(b_{31}+b_{32}) \end{bmatrix}$$

In general,

$$B'_{ii} = -\sum_m b_{im} \,,\; B'_{im} = b_{im}$$

Assume $r_{im} << x_{im}$, we have

$$b_{im} = \frac{-x_{im}}{r_{im}^{\;2} + x_{im}^{\;2}} \approx -\frac{1}{x_{im}}$$

Then

$$B'_{ii} = \sum_m \frac{1}{x_{im}} \,,\; B'_{im} = -\frac{1}{x_{im}}$$

$$PL_{im} = -b_{im}(\theta_{im} - \gamma_{im}) = \frac{\theta_i - \theta_m - \gamma_{im}}{x_{im}}$$

DC power flow equations with phase shifters are

$$P_G - P_D + \gamma / x = B'\theta$$

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = [J] \begin{bmatrix} \Delta \delta \\ \Delta V \end{bmatrix}$$

Procedures

In lieu of not completing the program for the security portion of the programming project, process flows will illustrate a technique for completion.

Import Line Impedances
Create B, Admittance Matrix
Form the Pg-Pd + y/x matrix = P
Solve for the angles, theta.
P = b*theta
Eliminate the reference bus values to help solve the theta values
Form Line Flows, PL(1-2) = (theta1 – theta2 – y12)/x12
For 1-N lines
Determine shift factors, add to the PTDF
Enter the equations into the constraint matrix
Enter line flow limits into the constraint matrix
Enter the phase shifter limits into the constraint matrix
Create bounds, xint
Add the A,Ae,b,be,lb,ub,xint to the unit commitment A,Ae,b,be,lb,ub,xint matrices.
Run the MIP

IV. Commentary

Conclusions

The SCUC programming project was a challenge that was not completed due to the author's time constraints (see project planning). The unit commitment portion had each subcomponent minus the minimum uptime/downtime completed and only for non-convex generator price functions. At the end of the project deadline, the program encounter bugs for the convex price functions that were not fixed due to the deadline. The MATLAB code in the appendix will demonstrate the unit commitment functionality for a non-convex function, generator one in the test case example.

Project Planning

I would like to add some personal commentary to the project. Firstly I spent lots of hours (> 20) on the final project only to have to abruptly end . I started the final project about 2 weeks from the deadline because of having to work on coursework and homework for two classes. I had a few wrong starts in the program design. I came to a few road blocks in generating some code to loop the subcomponents. In the few days before the deadline only did I become proficient in the process of developing subcomponents for the SCUC and fully understanding the security portion with the help of the last assignment due. Other personal blocks constrained my time available for the project: working full-time, taking another course, having to move entirely to a different apartment, being sick,etc. If I could redo the project, I would have started a month before the deadline, on the unit commitment portion, even though security-based assignments was not addressed until the past few weeks. One month development time seems more feasible than two weeks project planning. But I must add, that pursing the programming project offered more of an enjoyable challenge that writing a paper. I’m still glad that I chose the programming project even though was not completed!

V. Appendix

MATLAB Code – Main Program to formulate unit commitment for a non-convex function

```
% Generator # 1 Information
noload = [200];
startupcost=[100];
shutdowncost=[50];
price = [20 25 24 23];
quantity = [40 60 80 40];
NT = [24];
PMIN= [100];
PMAX= [220];
QSC = [15];
MSR = [2];
RU = [40];
RD = [50];
D =  [175.19 165.15 158.67 154.73 155.06 160.48 173.39 177.6 186.81 206.96 228.61 236.1 242.18 243.6 248.86 255.79 256 246.74
245.97 237.35 237.31 232.67 195.93 195.6];
SR = [2.63 2.48 	2.38 2.32 2.33 2.4 2.6 2.85 3.09 3.26 3.43 3.54 	3.63 3.66 3.73 3.84 	3.84 3.7 3.69 3.56 3.56
	3.41 3.02 2.95];
OR = [12.26 	11.56 11.1 10.83 10.85 11.23 12.14 13.33 14.39 15.2 	16 16.52 16.95 17.05 17.42 17.91 17.92 17.27 17.22
16.62 16.62 15.9 	14.07 13.78];

% Create objective function, constraints, bounds, based on generator values
% listed above

[f,Aeq,beq,Aeinq,beinq,lb,ub,x]=generator(noload,startupcost,shutdowncost,price,quantity,NT,PMIN,PMAX,QSC,MSR,RU,RD,D,S
R,OR);

A = Aeinq;
b = beinq;
xint = x;
clear Aeinq, clear beinq, clear x;

% Call MIP solver
[x,fval,exitflag,output,lambda]=mipprog(f,A,b,Aeq,beq,lb,ub,xint);

% output results
x

% objective function value
fval
```

MATLAB Code – Generator function

```
%generator  Makes generator objective function and constraints
%
%  SYNOPSIS: [f,Aeq,beq,Aeinq,beinq,lb,ub,x]=generator(noload,startupcost,shutdowncost,price,quantity, NT)
%
%  Determines type of price function (non-convex or convex)
%  Depending on type, makes objective function, equality constraints,
%  inequality constraints, lower bounds, upper bounds, and x
%  (integer/continous) array for 1 generator given a 1XN array for the
%  price, a 1XN array for the quantity, and a 1X1 array with the noload
%  cost, startup cost, and shut down cost. Everything else is dynamically created.
%  NT is the number of hours under study, other variables are defined.


function
[f,Aeq,beq,Aeinq,beinq,lb,ub,x]=generator(noload,startupcost,shutdowncost,price,quantity,NT,PMIN,PMAX,QSC,MSR,RU,RD,D,S
R,OR)

%
% Determine Function Type
%

% Dummy Variables
temp = price(1);
flag1 = 0;

for i=2:length(price)
if price(i) < temp
   flag1 = 1;
else
   temp = price(i);
end;
end;

%
% If flag1 = 1, this is a non-convex function (the price decreases)
% If flag1 = 2, this is a convex function, the price is non-decreasing
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Non-Convex Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if flag1 == 1
   % Form non-convex functions
   c = zeros(1,length(price));
   for i=1:length(c)
      c(i) = price(i);
   end;

   % generate p variables
   p = zeros(1,length(price));
   p(1) = 1;
   for i=1:length(p)
      p(i) = 1;
   end;

   % generate delta variables
   d = zeros(1,length(price));
   d(1) = 1;
   for i=1:length(d)
      d(i) = 1;
   end;

   u = zeros(1,1+length(d));
   f = zeros(1,length(u));
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Objective function
    f1 = [noload 0 c f startupcost shutdowncost 0 0];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    f = [];
    for i=1:NT
       f = [f f1];
    end;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Equality constraints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    e = zeros(1,length(d));
    e = [0 -1 p 0 e 0 0 0 0];
    Aeq = [];
    for i=1:NT
     Aeq = [Aeq e];
    end;

    beq = [0];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Inequality contraints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     % this generates the constraint array
    %make temp array
    temp = zeros(1,length(price));
    t = [-1;1];
    s = [0;0];
    temp2 = [];
    for i=1:length(price)
      for j=1:length(price)
         if i == 1
           if j ==1
             temp2 = [t];
           else
             temp2 = [temp2 s];
           end;
         else

          if j == i
            temp2 =  [temp2 t];
          else
            temp2  = [temp2 s];
          end;
         end;
      end;
    end;
temp2;
j=1;
k = length(price);
temp3 = [];
for i=1:length(price)
 temp3 = [temp3; temp2(1:2,j:k)];
 j = j + length(price);
 k = k + length(price);
end;
temp3;

mastermatrix = temp3;
deltamatrix = mastermatrix;

pcolumn = [zeros(1,length(mastermatrix))]';
scolumn = [zeros(1,length(mastermatrix))]';
ocolumn = [zeros(1,length(mastermatrix))]';
ycolumn = [zeros(1,length(mastermatrix))]';
zcolumn = [zeros(1,length(mastermatrix))]';
dcolumn = [zeros(1,length(mastermatrix))]';
```

```
ccolumn = [zeros(1,length(mastermatrix))]';
ucolumn = [0 -quantity(1) zeros(1,length(mastermatrix)-2)]';

% temp2(1:2,i:j)
% temp2(1:2,1:3) i                j = length(p)
% temp2(1:2,4:6) i = i + length(p)  j = j + length(p)
% temp2(1:2,7:9) i = i + length(p)  j = j + length(p)

% add the zero column

mastermatrix = [ucolumn pcolumn mastermatrix dcolumn];

% add the delta values

for i=1:length(deltamatrix)
    for j =1:length(deltamatrix)/2
        deltamatrix(i,j) = 0;
    end;
end;


% add delta values

for i=1:length(deltamatrix)
    for j =1:length(deltamatrix)/2
        if i == 1 && j == 1
            deltamatrix(i,j) = quantity(1);
        else
           if i == length(deltamatrix) &&  j == length(deltamatrix)/2
             deltamatrix(i,j) = -quantity(length(quantity));
           else
            if (j > 1 && j < length(deltamatrix)/2) && (i > 2 && i < length(deltamatrix)-1)

                if (j*2 == i) || (j*2-1 == i)
                        if j*2 == i
                           deltamatrix(i,j) = -quantity (j);
                        else
                           deltamatrix(i,j) = quantity(j);
                        end;
                else
                deltamatrix(i,j) = 0;
                end;


            else

               deltamatrix(i,j) = 0;
           end;

           end;

        end;
    end;
end;



mastermatrix = [mastermatrix deltamatrix ycolumn zcolumn scolumn ocolumn];


Aeinq = [];

for i=1:NT
    Aeinq = [Aeinq mastermatrix];
end;

beinq = zeros(1,2*length(quantity))';
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Start-up and Shut-down Indicators
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 6*length(price);
m = length(Aeinq);

%Initialize first array to zero
Aeq2 = zeros(n,m);

for j=1:m
      for i=1:n
     if i == 1 && j == 1
        Aeq2(i,j) = -1;
        Aeq2(i+1,j) = 1;
     else
        if ((j == 2*length(price) +4) && i == 1)
        Aeq2(i,j) = 1;
        Aeq2(i,j+1) = -1;
        else
        if (i >1 && (j > 2*length(price)+7))
        Aeq2(i,j) = Aeq2(i-1,j-(2*length(price)+7));
        else
        end;
        end;

     end;


   end;
end;

beq2 = zeros(1,6*length(price))';

% Add to master equality arrays
Aeq = [Aeq; Aeq2];
beq = [beq; beq2];


% Make Inequality arrays
n = 6*length(price);
m = length(Aeinq);

%Initialize first array to zero
Aeinq2 = zeros(n,m);

for j=1:m
    for i=1:n
        if ((j == 2*length(price)+4) && i == 1)
            Aeinq2(i,j) = 1;
            Aeinq2(i,j+1) = 1;
        else
         if (i >1 && j > (2*length(price)+7))
            Aeinq2(i,j) = Aeinq2(i-1,j-(2*length(price)+7));
         else
         end;
        end;
    end;
end;

beinq2 = ones(1,6*length(price))';


% Add to master equality arrays
Aeinq = [Aeinq; Aeinq2];
beinq = [beinq; beinq2];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Capacity constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 2*NT;
m = length(Aeq);

%Initialize first array to zero
Aeq2 = zeros(n,m);

for j=1:m
     for i=1:n
    if i == 1 && j == 2
        Aeq2(i,j) = PMIN;
        Aeq2(i+1,j) = -PMAX;
        Aeq2(i,j+1) = 1;
        Aeq2(i+1,j+1) = -1;
    else
        if (i >2 && j > (2*length(price)+7))
        Aeq2(i,j) = Aeq2(i-2,j-(2*length(price)+7));
        else
        end;

    end;


  end;
end;

beq2 = zeros(1,2*NT)';

% Add to master equality arrays
Aeq = [Aeq; Aeq2];
beq = [beq; beq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reserves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = NT;
m = length(Aeq);

%Initialize first array to zero
Aeq2 = zeros(n,m);

for j=1:m
     for i=1:n
    if i == 1 && j == 1
        Aeq2(i,j) = QSC;
        Aeq2(i,j+2*length(price)+5) = -1;
        Aeq2(i,j+2*length(price)+6) = 1;
    else
        if (i >1 && j > (2*length(price)+7))
        Aeq2(i,j) = Aeq2(i-1,j-(2*length(price)+7));
        else
        end;

    end;


  end;
end;

beq2 = 15*ones(1,NT)';

% Add to master equality arrays
Aeq = [Aeq; Aeq2];
beq = [beq; beq2];
```

```
n = 3*NT;
m = length(Aeinq);

%Initialize first array to zero
Aeinq2 = zeros(n,m);

for j=1:m
     for i=1:n
      if i == 2 && j == 1
         Aeinq2(i,j) = -10*MSR;
         Aeinq2(i-1,j+1) = 1;
         Aeinq2(i-1,j+2*length(price)+5) =  1;
         Aeinq2(i,j+2*length(price)+5) =  1;
         Aeinq2(i+1,j+2*length(price)+5) =  -1;
      else
         if (i >3 && j > (2*length(price)+7))
         Aeinq2(i,j) = Aeinq2(i-3,j-(2*length(price)+7));
         else
         end;

      end;


   end;
end;
beinq3 = [PMAX 0 0];
beinq2 = [];
for i=1:NT
  beinq2 = [beinq2 beinq3];
end;
beinq2 = beinq2';

% Add to master equality arrays
Aeinq = [Aeinq; Aeinq2];
beinq = [beinq; beinq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ramping constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 2*NT;
m = length(Aeq);

%Initialize first array to zero
Aeq2 = zeros(n,m);

for j=1:m
      for i=1:n
      if i == 1 && j == 2
         Aeq2(i,j) = 1;
         Aeq2(i+1,j) = -1;
         Aeq2(i+2,j) = -1;
         Aeq2(i+3,j) = 1;
         Aeq2(i,j+2*length(price) + 2) = -(PMIN-RU);
         Aeq2(i+1,j+2*length(price) + 3) = -(PMIN-RD);

      else
         if (i >2 && j > (2*length(price)+7))
         Aeq2(i,j) = Aeq2(i-2,j-(2*length(price)+7));
         else
         end;

      end;


   end;
end;

beq3 = [RU RD];
beq2 = [];
```

```
for i=1:NT
   beq2 = [beq2 beq3];
end;

beq2 = beq2';



% Add to master equality arrays
Aeq = [Aeq; Aeq2];
beq = [beq; beq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% System Load Balance and Reserve Requirements
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
% Load Balance
%%%%%%%%%%%%%%%%%%%%%%%

n = NT;
m = length(Aeq);

%Initialize first array to zero
Aeq2 = zeros(n,m);

for j=1:m
      for i=1:n
     if i == 1 && j == 2
         Aeq2(i,j) = 1;
     else
         if (i >1 && j > (2*length(price)+7))
         Aeq2(i,j) = Aeq2(i-1,j-(2*length(price)+7));
         else
         end;

     end;


   end;
end;

beq2 = -[D]';

% Add to master equality arrays
Aeq = [Aeq; Aeq2];
beq = [beq; beq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spinning Reserve Balance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = NT;
m = length(Aeinq);

%Initialize first array to zero
Aeinq2 = zeros(n,m);

for j=1:m
      for i=1:n
     if i == 1 && j == 14
         Aeinq2(i,j) = -1;
     else
         if (i >1 && j > (2*length(price)+7))
         Aeinq2(i,j) = Aeinq2(i-1,j-(2*length(price)+7));
         else
         end;
```

```
      end;


   end;
end;

beinq2 = -[SR]';

% Add to master equality arrays
Aeinq = [Aeinq; Aeinq2];
beinq = [beinq; beinq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Operating Reserve Balance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


n = NT;
m = length(Aeinq);

%Initialize first array to zero
Aeinq2 = zeros(n,m);

for j=1:m
       for i=1:n
      if i == 1 && j == 15
         Aeinq2(i,j) = -1;
      else
         if (i >1 && j > (2*length(price)+7))
         Aeinq2(i,j) = Aeinq2(i-1,j-(2*length(price)+7));
         else
         end;

      end;


   end;
end;

beinq2 = -[OR]';

% Add to master equality arrays
Aeinq = [Aeinq; Aeinq2];
beinq = [beinq; beinq2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Upper Bounds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ub = [];
for i=1:length(price)
 ub = [ub inf];
end;

ub1 = [inf ub];

ub2 = ones(1,length(price));
ub2 = [1 ub2];

ub1 = [1 ub1 ub2 1 1 1 1];
ub = [];
for i=1:NT
   ub = [ub ub1];
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lower bounds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
lb = zeros(1,length(Aeinq));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xint array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = [];
for i=1:length(price)
  x(i) = 1;
end;

x2 = [];
for i = 1:length(price)
  x2(i) = 0;
end;

x1 = [1 0 x2 1 x 1 1 1 1];

x = [];
for i=1:NT
    x = [x x1];
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convex Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

else
    % Form convex function
    c = zeros(1,length(price));
    c(1) = 0;
    for i=1:length(c)
        c(i) = price(i);
    end;

    % generate p variables
    p = zeros(1,length(price));
    p(1) = 1;
    for i=1:length(p)
        p(i) = 1;
    end;

    % generate delta variables
    %  none needed in convex, but used for consistentcy
    d = zeros(1,length(price));


    u = zeros(1,1+length(d));
    f = zeros(1,length(u));
    %objective function
    f1 = [noload 0 c f startupcost shutdowncost 0 0];
    f = [];
    for i=1:NT
        f = [f f1];
    end;

    %equality constraints
    e = zeros(1,length(d));
    e = [0 -1 p 0 e 0 0 0 0];
    Aeq = []

    for i=1:NT
      Aeq = [Aeq e];
    end;
```

```
    beq = [0];

    %inequality constraints

    %make temp array
    temp = zeros(length(p));
    for i=1:length(p)
        for j=1:length(p)
        if i == j
            temp(i,j) = -1;
        else
            temp(i,j) = 0;
        end;
        end;
    end;

    temp = [temp];

    temp2 = zeros(length(p));
    for i=1:length(p)
        for j=1:length(p)
        if i == j
            temp2(i,j) = 1;
        else
            temp2(i,j) = 0;
        end;
        end;
    end;

    temp2 = [temp2];
    % This is the inequality matrix
    Aeinq = [temp; temp2];
    [m,n] = size(Aeinq);

    mcolumn= zeros(1,m)';
    e = [mcolumn mcolumn Aeinq mcolumn zeros(m,n) mcolumn mcolumn mcolumn mcolumn];
    Aeinq = [];
    for i=1:NT
     Aeinq = [Aeinq e];
    end;


    % make the b equations
    temp = zeros(1,length(quantity));
    beinq = [temp'; quantity'];
```

```
% create upper bound

ub = [];
for i = 1:length(price)+1
  ub(i) = inf;
end;

ub = [inf 1 ub];

% create lower bound
lb = zeros(1,length(Aeinq));
```

```
% create x array

x = [];
for i=1:length(price)+1
  x(i) = 0;
end;

x = [0 1 x];


end;

output=[];
```