Numerical Methods Assignment № 5

Ebrar Ömer , Istanbul Technical University 14/04/2020

All graphs and tables are designed in Latex. All codes written using Jupyter Notebook.

1 Comparison of methods

Each method is explained with both mathematical solutions and python codes. Written python codes are based on same example for all methods, the solutions have been compared with same 3 initial values and in same 2 intervals, Mathematical solutions are different for each method to show methods' properties.

1.1 Bisection method:

*Bisection method has simple algorithm, easy to understand and implement.However, it may be needed large number of evaluations.

*It is based on safe rules like intermediate value theorem so it is robust.

*It is not require many additional data,continuity of the function is enough.

*But it is slow, reaching the stopping criteria could take time, so it is hard to generalize to system. Also if the function has more than one root, it can not be implement.

1.1.1 Example:

$$f(x) = x^3 + 4x^2 - 10 = 0$$

Find the root of this equation on the interval [1 , 2] using bisection method with an error smaller than 10^-4.

$$x_L = 1, x_R = 2$$

$$y_L = f(1.5) = 5$$

$$y_R = f(2) = 14$$

$$y_L.y_m < 0$$

that means it has root in this interval.

$$x_M = x_L + x_R/2 = 1.5$$

$$y_M = f(1.5) = 2.375$$

$$y_L.y_m < 0 -> x_L = x_M$$

$$x_M = 1.5 + 1/2 = 1.25$$

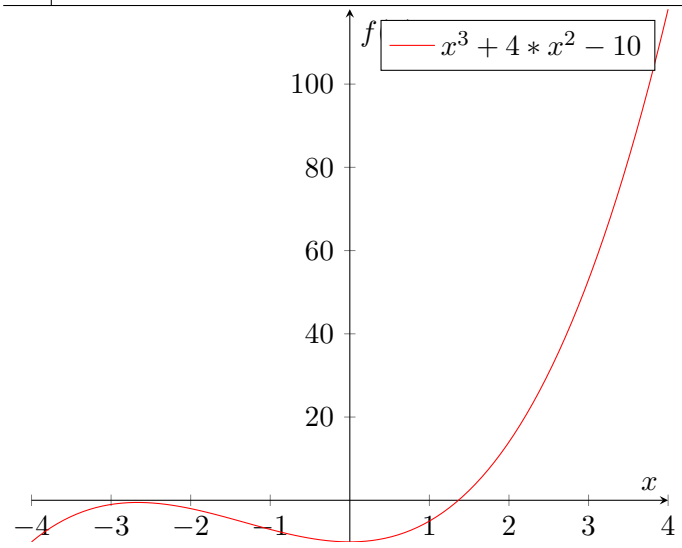
$$y_M = f(1.25) = -1.796875$$

$$x_M = 1.25 + 1.5/2 = 1.375$$

$$y_m = f(1.375) = 0.16211$$

Let's show the iterative on a table:

n	a_n	p_n	b_n	$f(p_n)$
1	1.0	1.5	2.0	2.375
2	1.0	1.25	1.4	-1.796875
3	1.25	1.375	1.5	0.1612109375
4	1.3125	1.34375	1.375	-0.8483886719
5	1.3125	1.34375	1.375	-0,350982666
6	1.34375	1.359375	1.375	-0.09640884399
7	1.359375	1.3671875	1.3671875	0.032325578537
8	1.359375	1.36328125	1.3671875	-0.03214997053
9	1.36328125	1.365234375	1.3671875	0.00007202476263



1.1.2 Example:

$$f(x) = x^3 - 7x^2 + 14x - 6 = 0$$

Find the root of this equation with smaller error than 10^-2 on the interval [0,1] with bisection method.(Real root = $2 - \sqrt{2}$)

$$x_L = 0, x_R = 1$$

$$y_L = f(0) = 6$$

$$y_R = f(1) = 2$$

$$y_L . y_m < 0$$

that means it has root in this interval.

$$x_M = x_L + x_R/2 = 0,5$$

$$y_M = f(0.5) = -0.625$$

Stopping criteria : $|x - x_n| < Error$

Error : $|2 - \sqrt{2} - 0.5| = 0.0858 > 0.01$, so continue...

$$x_M = 0.5 + 1/2 = 0.75$$

$$y_M = f(0.75) = 0.984375$$

Stopping criteria : $|x - x_n| = |2 - \sqrt{2} - 0.75| = 0.1642 > 0.01$ so continue...

$$x_M = 0.5 + 0.75/2 = 0.625$$

$$y_m = f(0.625) = 0.259765$$

Stopping criteria : $|x - x_n| = |2 - \sqrt{2} - 0.625| = 0.039 > 0.01$ so continue...

$$x_M = 0.5 + 0.625/2 = 0.5625$$

$$y_m = f(0.5625) = -0,16186523$$

Stopping criteria : $|x - x_n| = |2 - \sqrt{2} - 0.5625| = 0.02 > 0.01$ so continue...

Even though we've done 4 iteration it still has not reached the stopping criteria because it converges so slow.

1.1.3 Base Example

This is the base example which will solve with every methods. For the equation $f(x) = cosx - x = 0$do iterative with using Bisection method. Initial guesses are: $x_L = 0$ and $x_R = 1$ Stopping criteria is smaller error than 10^{-3}

We discussed about how to make iterative, so let's skip and examine in table.

n	x_0	$f(x_0)$	x_1	$f(x_1)$	x_m	$f(x_m)$
1	0	1	1	-0,4597	0,5	0,3776
2	0,5	0,3776	1	-0,4597	0,75	-0.0183
3	0,5	0,3776	0,75	-0.0183	0,625	0,186
4	0,625	0,186	0,75	-0.0183	0,6875	0,0853
5	0,6875	0,0853	0,75	-0.0183	0,7188	-0.0339
6	0,7188	-0.0339	0,75	-0.0183	0,7344	0,0079
7	0,7344	0,0079	0,75	-0.0183	0,7422	-0,0052
8	0,7344	0,0079	0,7422	-0,0052	0,7383	0.013
9	0,7383	0.013	0,7422	-0,0052	0,7402	-0.019
10	0,7383	0.013	0,7402	-0.019	0,7393	-0.0003

After 10 iterations, we've found approximate solution.

Listing 1: Sample Python code for example 1.1.3

```
from math import cos,ceil

def bisect(f,x1,x2,epsilon=1.0e-3):
    f1 = f(x1)
    if f1 == 0.0: return x1
    f2 = f(x2)
    if f2 == 0.0: return x2
    i=0
    while(True):
        i=i+1
        x3 = 0.5*(x1 + x2);
        f3 = f(x3)
        if ceil(f3) == 0.0 and (abs(x3-x2)<epsilon or abs(x3-x1)<epsilon):
            return x3,i
        if f2*f3 < 0.0:
            x1 = x3; f1 = f3
        else:
            x2 =x3;  f2 = f3

def f(x): return cos(x) - x

a,b = (0.0, 1.0)
x1 = a; f1 = f(a)
x2 = b; f2 =f(x2)
if f1*f2 > 0.0:
    print("No root in this interval.")
else:
    root,it = bisect(f,x1,x2)
    print ("The root is:",root)
    print("Number of iteration: ",it)
```

Listing 2: Sample Bash code and Output.

```
The root is: 0.7392578125
Number of iteration:  10
```

If you replace the intervals with[0,pi/4] and the absolute tolerance with 10^-3 this will be the outputs:

Listing 3: Sample Bash code and Output.

```
The root is: 0.7391869921623933
Number of iteration:  12
```

If you replace the intervals with[-5,5] and the absolute tolerance with $10^{-}9$ this will be the outputs:

Listing 4: Sample Bash code and Output.

```
1 The root is: 0.7390851335367188
2 Number of iteration: 34
```

1.2 Fixed point iteration:

*Fixed point iteration has algorithm more complex than bisection method. Number of evaluation may change depends on our chosen g(x) function.

*If appropriate g function is chosen, the method can be applied rapidly, and converge faster.

*It may be required additional data, again depend on chosen g(x).

*It can be generalized on many equations with many g(x) and convergent types.

1.2.1 Example:

Solve the Example 1.1.1 using Fixed Point Iteration. f(x)=$x^3 + 4xr - 10 = 0$ We know this equation has root on the [1,2] interval. Let's define $x = g(x)$ in several ways:

(a) $x = g_1(x) = x - x^3 - 4x^2 + 10$,

(b) $x = g_2(x) = \sqrt{\frac{10}{x} - 4x}$,

(c) $x = g_3(x) = \frac{1}{2}\sqrt{10 - x^3}$,

(d) $x = g_4(x) = \sqrt{\frac{10}{4+x}}$,

(e) $x = g_5(x) = x - \frac{x^3+4x^2-10}{3x^2+8x}$

(b) function is not continous on [1,2]

(a) for every $x \in [1, 2]$, g(x) not in this interval.

(a) function is diverge, (b) function is undefined.

Let's show the iterative on a table:

n	(a)	(b)	(c)	(d)	(e)
0	1.5	1.5	1.5	1.5	1.5
1	-0.875	0.81649658	1.286943768	1.348399725	1.37333333
2	6.7324188	2.99690881	1.402540804	1.367376372	1.365262015
3	-469.720012	$(-8.65)^{\frac{1}{2}}$	1.345458374	1.364957015	1.365262014
4	1.03 x 10^8		1.375170253	1.365264748	1.365262013
5			1.360094193	1.365225594	

Table 1: Results of Example 1.1.1 with Fixed Point Iterations

1.2.2 Example

(a) $sin(x) - \frac{x}{1.4} = 0$ to solve this equation find a fixed point function g(x).

(b) Using g(x) find the number of iteration to solve this question with p_0=1.4 and smaller error than 10^-6.

(a) x=1.4sinx

$$g(\frac{\pi}{2}) = 1.4 \sin \frac{\pi}{2} = 1.4 < \frac{\pi}{2} = 1.5708$$

and

$$g(1) = 1.4sin1 = 1.1781 > 1$$

so for every $x \in [1, \pi/2]$, $g(x) \in [1, \pi/2]$ this means g(x) has at least one fixed point in $[1, \pi/2]$. On the other hand

$$|g'(x)| = |1.4cosx| = 1.4|cosx| < 1.4max|cosx|$$

if we accept h(x)= cosx,

for $x \in [1, \frac{\pi}{2}]$ h'(x)=-sinx<0.

$$|h(1)| = |cos1| = 0.54030 > |h(\pi/2)| = cos(\pi/2) = 0$$

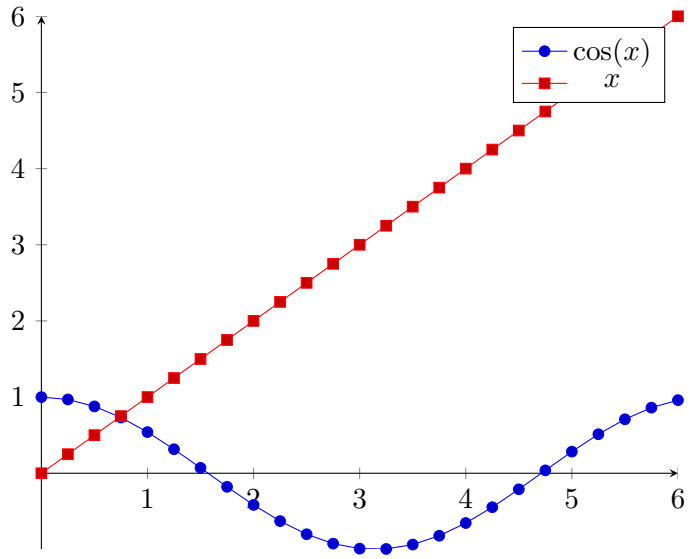
$$|g'(x)| < 1.4max|cosx| = 1.4cos1 = k = 0.75642 < 1$$

So they shows us there is one fixed point on $[1, \pi/2]$.

(b)$p_0 = 1.4$, $p_1 = g(p_1) = 1.4sin1.4 = 1.3796$ to show n iteration number $|p_n - p| < \frac{k^n}{1-k}|p_1 - p_0| < 10^-6$ with k=0.75642, n>40.605 is found.

1.2.3 Base Example

For the equation $f(x) = cosx - x = 0$ do iterative with using Fixed point method.



The graph shows there are only one fixed point is in the $[0, \pi/2]$.

for $p_0 = \pi/4$, $p_n = g(p_{n-1}) = cos(p_{n-1})$

n	p_n
0	0.7853981635
1	0.7071067810
2	0.7602445972
3	0.7246674808
4	0.7487198858
5	0.7325608446
6	0.743564213
7	0.7361282565

This function is so slow. $p_n = g(p_{n-1})$ could not reached yet.

Listing 5: Sample Python code for example 1.2.3

```python
#Fixed point iteration: initial guess:x=pi/4, stopping criteria:10^-3

from math import cos,pi
def g(x):
    #cos(x)-x=0 then cos(x)=x
    return cos(x)

def fixedPoint(x,epsilon=1.0e-3):
    i=0
    gx=g(x)
    if gx==0:
        return x,i
    while(True):
        i=i+1
        gx2 = g(gx)
        if abs(gx2-gx)<=epsilon:
            return gx,i
        gx=gx2

root,numiter = fixedPoint(pi/4)
print ("Root =",root)
print("Number of iterations =",numiter)
```

Listing 6: Sample Bash code and Output.

```
Root = 0.7394947711319744
Number of iterations = 12
```

If you replace the initial value with 0 and the absolute tolerance with 10^-3 this will be the outputs:

Listing 7: Sample Bash code and Output.

```
Root = 0.739567202212256
Number of iterations = 17
```

If you replace the initial value with 5 and the absolute tolerance with 10^-9 this will be the outputs:

Listing 8: Sample Bash code and Output.

```
Root = 0.7390851328107476
```

```
2 Number of iterations = 53
```

1.3 Newton's method:

*Newton's method has not simple algorithm.

*It can be diverge and converge, it is not easy to get without calculate some equations, so this method is not so safe or robust.

*Unlike the other methods, it needed additional data for example derivative of f. Also continuity of f is required too.

*However, it is a fast method, it do not need so many iterations.

*It can be generalizes to different systems.It could be most popular method.

1.3.1 Base Example

For the equation $f(x) = cosx - x = 0$ do iterative with using Newton's method.

$f'(x) = -sinx - 1$

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} = \frac{\cos p_{n-1} - p_{n-1}}{-\sin p_{n-1} - 1}$$

Here is the table with Newton's Method:

n	p_n
0	0.7853981635
1	0.7395361337
2	0.7390851781
3	0.7390851332
4	0.7390851332

Since p_3 and p_4 is equal, only after 3 iteration we reached the spotting criteria.

Now examine the python code for Newton Method to solve this example.

Listing 9: Sample Python code for example 1.3.1

```
1  #Newton's Method: initial guess:x=0, stopping criteria:10^-3
2
3  from math import cos,sin
4  def f(x):
5      return cos(x)-x
6  def df(x):
7      return -sin(x)-1
8  def newtonRaphson(x,epsilon=1.0e-3):
9      for i in range(30):
10         dx = f(x)/df(x)
11         x = x - dx
```

```
12          if abs(dx) < epsilon:
13              return x,i
14
15  root,numIter = newtonRaphson(0.0)
16  print ("Root =",root)
17  print("Number of iterations =",numIter)
```

Listing 10: Sample Bash code and Output.

```
1  Root = 0.739085133385284
2  Number of iterations = 3
```

If you replace the initial guess with pi/4 this will be the outputs:

Listing 11: Sample Bash code and Output.

```
1  Root = 0.7390851781060102
2  Number of iterations = 1
```

If you replace the initial guess with 5 and the stopping criteria 10^-9 this will be the outputs:

Listing 12: Sample Bash code and Output.

```
1  Root = 0.7390851332151607
2  Number of iterations = 21
```

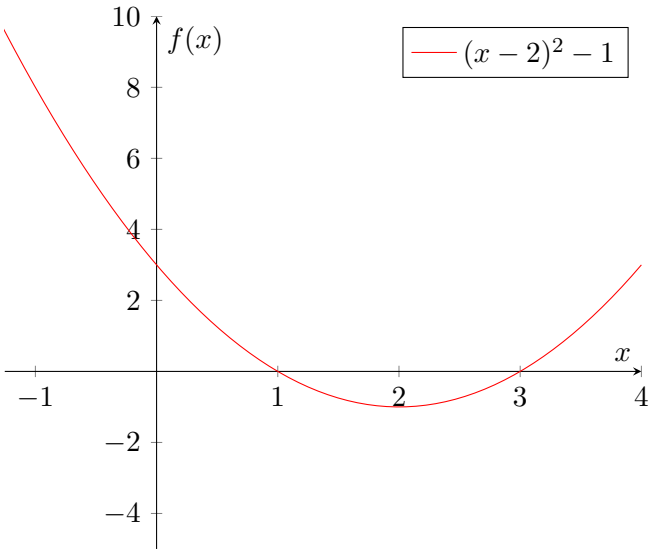
The reason is the increase in the iterations, the initial guess is so important because of it's tangent line. Sometimes you may get away from the root.

We mentioned about Newton's Method is not robust always. To show you this situation I want to show an example in which Newton's Method can not be apply-able.

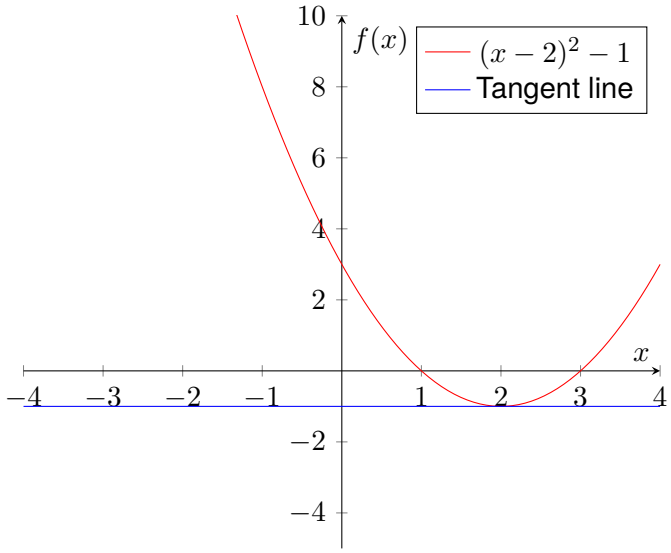
1.3.2 Example

$f(x) = (x-2)^2 - 1$ and inital guess $x_1 = 2$.

Let's look at the graph of f first.



Now, with considering Newton's Method's idea; we have to find f(2), draw to tangent line(derivative), do another iterative for cross point with x-axis. But when we draw the tangent line it seems like this:



So, It is impossible to make another iteration and we stuck at this point.

$$\frac{|f(x).f''(x)|}{|f'(x)^2|} < 1$$

then the convergence is assured.

1.4 Secant Method

*Secant Method has a algorithm, depends on Newton's method and definition of differentiation together.

*It is a two-step method. It requires two initial guess similar to Bisection Method. However Secant Method is way too fast.

*It's properties is likely Newton's' properties because the idea is the same, but Secant Method is faster.

1.4.1 Base Example

We solved the base example with both Fixed Point iteration, Newton's Method and bisection method. Now, let's try to solve this question with Secant Method.

The equation was $f(x) = cosx - x = 0$. Chosen first two inital guess are $x_0 = 0$, and $x_1 = 1$.

First iterative:

$$x_2 = 0 - 1 * \frac{f(0) * f(1)}{f(1) - f(0)}$$

$$x_2 = 0,6851$$

$$f(x_2) = \cos 0,6851 - 0,6851 = 0,0893$$

After doing the same operations, ended up results like in table:

n	x_0	$f(x_0)$	x_1	$f(x_1)$	x_2	$f(x_2)$
1	0	1	1	-0,4597	0,6851	0,0893
2	1	-0,4597	0,6851	0,0893	0,7363	0,0047
3	0,6851	0,0893	0,7363	0,0047	0,7391	0

We used same inital guesses as in Bisection Method. However, if you take the stopping criteria 10^-2 it takes only 2 iteration.

Listing 13: Sample Python code for example 1.4.1

```
#Secant Method: initial guesses:x=0, x=1, stopping criteria:10^-3


from math import cos,sin,ceil
def f(x):
    return cos(x)-x
def secant(x0,x1,epsilon=1.0e-2):
    for i in range(30):
        fx0 = f(x0);fx1 = f(x1)
        if (fx1-fx0!=0):
            div = (x1-x0)/(fx1-fx0)
            x2 = x0 - fx0*div
            fx2=f(x2)
            if (x2-x1)<=epsilon and ceil(fx2)==0.0:
                return x2,x1,i
            else:
                x0=x1
                x1=x2

root1,root2,numiter = secant(0.0,1.0)
print ("Root =",root1)
print("Number of iterations =",numiter)
```

Listing 14: Sample Bash code and Output.

```
1
2 Root = 0.7390851332151607
3 Number of iterations = 5
```

If you replace the interval with [0,pi/4] ,as we did when we solved this question with Bisection method, this will be the outputs:

Listing 15: Sample Bash code and Output.

```
1
2 Root = 0.7390851332151607
3 Number of iterations = 4
```

If you replace the interval with [-5,5] and the tolerance with 10^-9 ,as we did when we solved this question with Bisection method, this will be the outputs:

Listing 16: Sample Bash code and Output.

```
1
2 Root = 0.7390851332151607
3 Number of iterations = 6
```

You see, number of iteration is not getting too much larger like Bisection.

1.4.2 Example

$f(x) = e^-x - x$ calculate the root using Secant Method with inital guesses $x_0 = 0.0$ and $x_1 = 1.0$

$$f(x_0) = e^{-x_0} - x_0 -> f(x_0) = e^0 - 0 = 1$$

$$f(x_1) = e^{-x_1} - x_1 -> f(x_1) = 1/e - 1$$

$$x_{i+1} = x_i - \frac{f(x_i) * (x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

$$x_{i+1} = 1 - \frac{-0.6321205588 * (0 - 1)}{1 + 0.6321205588}$$

$$x_2 = 0.61270$$

This was the first iteration. Let's see the others on the table. At the last iteration, the root almost never changed and the function is almost equal to zero.

n	x_0	$f(x_0)$	x_1	$f(x_1)$	x_2
1	0	1	1	-0.63212056	0.61269984
2	1	-0.63212056	0.61269984	-0.07081395	0,56383839
3	0.61269984	-0.07081395	0,56383839	0.02518235	0,56717036
4	0,56383839	0.02518235	0,56717036	-0,00004242	0,56714331
5	0,56717036	-0,0000424	0,56714331	-0,00000003	0,56717329

2 LU Decomposition

A= $\begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{bmatrix}$

$$l_{21} = \frac{0}{2} = 0, l_{31} = \frac{0}{2} = 0, l_{41} = \frac{-1}{2} = -0.5$$

M1= $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}$ $A^{(1)} = M^{(1)} * A$= $\begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & 1.5 & -1 & 0 \end{bmatrix}$

For pivotting, let's change row 2 and 3.

$P^{(1)}$= $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $A^{(2)} = P^{(1)} * M^{(1)} * A$= $\begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 1.5 & -1 & 0 \end{bmatrix}$

$$l_{42} = \frac{-1.5}{1} = -3/2$$

$M^{(2)}$= $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{3}{2} & 0 & 1 \end{bmatrix}$ $A^{(3)} = M^{(2)} * P^{(1)} * M^{(1)} * A$= $\begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 2 & \frac{-3}{2} \end{bmatrix}$

$$l_{43} = \frac{2}{-1} = -2$$

$M^{(3)}$= $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix}$ $A^{(4)} = M^{(3)} * M^{(2)} * P^{(1)} * M^{(1)} * A$= $\begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$

PA=LU $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-1}{2} & \frac{-3}{2} & -2 & 1 \end{pmatrix} * \begin{pmatrix} 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$

L.Ux=b, Let's call Ux=y

L.y=b $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-1}{2} & \frac{-3}{2} & -2 & 1 \end{pmatrix} * \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Using forward substitution we get the results:

$$y_1 = 1, y_2 = 0, y_3 = 0$$

$$-y_1/2 - 3y_2 - 2y_3 = 0 + y_4 = 0, y4 = 1/2$$

U.x= $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$

Now we will use backward substitution to get results:

$$x_4 = 1$$

$$-x_3 + x_4 = 0, x_3 = 0$$

$$-x_2 + 2 - 1 = 0, x_2 = 1$$

$$2x_1 - 1 = 1, x_1 = 1$$

X= $\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$