**2)** Initial table of assignment problem is as follows:

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 50 | 70 | 90 | 70 | 60 |
| T2 | 50 | 40 | 30 | 20 | 45 |
| T3 | * | 20 | 50 | 60 | 30 |
| T4 | 20 | 30 | * | 70 | 80 |

Now, make the table square to apply Hungarian Algorithm. We add a new row filled with 0s:

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 50 | 70 | 90 | 70 | 60 |
| T2 | 50 | 40 | 30 | 20 | 45 |
| T3 | * | 20 | 50 | 60 | 30 |
| T4 | 20 | 30 | * | 70 | 80 |
| T5 | 0 | 0 | 0 | 0 | 0 |

Subtract min elements in each rows:

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 0 | 20 | 40 | 20 | 10 |
| T2 | 30 | 20 | 10 | 0 | 25 |
| T3 | * | 0 | 30 | 40 | 10 |
| T4 | 0 | 10 | * | 50 | 60 |
| T5 | 0 | 0 | 0 | 0 | 0 |

Since all columns have 0 as their minimum, there's no column reduction.

As the next step, cover 0s with as few lines as possible

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 0 | 20 | 40 | 20 | 10 |
| T2 | 30 | 20 | 10 | 0 | 25 |
| T3 | * | 0 | 30 | 40 | 10 |
| T4 | 0 | 10 | * | 50 | 60 |
| T5 | 0 | 0 | 0 | 0 | 0 |

Optimality check: lines required<n, we need to create new 0s. Smallest uncovered one should be subtracted from all uncovered elements and be added to all elements covered by 2 lines

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 0 | 10 | 30 | 10 | 0 |
| T2 | 40 | 20 | 10 | 0 | 25 |
| T3 | * | 0 | 30 | 40 | 10 |
| T4 | 0 | 0 | * | 40 | 50 |
| T5 | 10 | 0 | 0 | 0 | 0 |

Optimality Check: Line number = n

And, replace the lines to provide proper coverage:

Then, assign one contractor to one tender at 0 elements (colored with red)

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 0 | 10 | 30 | 10 | 0 |
| T2 | 40 | 20 | 10 | 0 | 25 |
| T3 | * | 0 | 30 | 40 | 10 |
| T4 | 0 | 0 | * | 40 | 50 |
| T5 | 10 | 0 | 0 | 0 | 0 |

The corresponding elements at initial table(the one at which we added 0 column):

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| T1 | 50 | 70 | 90 | 70 | 60 |
| T2 | 50 | 40 | 30 | 20 | 45 |
| T3 | * | 20 | 50 | 60 | 30 |
| T4 | 20 | 30 | * | 70 | 80 |
| T5 | 0 | 0 | 0 | 0 | 0 |

T1->C5, T2->C4, T3->C2, T4->C1.

Arbitrary T5 is won by C3. So, C3 has no tenders in this assignment.

So, minimum cost of assignment is 60+20+20+20=120.

**3)**

$$\rho = \frac{\text{Length of whole line}}{\text{Length of larger part of line}} = \frac{\text{length of larger part of line}}{\text{length of smaller part of line}}$$

$$1 - p = p^2$$

$$p^2 + p - 1 = 0$$

$$\Delta = 1 - 4 \times 1 \times (-1) = 1 + 4 = 5$$

$$x_{1,2} = \frac{-1 \pm \sqrt{5}}{2}$$

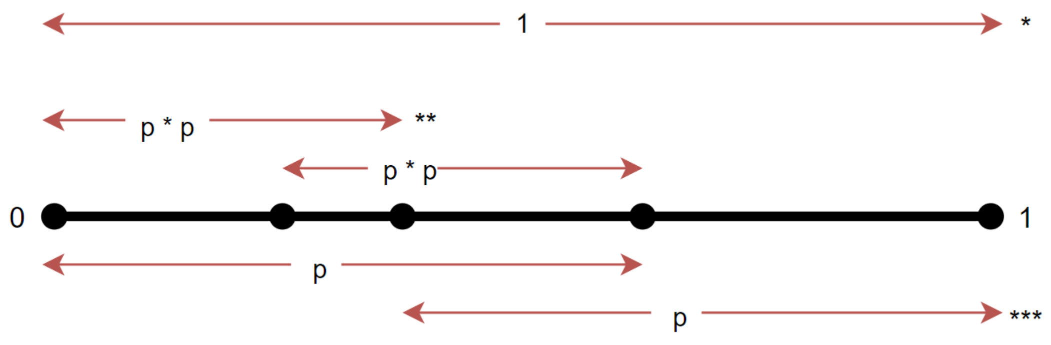
$$x_1 = \frac{\sqrt{5} - 1}{2}$$

$$x_2 = \frac{-\sqrt{5} - 1}{2}$$

**Length of a line can not be negative value. So, $x_2$ can not be a meaningful root for our discussion**

$$\rho = \frac{\sqrt{5} - 1}{2}$$

**Also we want the property in such a way that second iteration will use the data we created in first iteration. The property can be examined below:**



**From the graph, if we use *, **, ***:**

$$p^2 + p = 1$$

**After this result we are able to come up with the same solution we reached before.**

## 4)

**cost of a console: 250$**
**x: the price of a console**
**profit from game sales per user: 100$**
**demand function $S(x)$:**

$$\frac{80 \times x^2}{3} - 24400 \times x + \frac{17260000}{3}$$

$$total\_profit(x) = S(x) \times (x - cost + game\_profit$$

$$f(x) = S(x) \times (x - 150)$$

$$maximize : f(x) = \frac{80 \times x^3}{3} - 28400 \times x^2 + \frac{28240000 \times x}{3} + 863000000$$

**for case1:**
**first find local maximas:**

$$f'(x) = 0 \qquad and \qquad f''(x) < 0$$

$$f'(x) = 80 \times x^2 - 56800 \times x + \frac{28240000}{3}$$

$$f''(x) = 160 \times x - 56800$$

$$f'(x) = 0 \rightarrow 80 \times x^2 - 56800 \times x + \frac{28240000}{3} = 0$$

$$3 \times x^2 - 2130 \times x + 353000 = 0$$

$$\Delta = (-2130)^2 - 4 \times 3 \times 353000$$

$$\Delta = 300900$$

$$x_{1,2} = \frac{2130 \pm \sqrt{300900}}{6}$$

$$x_1 = 446,4239$$

$$x_2 = 263,5761$$

**We are asked to find a value inside the $200 \leq x \leq 400$. So, we need to test the second derivative of the only second value**

$$f''(x_2) = -14627,824$$

**Our constraints are satisfied for only one value for Case1**

**for case2:**
**We have no breaking points, first and second derivative of $f(x)$ are exists. So, we don't have any candidate point according to case2.**

**for case3:**
**A: the left cut point of the interval, 200**

**B: the right cut point of the interval, 400**
**We need to check:**

$$f'(A) < 0?$$

$$f'(B) > 0?$$

$$f'(A) = f'(200) = 80 \times 200^2 - 56800 \times 200 + \frac{28240000}{3} = 1253333.3$$

**A failed the test**

$$f'(B) = f'(400) = 80 \times 400^2 - 56800 \times 400 + \frac{28240000}{3} = -506666.6$$

**B also failed the test**

**Our optimal point is 263,5761.**

## 5)

We take 4 subinterval of [-4, 4] in which f is locally convex and run the bisection algorithm for those intervals.

Outputs of program:

***Interval 1:***

```
Initial point: (-3,8.3411) and search interval is: [-4, -2.036]
Step: 0 Search interval is: [-4, -2.036]       Point is: (-3.018,8.31335)
Step: 1 Search interval is: [-4, -3.018]       Point is: (-3.509,8.37231)
Step: 2 Search interval is: [-3.509, -3.018]    Point is: (-3.2635,8.15523)
Step: 3 Search interval is: [-3.2635, -3.018]   Point is: (-3.14075,8.18216)
Step: 4 Search interval is: [-3.2635, -3.14075]        Point is: (-3.20212,8.15577)
Step: 5 Search interval is: [-3.2635, -3.20212]        Point is: (-3.23281,8.15233)
Step: 6 Search interval is: [-3.2635, -3.23281]        Point is: (-3.24816,8.153)
Step: 7 Search interval is: [-3.24816, -3.23281]       Point is: (-3.24048,8.15247)
Step: 8 Search interval is: [-3.24048, -3.23281]       Point is: (-3.23665,8.15235)
Step: 9 Search interval is: [-3.23665, -3.23281]       Point is: (-3.23473,8.15233)
Step: 10 Search interval is: [-3.23473, -3.23281]      Point is: (-3.23377,8.15233)
Step: 11 Search interval is: [-3.23473, -3.23377]      Point is: (-3.23425,8.15233)
Step: 12 Search interval is: [-3.23425, -3.23377]      Point is: (-3.23401,8.15233)
Step: 13 Search interval is: [-3.23425, -3.23401]      Point is: (-3.23413,8.15233)
Step: 14 Search interval is: [-3.23413, -3.23401]      Point is: (-3.23407,8.15233)
Algorithm ended
Final local optimum point is: -3.23407 and value of that point is: 8.15233
```

***Interval 2***:

```
Initial point: (-1.5,9.59136) and search interval is: [-2.036, 0]
Step: 0 Search interval is: [-2.036, 0]        Point is: (-1.018,9.08925)
Step: 1 Search interval is: [-2.036, -1.018]    Point is: (-1.527,9.64675)
```

```
Step: 2 Search interval is: [-1.527, -1.018]    Point is: (-1.2725,9.20124)
Step: 3 Search interval is: [-1.2725, -1.018]   Point is: (-1.14525,9.09175)
Step: 4 Search interval is: [-1.14525, -1.018]  Point is: (-1.08162,9.07646)
Step: 5 Search interval is: [-1.08162, -1.018]  Point is: (-1.04981,9.07932)
Step: 6 Search interval is: [-1.08162, -1.04981]        Point is: (-1.06572,9.077)
Step: 7 Search interval is: [-1.08162, -1.06572]        Point is: (-1.07367,9.07651)
Step: 8 Search interval is: [-1.08162, -1.07367]        Point is: (-1.07765,9.07643)
Step: 9 Search interval is: [-1.08162, -1.07765]        Point is: (-1.07964,9.07643)
Step: 10 Search interval is: [-1.07964, -1.07765]       Point is: (-1.07864,9.07642)
Step: 11 Search interval is: [-1.07964, -1.07864]       Point is: (-1.07914,9.07642)
Step: 12 Search interval is: [-1.07914, -1.07864]       Point is: (-1.07889,9.07642)
Step: 13 Search interval is: [-1.07889, -1.07864]       Point is: (-1.07877,9.07642)
Step: 14 Search interval is: [-1.07877, -1.07864]       Point is: (-1.0787,9.07642)
Algorithm ended
Final local optimum point is: -1.0787 and value of that point is: 9.07642
```

***Interval 3:***

```
Initial point: (0.5,10.0366) and search interval is: [0, 2.039]
Step: 0 Search interval is: [0, 2.039]  Point is: (1.0195,9.10902)
Step: 1 Search interval is: [1.0195, 2.039]     Point is: (1.52925,9.68198)
Step: 2 Search interval is: [1.0195, 1.52925]   Point is: (1.27438,9.22905)
Step: 3 Search interval is: [1.0195, 1.27438]   Point is: (1.14694,9.11547)
Step: 4 Search interval is: [1.0195, 1.14694]   Point is: (1.08322,9.09816)
Step: 5 Search interval is: [1.0195, 1.08322]   Point is: (1.05136,9.10004)
Step: 6 Search interval is: [1.05136, 1.08322]  Point is: (1.06729,9.09822)
```

```
Step: 7 Search interval is: [1.06729, 1.08322]  Point is: (1.07525,9.09797)
Step: 8 Search interval is: [1.07525, 1.08322]  Point is: (1.07924,9.09801)
Step: 9 Search interval is: [1.07525, 1.07924]  Point is: (1.07725,9.09797)
Step: 10 Search interval is: [1.07525, 1.07725]        Point is: (1.07625,9.09797)
Step: 11 Search interval is: [1.07525, 1.07625]        Point is: (1.07575,9.09797)
Step: 12 Search interval is: [1.07525, 1.07575]        Point is: (1.0755,9.09797)
Step: 13 Search interval is: [1.0755, 1.07575]  Point is: (1.07563,9.09797)
Step: 14 Search interval is: [1.07563, 1.07575]        Point is: (1.07569,9.09797)
Algorithm ended
Final local optimum point is: 1.07569 and value of that point is: 9.09797
```

***Interval 4:***

```
Initial point: (2.5,9.67731) and search interval is: [2.039, 4]
Step: 0 Search interval is: [2.039, 4]  Point is: (3.0195,8.37152)
Step: 1 Search interval is: [3.0195, 4]        Point is: (3.50975,8.44358)
Step: 2 Search interval is: [3.0195, 3.50975]   Point is: (3.26462,8.22075)
Step: 3 Search interval is: [3.0195, 3.26462]   Point is: (3.14206,8.24417)
Step: 4 Search interval is: [3.14206, 3.26462]  Point is: (3.20334,8.21958)
Step: 5 Search interval is: [3.20334, 3.26462]  Point is: (3.23398,8.21701)
Step: 6 Search interval is: [3.20334, 3.23398]  Point is: (3.21866,8.2175)
Step: 7 Search interval is: [3.21866, 3.23398]  Point is: (3.22632,8.21705)
Step: 8 Search interval is: [3.22632, 3.23398]  Point is: (3.23015,8.21698)
Step: 9 Search interval is: [3.23015, 3.23398]  Point is: (3.23207,8.21698)
Step: 10 Search interval is: [3.23015, 3.23207]        Point is: (3.23111,8.21698)
Step: 11 Search interval is: [3.23015, 3.23111]        Point is: (3.23063,8.21698)
Step: 12 Search interval is: [3.23063, 3.23111]        Point is: (3.23087,8.21698)
```

```
Step: 13 Search interval is: [3.23087, 3.23111]        Point is: (3.23099,8.21698)

Step: 14 Search interval is: [3.23087, 3.23099]        Point is: (3.23093,8.21698)

Algorithm ended

Final local optimum point is: 3.23093 and value of that point is: 8.21698
```

We have 4 local optimal points and the smallest of them is the global minimum of the given interval.

Global optimum point is: -3.23407 and value of that point is: 8.15233

## 6)

Function is $f(x_1, x_2) = (5x_1 - x_2)^4 + (x_1 - 2)^2 + x1 - 2x_2 + 12$

Initial point is $(x_1, x_2) = (6.35,33.01)$

Epsilon = 0.00001

Limits in bisection are a = -5, b = 12

Epsilon in bisection = Epsilon$^2$

I implemented the pseudo code correctly by using Python3 without importing any package, then made long numerical analyses to find a suitable starting point as the algorithm is heavily dependent on starting point and epsilon

## Output of the program printed to terminal:

```
x(k):  [6.44046881237031, 32.99208533923872]

f(x(k)):  -27.436947773227942

iteration k:  1

x(k):  [6.082357861796569, 33.349457229974504]

f(x(k)):  42.524139095153345

iteration k:  2

x(k):  [6.494752790373269, 33.26711662041133]
```

```
f(x(k)):  -27.440522798531426

iteration k:  3

x(k):  [6.462976713678614, 33.29865844195844]

f(x(k)):  -27.279517016834234

iteration k:  4

x(k):  [6.499547007543444, 33.29140379150062]

f(x(k)):  -27.44055057995852

iteration k:  5

x(k):  [6.496008436164296, 33.29428586956791]

f(x(k)):  -27.438912012395285

iteration k:  6

x(k):  [6.4999611564132085, 33.29350316784991]

f(x(k)):  -27.44055078743005

iteration k:  7

x(k):  [6.499469102559037, 33.293788027837344]

f(x(k)):  -27.44052202345256

iteration k:  8

x(k):  [6.499996773809151, 33.29368356016564]

f(x(k)):  -27.440550788963108

iteration k:  9

x(k):  [6.499695532107788, 33.293759294221985]

f(x(k)):  -27.44054123468466

iteration k:  10

x(k):  [6.500000016772926, 33.29369901031928]

f(x(k)):  -27.44055078896649

iteration k:  11

x(k):  [6.499274114586207, 33.293844110246575]

f(x(k)):  -27.440496284090514

iteration k:  12
```

```
x(k):  [6.5000003975055884, 33.2937003105982]

f(x(k)):  -27.44055078895765

iteration k:  13

x(k):  [6.498991681034815, 33.29390014586553]

f(x(k)):  -27.440445483065908

iteration k:  14

x(k):  [6.500000472499336, 33.29370040746508]

f(x(k)):  -27.440550788952663

iteration k:  15

x(k):  [6.498863817827687, 33.2939254704026]

f(x(k)):  -27.440417008031403

iteration k:  16

x(k):  [6.500000359478199, 33.29370043587219]

f(x(k)):  -27.440550788962547

iteration k:  17

x(k):  [6.499135616366868, 33.2938716589991]

f(x(k)):  -27.44047344944159

iteration k:  18

x(k):  [6.500000281364681, 33.293700458918956]

f(x(k)):  -27.440550788967855

iteration k:  19

x(k):  [6.499325015900659, 33.29383416146129]

f(x(k)):  -27.44050366734858

iteration k:  20

x(k):  [6.500000251196011, 33.29370046933573]

f(x(k)):  -27.440550788969574

iteration k:  21

x(k):  [6.4993988525327495, 33.29381954332754]

f(x(k)):  -27.440513424683672
```

```
iteration k:  22

x(k):  [6.499999460729827, 33.293700627450356]

f(x(k)):  -27.44055078894627

iteration k:  23

x(k):  [6.5000000330036904, 33.293700514151105]

f(x(k)):  -27.440550788976026

iteration k:  24

normal of gradient:  6.8816823913503395e-06

normal is smaller than epsilon

result:  [6.5000000330036904, 33.293700514151105]

value of the resulting point in function f:  -27.440550788976026
```

Comment:

So, as seen by the output of the program, steepest descent function lasted for 24 iterations until the normal of gradient of function value at the resulting point is smaller than the epsilon.