TASK REPORT

Solving the economic dispatch problem for units with tabular data by curve fitting and curve-based methods and comparing the results with the result of the dynamic programming method.

Course: Power System Operation and Control
Lecturer: Prof. Taher
Student: Hamed Najafi

November 2022

Contents:

The order of the contents is in accordance with the steps that I personally went through while solving and investigating this problem.

I. Introducing Problem

Problem: Three thermal units with the following specifications are available to supply a power system:

Unit number	*Pmin(MW)*	*Pmax(MW)*
1	100	500
2	100	500
3	200	1000

	Fi:Cost of unit i ($/h)		
Pi(MW)	F1	F2	F3
0	∞	∞	∞
100	500	400	∞
200	950	1000	1020
300	1400	1440	1450
400	1840	1800	1900
500	2320	2400	2350
600	∞	∞	2800
700	∞	∞	3240
800	∞	∞	3680
900	∞	∞	4130
1000	∞	∞	4570

Assumptions:

1. All units are committed
2. Power system is lossless
3. Total Load Demand = 800 MW

So, the form of our problem can be shown as below:

$$\underset{P_1,P_2,P_3}{\text{Min}} [Objective\ Function] = F_{total} = \sum_{i=1}^{3} f_i(P_i)$$

$$\text{Such that} \begin{cases} 100 \leq P_1 \leq 500 \\ 100 \leq P_2 \leq 500 \\ 200 \leq P_3 \leq 1000 \\ P_1 + P_2 + P_3 = 800 \\ f_i\ and\ P_i\ values\ are\ in\ the\ above\ table \end{cases}$$

II. Solving Problem by Dynamic Programming

Since there are three units, there are two steps:

Step1: $f_2(\mathrm{D}) = \min\limits_{\{P_2\}}\{F_1(D-P_2) + F_2(P_2)\}$

	$F_1(D-P_2)+F_2(P_2)$											Optimum values		
Dj / P2:	0	100	200	300	400	500	600	700	800	900	1000	$f_2(Dj)$	P2*	P1*
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0
100	″	″	″	″	″	″	″	″	″	″	″	∞	0	100
200	″	900	″	″	″	″	″	″	″	″	″	900	100	100
300	″	1350	1500	″	″	″	″	″	″	″	″	1350	100	200
400	″	1800	1950	1940	″	″	″	″	″	″	″	1800	100	300
500	″	2240	2400	2390	2300	″	″	″	″	″	″	***2240***	***100***	***400***
600	″	2720	2840	2840	2750	2900	″	″	″	″	″	2720	100	500
700	″	∞	3320	3280	3200	3350	″	″	″	″	″	3200	400	300
800	″	″	∞	3760	3640	3800	″	″	″	″	″	3640	400	400
900	″	″	″	∞	4120	4240	″	″	″	″	″	4120	400	500
1000	″	″	″	″	∞	4720	″	″	″	″	″	4720	500	500

Step2: $f_3(\mathrm{D}) = \min\limits_{\{P_3\}}\{f_2(D-P_3) + F_3(P_3)\}$

	$f_2(D-P_3)+F_3(P_3)$											Optimum values	
Dj / P3:	0	100	200	**300**	400	500	600	700	800	900	1000	$f_3(Dj)$	P3*
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
100	″	″	″	″	″	″	″	″	″	″	″	″	0
200	″	″	″	″	″	″	″	″	″	″	″	″	0
300	″	″	″	″	″	″	″	″	″	″	″	″	0
400	″	″	1920	″	″	″	″	″	″	″	″	1920	200
500	″	″	2370	2350	″	″	″	″	″	″	″	2350	300
600	″	″	2820	2800	2800	″	″	″	″	″	″	2800	300
700	″	″	3260	3250	3250	3250	″	″	″	″	″	3250	300
800	″	″	3740	***3690***	3700	3700	3700	″	″	″	″	***3690***	***300***
900	″	″	4220	4170	4140	4150	4150	4140	″	″	″	4140	400
1000	″	″	4660	4650	4620	4590	4600	4590	4580	″	″	4580	800

Result:

Optimum operating point: {P_1 = 400, P_2 = 100, P_3 = 300, Ftotal = 3690}

And for future discussions, we will also bold four other nearby points:

Point A: {P_1 = 500, P_2 = 100, P_3 = 200, Ftotal = 3740}

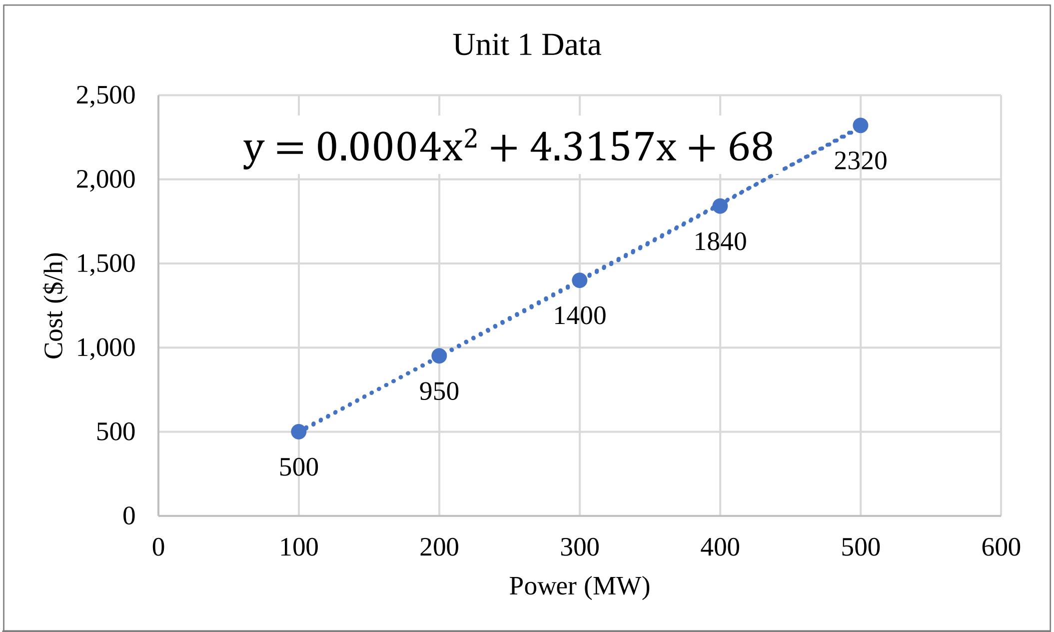
Point B: {P_1 = 300, P_2 = 100, P_3 = 400, Ftotal = 3700}

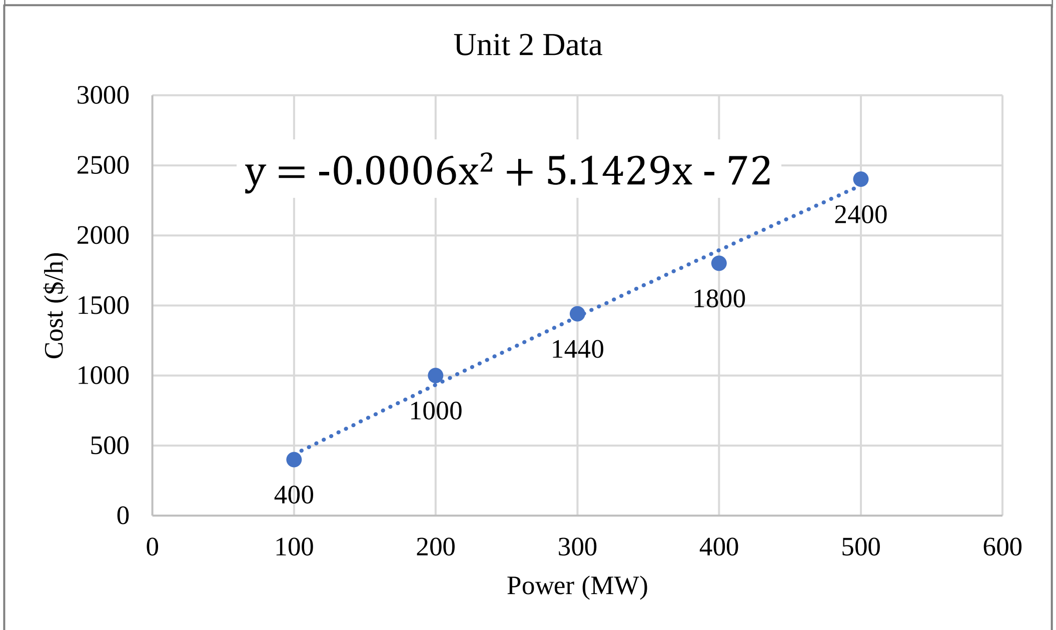
Point C: {P_1 = 200, P_2 = 100, P_3 = 500, Ftotal = 3700}

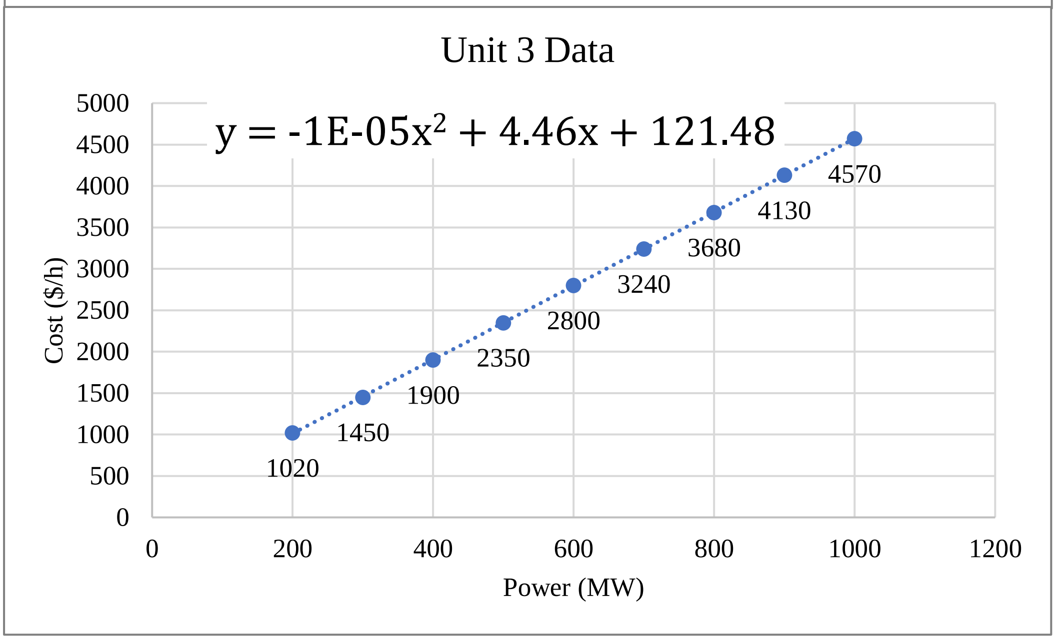
Point D: {P_1 = 100, P_2 = 100, P_3 = 600, Ftotal = 3700}

III. Fitting Curve on data

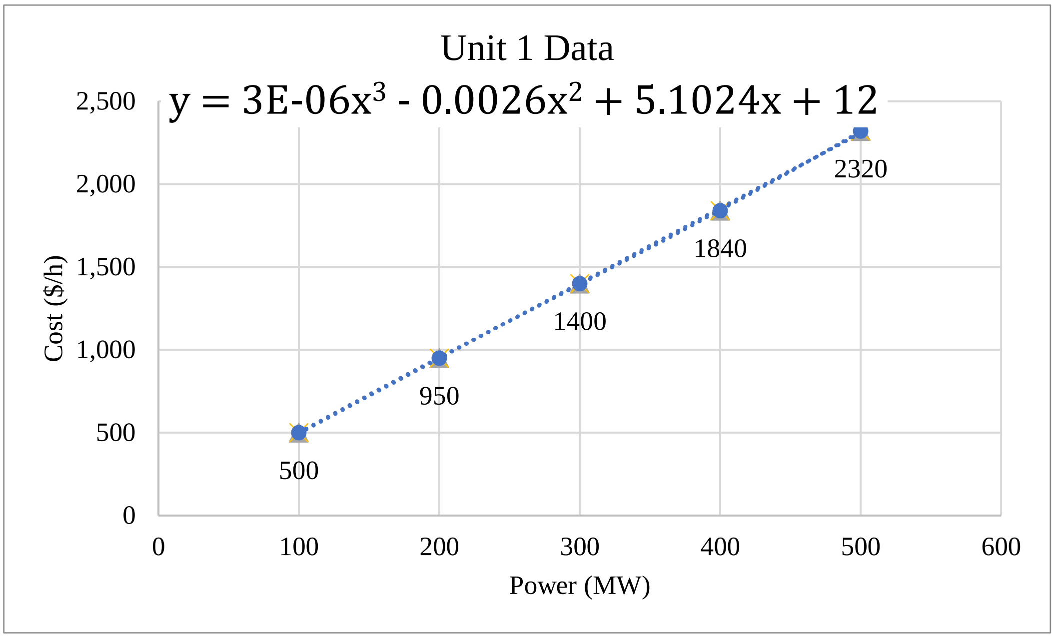
1. Fitting 2[nd] degree polynomial equations to the given data:

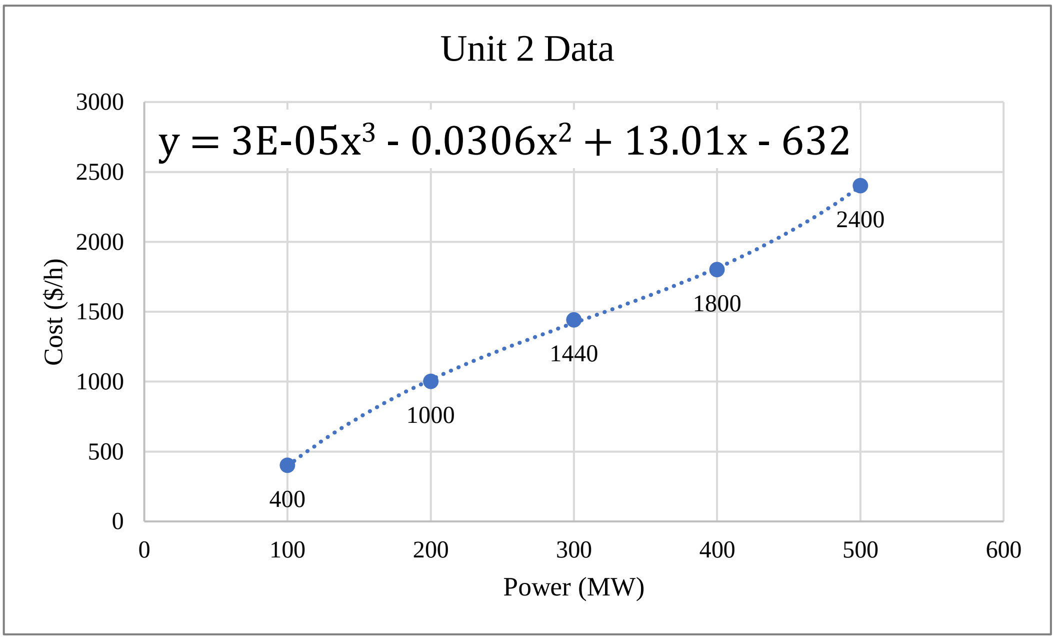


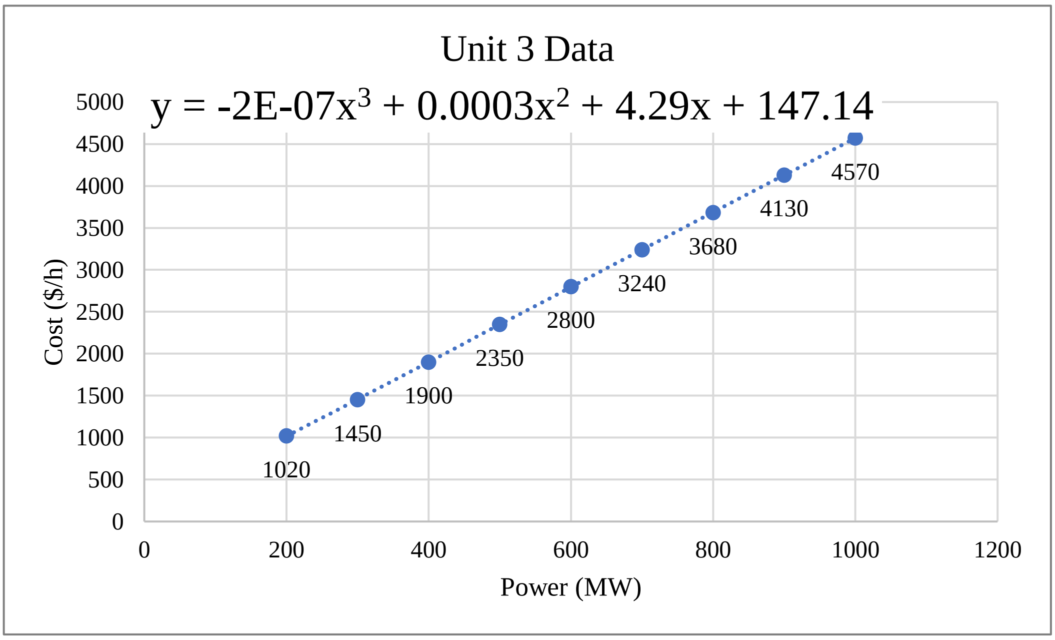




2. Fitting 3rd degree polynomial equations to the given data:







IV. Solving by Lagrange method using 2^{nd} degree polynomial functions

Using *Lagrange* method, we can continue:

```
% p1,p2,p3,L are symbolic variables and f1,f2,f3,df1,df2,df3,P1,P2,P3 are symbolic functions and ...
        ... L1,Ftot,Ftot_min are numerical variables

f1(p1) = 0.0003571.*p1.^2 + 4.316.*p1 + 68;          % Definition of Cost functions : fi(Pi)
f2(p2) = -0.0005714.*p2.^2 + 5.143.*p2 - 72;         % "
f3(p3) = (-9.74e-06).*p3.^2 + 4.46.*p3 + 121.5;      %"
df1=diff(f1,p1);                                % Evaluating Incremental Cost functions : f'i=λi(Pi)
df2=diff(f2,p2);                                % "
df3=diff(f3,p3);                                % "
P1(L)=(L-df1(0))/(df1(1)-df1(0));          % Evaluating Power of units as function of λ: Pi(λi)
P2(L)=(L-df2(0))/(df2(1)-df2(0));          % "
P3(L)=(L-df3(0))/(df3(1)-df3(0));          % "

L1=eval(solve(P1+P2+P3-800));              % Obtaining the equal incremental cost

p1_final=eval(P1(L1))                                % Evaluating P1 using found λ
p2_final=eval(P2(L1))                                % Evaluating P2 using found λ
p3_final=eval(P3(L1))                                % Evaluating P3 using found λ
Ftot=eval(f1(p1_final)+f2(p2_final)+f3(p3_final))    % Evaluating Total Cost

eval(f1(0)+f2(0)+f3(800))                            % Evaluating total cost of an arbitrary point
ftot_min=eval(f1(187.5)+f2(100)+f3(512.5))           % Evaluating total cost of an arbitrary point
```

Results:

---

p1_final = 201.6043
p2_final = 597.6673   Out of limit !
p3_final = 0.7284   Out of limit !

L1 = 4.4600   % (==>$\lambda_1 = \lambda_2 = \lambda_3 = 4.4600$)

Ftot =
  3.8751e+03

Evaluating Ftot=f1(0) + f2(0) + f3(800) =
  3.6793e+03

Evaluating ftot_min= eval(f1(187.5) + f2(100) + f3(512.5)) = ftot_min =
  3.7311e+03

P2 and P3 are out of limits so we should fix them on their bounds.

So, we consider p2_final=P2max=500 and p3_final=P3min=200 then p1_final=100

If we assume $\lambda_{std} = \lambda_{1\ new} = f'_1(P_1=100) = 4.3874$ and after calculations we have

$\lambda_{2\ new} = f'_2(P_2=500) = 4.5716$ and $\lambda_{3\ new} = f'_3(P_3=200) = 4.4561$

So, $\lambda_{3\ new} = \lambda_3|_{P3min} > \lambda_{std}$ shows unit three can be fixed on the minimum (200 MW).

However, $\lambda_{2\ new} = \lambda_2|_{P2max} > \lambda_{std}$ shows unit two is more expensive than unit one and it shouldn't be fixed on the maximum.

Therefore, we fix P3=200 MW and try to economically dispatch 600 MW of Load between unit one and two.

```
L2=eval(solve(P1+P2+200-800));
p1_final_2=eval(P1(L2))
p2_final_2=eval(P2(L2))
Ftot=eval(f1(p1_final_2)+f2(p2_final_2)+f3(200))
```

Results:

p1_final_2 = -329.7247	Out of limit !
p2_final_2 = 929.7247	Out of limit !
Ftot = 3.9125e+03	

This result is surprising. we continue with other methods and other aspects of the problem.

V. Solving by *Newton* method

By considering mentioned 2nd degree polynomial fitted equations we can write:

```
gendata = [68   4.316   0.0003571                  % Introducing equations coefficients
           - 72    5.143   -0.0005714              %"
           121.5    4.46   (-9.74e-06)];           %"
power = [800/3  800/3   800/3];                    % Initial guess for powers
Pload = 800;                                       % Total Load demand
n = length( gendata );

H = zeros(n+1,n+1);                                % Forming Hessian Matrix
for i = 1 : n                                      %"
H(i,i) = gendata(i,3) * 2;                         %"
H(i,n+1) = -1;                                     %"
H(n+1,i) = -1;                                     %"
end
H                                                  % Display Hessian Matrix
x0 = zeros(n+1,1);                                 % Forming Powers and Lambda vector
x0(1:n,1) = transpose( power );                    %"
x0(n+1,1)=(4.5065+4.8383+4.4548)/n;                %"
for kk = 1 : 3                                     % Forming Lagrange function gradient vector( ∇𝓛)
disp(kk)                                           % "
gradient = zeros(n+1,1);                           %"
gradient(n+1,1) = Pload;
   for i = 1 : n
   gradient(i,1) = gendata(i,2) + 2 * gendata(i,3) * x0(i,1) - x0(n+1,1);
   gradient(n+1,1) = gradient(n+1,1) - x0(i,1);
   end
dx =- H \ gradient;                                               % Calculating Δx vector
cost = 0;
   for i = 1 : n                                                  % Calculating total cost
   cost = cost + gendata(i,1) + gendata(i,2) * x0(i) + gendata(i,3) * x0(i) * x0(i);
   end
disp( [x0', cost/1000] )
x0 = x0 + dx;                                                     % Updating x vector
end
```

Result:

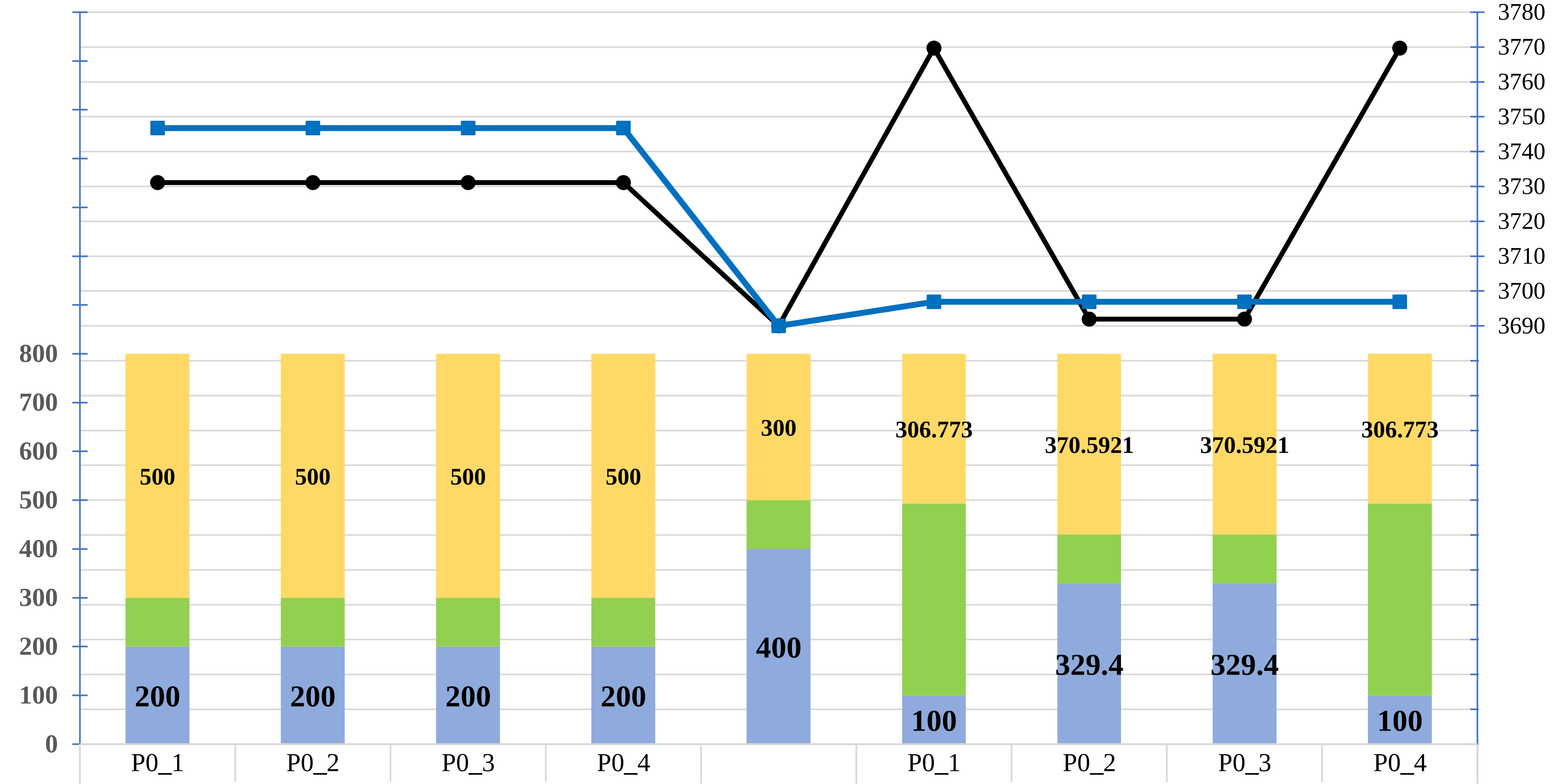
```
H =
   0.0007         0         0   -1.0000
        0   -0.0011         0   -1.0000
        0         0   -0.0000   -1.0000
  -1.0000   -1.0000   -1.0000         0

 1 → 266.6667  266.6667  266.6667   4.5999   3.8133
 2 → 201.6043  597.6673    0.7284   4.4600   3.8751
 3 → 201.6043  597.6673    0.7284   4.4600   3.8751
       P1          P2          P3         λ       Ft
```

=> It is exactly the same as the results of the previous section.

VI. Solving by Interior-point method

The following in chart results appeared after using 2nd and 3rd degree polynomial fitted equations, using the provided MATLAB function for finding minimum of constrained nonlinear multivariable function (*fmincon*) by its default algorithm (interior-point method), and taking into account four different initial guesses for power generation of each unit.

*DO: means Dynamic Optimization results.



	P0_1	P0_2	P0_3	P0_4		P0_1	P0_2	P0_3	P0_4
	2nd Order				DO(Table)	3rd order			
P3	500	500	500	500	300	306.773	370.5921	370.5921	306.773
P2	100	100	100	100	100	393.227	100	100	393.227
P1	200	200	200	200	400	100	329.4079	329.4079	100
f_t	3731.135	3731.135	3731.135	3731.135	3690	3769.687	3691.9096	3691.9096	3769.687
f_t(|DO)	3746.74	3746.74	3746.74	3746.74	3690	3696.94	3696.94	3696.94	3696.94

P0_i=[P0_1, P0_2, P0_3] P0_2=[500,200,100] P0_3=[400,100,300]

P0_1=[100,500,200]

P0_4=[800/3,800/3,800/3]

P0_2=[500,200,100] P0_3=[400,100,300]

P0_1=[100,500,200]

P0_4=[800/3,800/3,800/3]

When compared to the previous results, the result for 2nd degree equations appears to be more correct and accurate, especially when compared to *Figure 1* (on *Page 7*), and interestingly, this result is exactly the same as *point C* in the dynamic programming results (on *Page 2*), but the difference in total costs (3700-3731.135=-31.135) is due to curve fitting error. As a matter of fact, this is the first reliable feasible solution so far.

For 3rd degree equations it seems that the resulting surface is curved in space and has local minimum points. Points earned from P0_2 and P0_3 are very close to the *point B* in the dynamic programming results (on *Page 2*).

VII. Solving by Lambda Iteration method

We will only discuss 3rd degree equations from now on.

There were three difficulties in implementing the Lambda Iteration method. The first was to calculate P_i from λ_i because each equation has two different roots. The second was the initial guess for λ, which is critical in order to avoid divergence (it needs to be within one decimal number of the final value). The last one was the value of $\Delta\lambda$ when updating, for the same reason as before.

After some trial and error, it was determined that the lower roots should be used as a solution set for the first one. For the second one, previous section results were helpful, and for the last one, we couldn't use 10% of present value in the first iteration, so 0.1% of present value was used instead, and for other iterations, two methods of linear spotting and binary search were tested, and the binary search method was unable to even start the loop because of inappropriate initial guess for λ leads to complex roots for equations at the very first step and broke the loop.

Results:

Iteration	*λ*	*Total Generation (MW)*	*P1*	*P2*	*P3*
1	4.46	854.5569075	189.3935	216.4619	448.7015
2	4.45554	829.2814891	192.4328	216.7121	420.1366
3	4.450373	805.4855273	196.123	217.0029	392.3597
4	4.449182	800.588886	197.0024	217.07	386.5165
5	4.449039	800.0125246	197.1089	217.0781	385.8255
6	4.449036	800.000029	197.1112	217.0783	385.8105

Ftotal = 3.8710e+03

This result is also unsatisfactory because it costs more than all four points obtained using the previous methodology (interior-point method, on *Page 9*) At least, the calculated powers are within their allowed ranges, and $f_1(400) + f_2(100) + f_3(300) = 3.6971e+03$ would be the total cost of dynamic programming result if we wanted to compare this result with the dynamic programming result. that is less expensive, but if we figure out the incremental unit costs for D.P. results, we find:

$\lambda_1 = F'_1(400) = 4.5881$
$\lambda_2 = F'_2(100) = 7.8952$
$\lambda_3 = F'_3(300) = 4.4273$

Incremental costs of units are neither equal nor close to each other.

Figures below show the incremental cost functions and their value for the result of dynamic programming method.

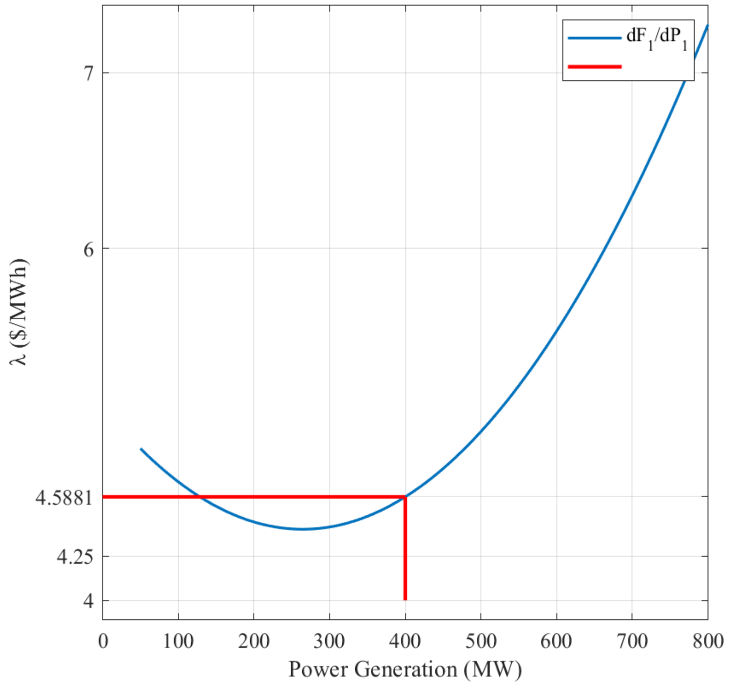


Figure 1: Incremental cost of unit 1

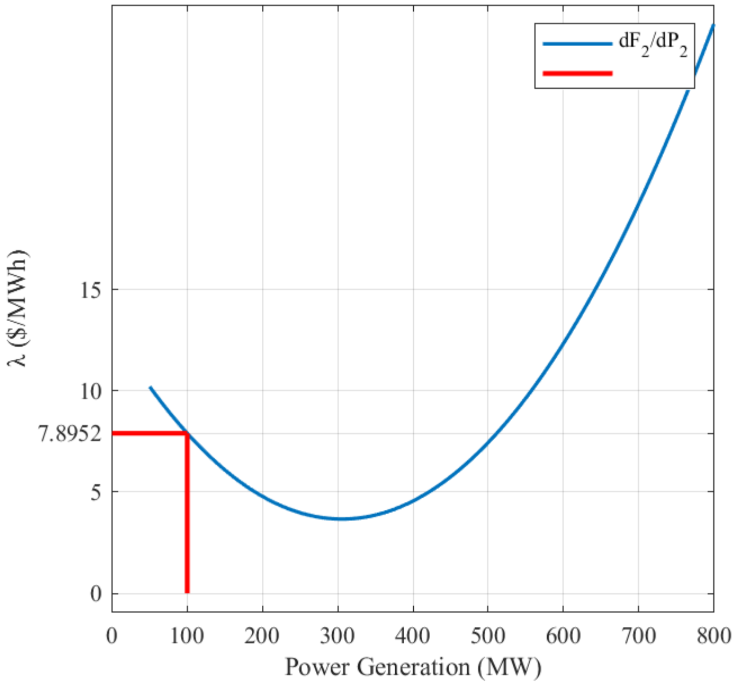


Figure 2: Incremental cost of unit 2

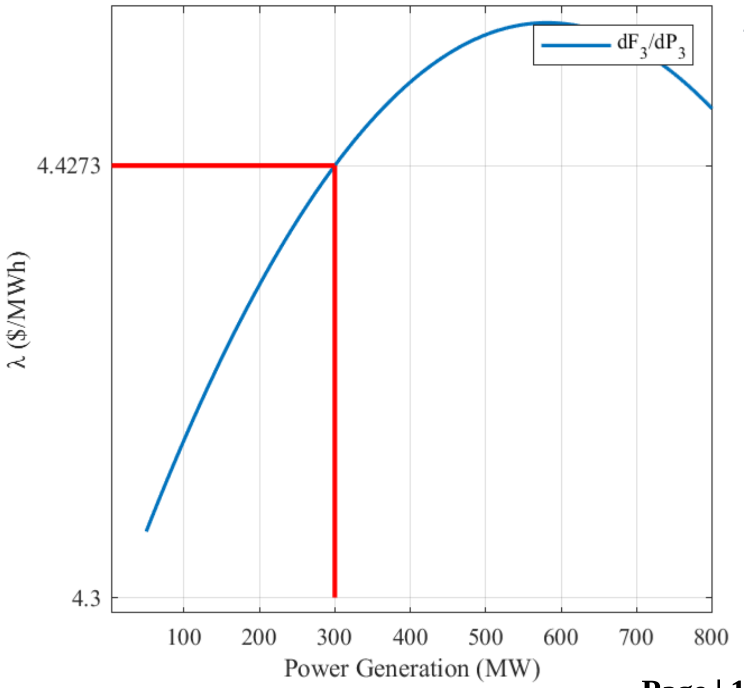


Figure 3: Incremental cost of unit 3

These final three figures demonstrate three ideas: 1. Units 1 and 2 have incremental cost functions that are convex, but unit 3 has cost function that is concave. 2. It was incorrect for us to use our adopted approach of using just lower roots of 2^{nd} degree equations (derivation of 3^{rd} degree cost functions) in our computations (in *Figure 1* the lower power for the same value of λ_1 is about 130 MW that is ignored). 3. We do not have equal incremental costs at the optimal point (which was known before by calculations).

VIII. Drawing total cost function surface

After going through all the mentioned steps, drawing could be a saving idea in order to better understand the behavior of the objective function and find the optimal point.

Notice that the total generation cost ($f_t = \sum_{i=1}^{3} f_i(P_i)$) is a function of three variables, and drawing its characteristics in three dimensions is impossible. To reduce the problem's dimensions, we can consider the generation of the third unit as a variable that is dependent on the generation of units one and two via the power balance constraint.

So, for all drawings of total cost in this report we consider:

P_1 = independent var. between 100 and 500 ($100 \leq P_1 \leq 500$)
P_2 = independent var. between 100 and 500 ($100 \leq P_2 \leq 500$)
$P_3 = 800 – (P_1 + P_2)$ ($P_1 + P_2 + P_3 = 800$)
($P_1 + P_2$) can't be greater than 600 (that implies "$200 \leq P_3$")

So $F_t(P_1, P_2) = f_1(P_1) + f_2(P_2) + f_3(800 - (P_1 + P_2))$

Now the total generation cost (objective function) is a function of two variables and can be drawn in a three-dimensional figure.
As the first drawing of objective function, we use 2nd degree cost equations:
The code below generates plot of the total cost curve according to the power of unit one and two:

```
figure
x=0:10:800;                  % Defining of power production points from 0 to 800 MW with a step of 10
[X, Y]=meshgrid(x,x);        % Generating a mesh Grid of P1(:X) and P2(:Y)
z=800-(X+Y);                 % Obtaining P3(:z) points for each (P1,P2) point on the mesh
w2nd=eval(f1(X)+f2(Y)+f3(z));          % Evaluating total generation cost for each (P1,P2) point on the mesh
mesh=X+Y;                    % defining a variable containing sum of (P1,P2)
X(mesh>800) =NaN;            % Removing Point of P1 which cause (P1+P2)>800
Y(mesh>800) =NaN;            % Removing Point of P2 which cause (P1+P2)>800
X(X<100) =NaN;               % Removing Points due to Boundary Constraints
Y(Y<100) =NaN;               % "
X(X>500) =NaN;               % "
Y(Y>500) =NaN;               % "
X(mesh>600) =NaN;            % "
Y(mesh>600) =NaN;            % "
s2=surf(X, Y,w2nd);          % Generating Plot
xlabel('P1 (MW)');                                       % Labeling
ylabel('P2 (MW)');                                       % "
zlabel("f1(P1) +f2(P2) +f3(800-(P1+P2)) - 2nd Degree");  % "
axis vis3d;
hold on;
plot3(187.5,100, f1(187.5) + f2(100) + f3(512.5),'d')   % Highlighting an arbitrary point that has a low cost
```

Result is shown in *Figure 4* on the next page.

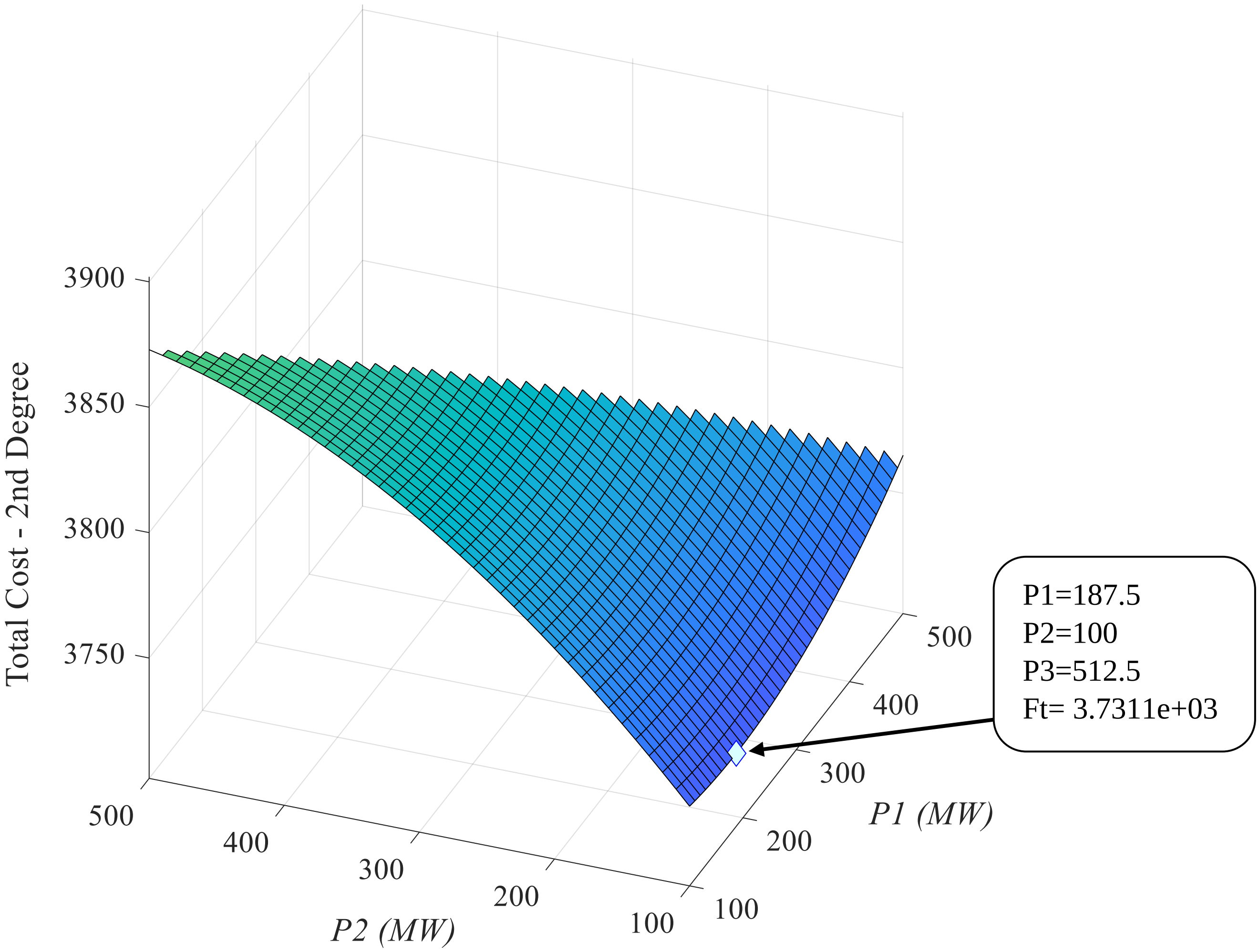


Figure 4: Total cost surface obtained from 2nd degree equations

With this result, all of failed attempts to finding an optimal minimum point make sense and the arbitrary point plotted by a diamond could be compared to results of all of 2nd degree total cost function minimizations at former steps.

Figure 4 shows that with the mentioned 2nd degree polynomial fitted equations, our minimization problem, actually, is a search for a minimum point on a piece of a whole surface with no bottom point where the gradient reaches zero. As a result, the *Lagrange* method and all other zero-gradient-based methods will be unappliable.

By replacing 3rd degree cost equations in the above code, the surface changes into the next plot.

Result is on the next page.

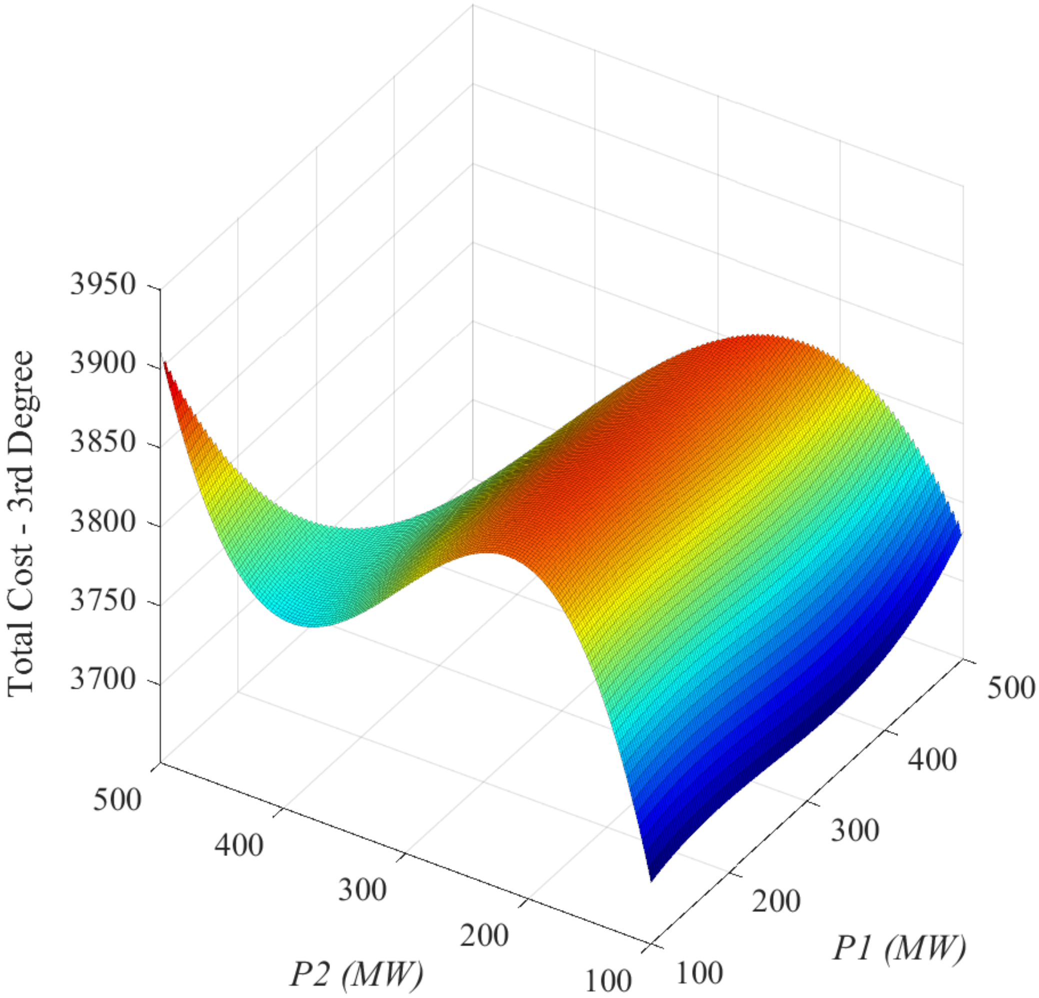


Figure 5: Total cost surface obtained from 3rd degree equations

This result is surprising too however may justify the former results. To clear doubts we can draw total cost real values specified by problem's tables with condition which in P_1 is constantly equal to 100 MW and P_2 varies from 100 to 500 MW (and P_3=800-P_1-P_2) then we see:

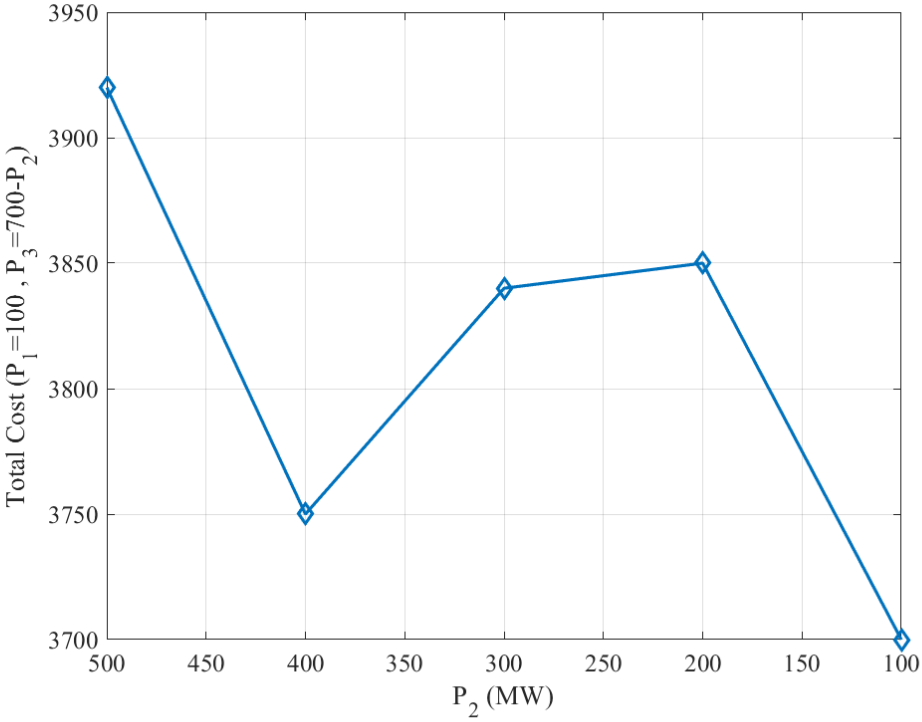


Figure 6: Total cost real values by considering constant value of P_1 at 100 MW

Figure 6 demonstrates that *Figure 5* is not out of normal or unacceptable, and if it was looking like anything other than the curve produced by the intersection of *Figure 5*'s surface with the P_1=100 plane, there would be cause for concern and further examination.

Because of considerable difference between *Figure 5* and *Figure 6* (existing a local minimum near the points where P_2=400) it would be informative to draw total cost functions obtained from 2nd and 3rd degree equations (with former conditions) in the *Figure 6* plot and perform a comparison.

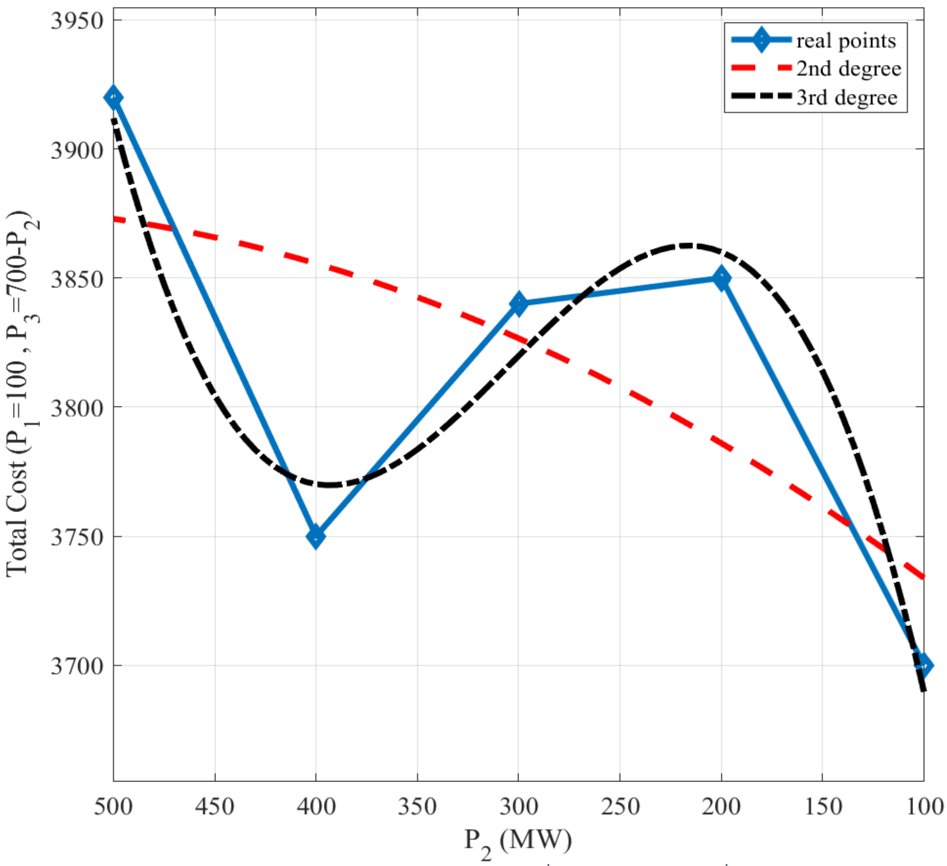


Figure 7: Total cost real values and 2nd degree and 3rd degree values by considering constant value of P_1 at 100 MW

Figure 7, while confirming the previous three-dimensional drawings (both *Figure 4* and *Figure 5*), justifies the previous results (such as the absence of a global minimum point with a zero gradient) and also shows that the 3rd degree equations are closer to reality, and the reason for having a local minimum in the 3rd degree surface as opposed to the 2nd degree equations is that the 2nd degree equations are unable to have the required curvature, which causes a lot of error (see *Figure 7*, where P_2=400) and the local minimum point (close to P_2=400) is exactly the point where the Lambda-Iteration method trapped in twice (see *Page 9*).

Figure 8 demonstrates real values of total cost.

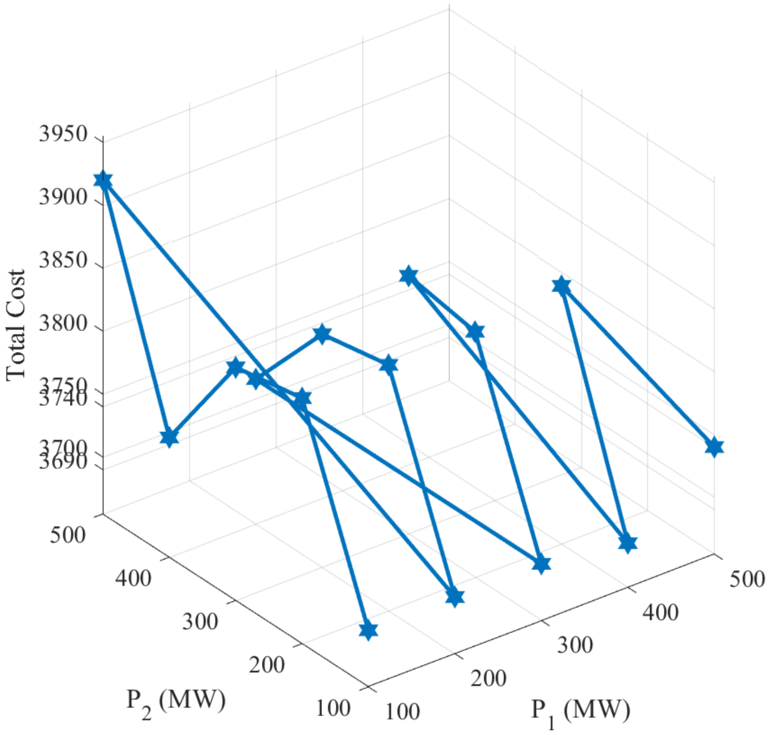


Figure 8: Total cost real values

By searching for a minimum value in the cost surface matrix it will be found that the minimum point (including powers as components) on the 3rd degree total cost surface is

P_1=100, P_2=100, P_3=600, *Ft*=3.6897e+03

That power components are equal to Point D on *Page 2*, and total cost calculated by 3rd degree equations for optimal point resulting from dynamic programming method is *Ft*=3.6969e+03. Its difference from the real value (3690) and the difference between the last found minimum cost and its real value for same power components (3700) are due to curve fitting error.

IX. Conclusion

It was anticipated that after using a dynamic programming methodology, we would perform straightforward curve fitting procedures and use an equation-based method to arrive at results and numbers that were same or almost so. However, the results were not even close, and using other methods did not help. As a result, the only option left was to draw the objective function in order to determine why our findings deviate from the objective result and to identify the optimal point and to find a justification for that odd behavior. Drawn shapes were also unexpected and required additional research and study that was conducted and mentioned.

After all it could be said that this problem has a different nature from the other problems that we have encountered so far, and feasible solutions surface has no bottom point where the gradient reaches zero. As a result, the conventional methods are ineffective, and it appears that the only option available to us is to numerically calculate the total cost surface (or space) for various amounts of possible points ($P_1, P_2, \ldots, P_{n-1}$) and find the minimum point in resulting $(n-1)$-dimensional matrix that obtains corresponding points too (as done for previous page).

End of Report