Assignment in Numerical Analysis LaTeX

Full Name: Savvidis Theocharis 4555

10 December 2024

1 First Exercise

1.1 Bisection method

After testing the final code 5 times with different values of the lower (a) and upper (b) bounds the results are the following :

1. **For root = -1.1976237**

a = -2
b = -1
Iterations: 23

a = -1.5
b = -1
Iterations: 22

a = -1.2
b = -1
Iterations: 20

a = -1.19
b = -1.2
Iterations: 16

a = -1.197
b = -1.198
Iterations: 13

2. **For root = 1.5301335**

a = -1
b = 2
Iterations: 24

a = 0.5
b = 2
Iterations: 23

a = 1.4
b = 1.6
Iterations: 20

a = 1.5
b = 1.6
Iterations: 19

a = 1.53
b = 1.531
Iterations: 13

3. **For root = 0**

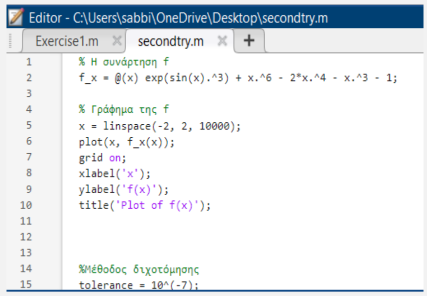
a = -1.4
b = 1.4
Iterations: 1

a = -1.3
b = 1.3
Iterations: 1

a = -1.2
b = 1.2
Iterations: 1

a = -1.25
b = 1.25
Iterations: 1

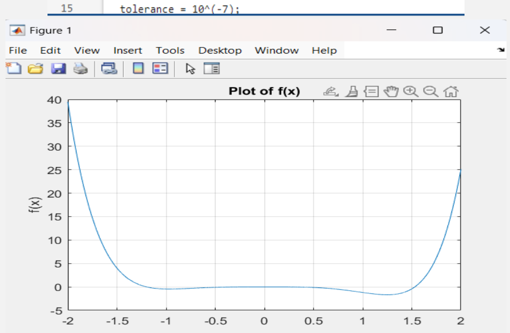
a = -1.199
b = 1.199
Iterations: 1

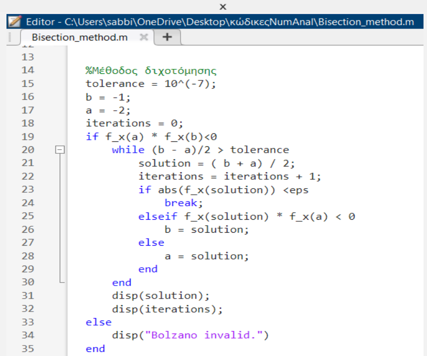


Editor - C:\Users\sabbi\OneDrive\Desktop\secondtry.m

Exercise1.m | secondtry.m

```
% Η συνάρτηση f
f_x = @(x) exp(sin(x).^3) + x.^6 - 2*x.^4 - x.^3 - 1;

% Γράφημα της f
x = linspace(-2, 2, 10000);
plot(x, f_x(x));
grid on;
xlabel('x');
ylabel('f(x)');
title('Plot of f(x)');



%Μέθοδος διχοτόμησης
tolerance = 10^(-7);
```





Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_method.m

Bisection_method.m

```

%Μέθοδος διχοτόμησης
tolerance = 10^(-7);
b = -1;
a = -2;
iterations = 0;
if f_x(a) * f_x(b)<0
    while (b - a)/2 > tolerance
        solution = ( b + a) / 2;
        iterations = iterations + 1;
        if abs(f_x(solution)) <eps
            break;
        elseif f_x(solution) * f_x(a) < 0
            b = solution;
        else
            a = solution;
        end
    end
    disp(solution);
    disp(iterations);
else
    disp("Bolzano invalid.")
end
```

Note: The above code was implemented without the assistance of language model.

1.2 Newton-Raphson method

1. For root = -1.1976237

   x_{n-1} = -2
   Iterations: 7

   x_{n-1} = -1.7
   Iterations: 5

   x_{n-1} = 1.3
   Iterations: 4

   x_{n-1} = -1.2
   Iterations: 2

   x_{n-1} = -1
   Iterations: 9

2. For root = 0

   The initialization x_{n-1} = 0 is itself leading to 0 ($f(0) = 0$) and it cannot be approached by other initializations except those being 5 or more precision digits close to 0

3. For root = 1.5301335

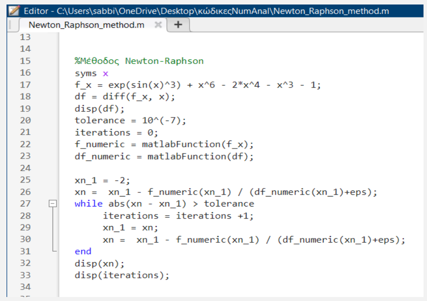
   x_{n-1} = 1.3
   Iterations: 8

   x_{n-1} = 1.5
   Iterations: 3

   x_{n-1} = 1.6
   Iterations: 4

   x_{n-1} = 1.8
   Iterations: 5

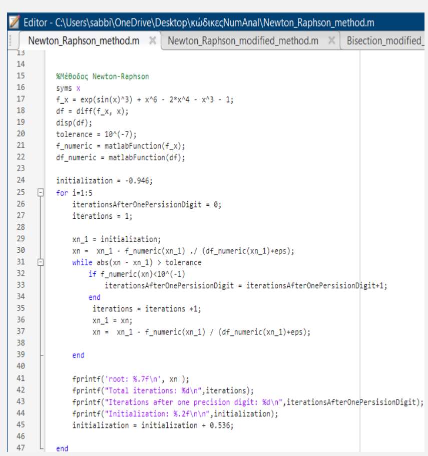
   x_{n-1} = 1.9
   Iterations: 6



```
%Μέθοδος Newton-Raphson
syms x
f_x = exp(sin(x)^3) + x^6 - 2*x^4 - x^3 - 1;
df = diff(f_x, x);
disp(df);
tolerance = 10^(-7);
iterations = 0;
f_numeric = matlabFunction(f_x);
df_numeric = matlabFunction(df);

xn_1 = -2;
xn =  xn_1 - f_numeric(xn_1) / (df_numeric(xn_1)+eps);
while abs(xn - xn_1) > tolerance
     iterations = iterations +1;
     xn_1 = xn;
     xn =  xn_1 - f_numeric(xn_1) / (df_numeric(xn_1)+eps);
end
disp(xn);
disp(iterations);
```

Note: Code displaying the Newton-Raphson method. No language model was used to create the code .

- In our scenario where seven precision digits are needed, we observe quadratic convergence after finding one precision digit, three or less other iterations are needed to approach our root with 7 precision digits. That is because after finding the first precision digit then the *error* almost becomes squared in each next iteration. That way after:
  - **one iteration** we find **2 precision digits**
  - **two iterations** we find **4 precision digits**
  - **three iterations** we find **8 precision digits** (we have reached the required 7 precision digits)



```
%Μέθοδος Newton-Raphson
syms x
f_x = exp(sin(x)^3) + x^6 - 2*x^4 - x^3 - 1;
df = diff(f_x, x);
disp(df);
tolerance = 10^(-7);
f_numeric = matlabFunction(f_x);
df_numeric = matlabFunction(df);

initialization = -0.946;
for i=1:5
    iterationsAfterOnePersisionDigit = 0;
    iterations = 1;

    xn_1 = initialization;
    xn =  xn_1 - f_numeric(xn_1) ./ (df_numeric(xn_1)+eps);
    while abs(xn - xn_1) > tolerance
        if f_numeric(xn)<10^(-1)
            iterationsAfterOnePersisionDigit = iterationsAfterOnePersisionDigit+1;
        end
         iterations = iterations +1;
         xn_1 = xn;
         xn =  xn_1 - f_numeric(xn_1) / (df_numeric(xn_1)+eps);

    end

    fprintf('root: %.7f\n', xn );
    fprintf("Total iterations: %d\n",iterations);
    fprintf("Iterations after one precision digit: %d\n",iterationsAfterOnePersisionDigit);
    fprintf("Initialization: %.2f\n\n",initialization);
    initialization = initialization + 0.536;

end
```

Note: The code used for testing Quadratic convergence for all of the roots.

- After experimentation with a variety of values, the method seems to **not** exhibit **quadratic convergence** for the interval **[-0.95, 1.22]** (approximation), where the derivative of f is approaching 0 and in which the iterations needed to approach the root 0 are not bounded by a quadratic decrease of error but the error follows a linear decrease as it can be seen in the table below.

```
Command Window
root: 0.0000320
Total iterations: 34
Iterations after one precision digit: 33
Initialization: -0.95

root: -0.0000688
Total iterations: 31
Iterations after one precision digit: 30
Initialization: -0.41

root: 0.0000101
Total iterations: 28
Iterations after one precision digit: 27
Initialization: 0.13

root: -0.0000672
Total iterations: 35
Iterations after one precision digit: 34
Initialization: 0.66

root: -0.0000540
Total iterations: 31
Iterations after one precision digit: 30
Initialization: 1.20
```

*Note: Method does **not** converge quadratically for root 0 .*

- For the other two intervals **(-2, -0.95)** and **(1.22, 2)** the method converges quadratically for the roots **-1.1976237** and **1.5301335** correspondingly , as the iterations needed to approach a 7 digit precision root after finding the first digit of precision are less or equal to 3. The code follows similar structure to the already displayed one and the results of the experimentation are shown below:

```
Command Window
root: -1.1976237
Total iterations: 8
Iterations after one precision digit: 3
Initialization: -2.00

root: -1.1976237
Total iterations: 8
Iterations after one precision digit: 4
Initialization: -1.74

root: -1.1976237
Total iterations: 6
Iterations after one precision digit: 3
Initialization: -1.48

root: -1.1976237
Total iterations: 4
Iterations after one precision digit: 3
Initialization: -1.22

root: 1.5301335
Total iterations: 13
Iterations after one precision digit: 3
Initialization: -0.96
```

```
Command Window
root: -1.1976237
Total iterations: 5
Iterations after one precision digit: 4
Initialization: 1.22

root: 1.5301335
Total iterations: 6
Iterations after one precision digit: 3
Initialization: 1.42

root: 1.5301335
Total iterations: 5
Iterations after one precision digit: 3
Initialization: 1.61

root: 1.5301335
Total iterations: 6
Iterations after one precision digit: 3
Initialization: 1.81

root: 1.5301335
Total iterations: 7
Iterations after one precision digit: 3
Initialization: 2.00
```

Note: Method converges quadratically for roots -1.1976237 and 1.5301335.

1.3 Secant method

1. For root = -1.1976237

x_{n-1} =-1 and $x_{n-2} = -2$
Iterations: 14

x_{n-1} =-1.2 and $x_{n-2} = -2$
Iterations: 4

x_{n-1} =-1.2 and $x_{n-2} = -1.5$
Iterations: 3

x_{n-1} = -1.1 and $x_{n-2} = 2$
Iterations: 8

x_{n-1} = 2 and $x_{n-2} = -1.1$
Iterations: 12

2. For root = 0

x_{n-1} =-1 and $x_{n-2} = 0.2$
Iterations: 49

x_{n-1} = -1 and $x_{n-2} = 2$
Iterations: 60

x_{n-1} = -0.5 and $x_{n-2} = 0.5$
Iterations: 53

x_{n-1} = -0.01 and $x_{n-2} = 0.01$
Iterations: 3

x_{n-1} = -0.005 and $x_{n-2} = 0.005$
Iterations: 3

3. For root = 1.5301335

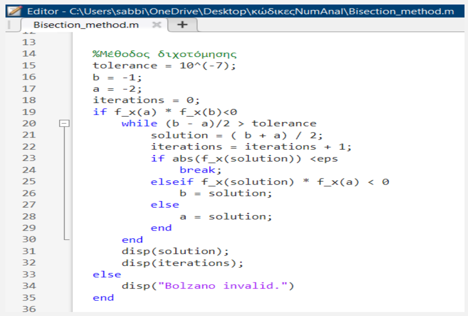
$x_{n-1} = 1.4$ and $x_{n-2} = 1.6$
Iterations: 6

$x_{n-1} = 1.6$ and $x_{n-2} = 1.4$
Iterations: 5

$x_{n-1} = 1.52$ and $x_{n-2} = 1.53$
Iterations: 3

$x_{n-1} = 1.54$ and $x_{n-2} = 1.53$
Iterations: 3

$x_{n-1} = 1.53$ and $x_{n-2} = 1.54$
Iterations: 2



Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_method.m

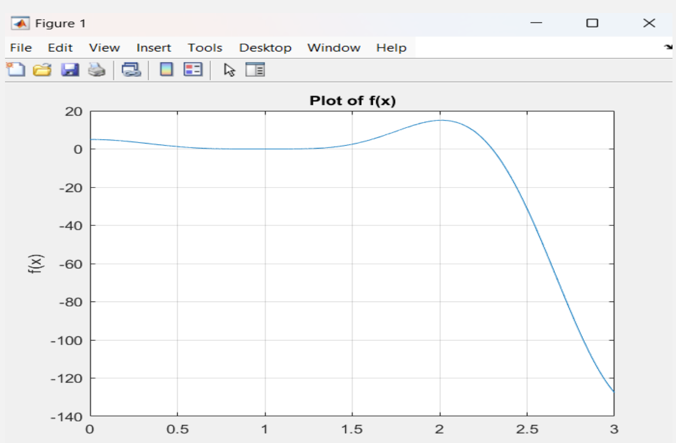
Bisection_method.m

```
%Μέθοδος διχοτόμησης
tolerance = 10^(-7);
b = -1;
a = -2;
iterations = 0;
if f_x(a) * f_x(b)<0
    while (b - a)/2 > tolerance
        solution = ( b + a) / 2;
        iterations = iterations + 1;
        if abs(f_x(solution)) <eps
            break;
        elseif f_x(solution) * f_x(a) < 0
            b = solution;
        else
            a = solution;
        end
    end
    disp(solution);
    disp(iterations);
else
    disp("Bolzano invalid.")
end
```

Note: The above code was implemented without the assistance of language model.

2 Second Exercise

The graph of the given function was created using the same code used for the previous one, only with changing the function.



Note: The graph of the given function.

Chat GPT language model was used in the second exercise **only for verifying the number of solutions** in the interval [0,3] of the given function. The following code was used:

```
% Count roots numerically using fzero
roots = []; % To store roots
for i = 1:length(domain)-1
    % Check if the function changes sign between domain points
    if f_numeric(domain(i)) * f_numeric(domain(i+1)) < 0
        root = fzero(f_numeric, [domain(i), domain(i+1)]);
        % Avoid duplicate roots due to floating-point precision
        if isempty(roots) || all(abs(roots - root) > 1e-6)
            roots = [roots; root];
        end
    end
end

% Display the roots and their count
disp('Roots of the function:');
disp(roots);
disp(['Total number of roots: ', num2str(length(roots))]);
```

2.1 Modified Newton-Raphson method

2.1.a

1. For $x_{n-1} = 0.8$

   root = 0.8410686
   Iterations: 5

2. For $x_{n-1} = 2.5$

   root = 2.3005239
   Iterations: 5

3. For $x_{n-1} = 1.05$

   root = 1.0472017
   Iterations: 16

Editor - C:\Users\sabbi\OneDrive\Desktop\modifiedNR.m *

modifiedNR.m *

```matlab



%Τροποποιημένη μέθοδος Newton-Raphson
syms x
f_x = 94.*(cos(x).^3) - 24.*cos(x)+177.*(sin(x).^2) - 108*(sin(x).^4)-72.*(cos(x).^3 .* sin(x).^2) -65;
df = diff(f_x, x);

tolerance = 10^(-7);
f_numeric = matlabFunction(f_x);
df_numeric = matlabFunction(df);

xn_1 = 0.8;
xn =  xn_1 - f_numeric(xn_1) / df_numeric(xn_1) - (1./2)*f_numeric(xn_1).^2 * df_numeric(df_numeric(xn_1)) ./df_numeric(xn_1).^3;
iterations = 1;
while abs(xn - xn_1) > tolerance
    xn_1 = xn;
    xn =  xn_1 - (f_numeric(xn_1) +eps) / (df_numeric(xn_1)+eps) - (1./2)*(f_numeric(xn_1)+eps).^2 * (df_numeric(eps+df_numeric(xn_1))+eps) ./(df_numeric(xn_1)+eps).^3;
     iterations = iterations +1;
end
    disp(xn);
    disp(iterations);
```

Note: The above code was implemented without the assistance of language model.

2.2 Modified Bisection method

2.2.a

1. For a = 2 and b = 2.35

   root = 2.3005240
   Iterations: 21

2. For a = 0.99 and b = 1.5

   root = 1.0471905
   Iterations: 20

3. For a = 0.3 and b = 0.85

   root = 0.8410687
   Iterations: 18

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_modified_method.m *

Newton_Raphson_modified_method.m | Bisection_modified_method.m * | Secant_modified_method.m

```matlab
%Τροποποιημένη μέθοδος διχοτόμησης
syms x
f_x = 94.*(cos(x).^3) - 24.*cos(x)+177.*(sin(x).^2) - 108*(sin(x).^4)-72.*(cos(x).^3 .* sin(x).^2) -65;
f_numeric = matlabFunction(f_x);
tolerance = 10^(-7);

b =0.85;
a = 0.3;
iterations = 0;
solution = a;
if f_numeric(a) * f_numeric(b) <0
    while abs((b - a)) > tolerance || abs(f_numeric(solution)) > tolerance
       iterations = iterations + 1;
       if abs(f_numeric(a)) <= abs(f_numeric(b))
           solution = a + (1/3) *(b-a);
       else
           solution = b - (1/3)*(b-a);
       end
       if sign(f_numeric(solution)) == sign(f_numeric(a))
           a = solution;
       else
           b = solution;
       end
    end
    fprintf('%.7f' , solution);
    disp(iterations);
else
    disp("Bolzano invalid.");
end
```

Note: The above code was implemented without the assistance of language model.

2.2.b

After adding a loop that iterates 20 times in the bisection modified method the result supports that the algorithm converges in a different number of iterations. Two experiments were conducted leading to the following results.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_modified_method.m

Newton_Raphson_modified_method.m | Bisection_modified_method.m | Secant_

```matlab
%Επανάληψη Τροποποιημένης Μεθόδου Διχοτόμησης 20 φορές
changeOfStep = 0.001;
for i =1:20
    b =1.046 -changeOfStep;
    a = 0.7 + changeOfStep;
    iterations = 0;
    solution = a;
    if f_numeric(a) * f_numeric(b) <0
      while abs((b - a)) > tolerance || abs(f_numeric(solution)) > tolerance
           iterations = iterations + 1;
           if abs(f_numeric(a)) <= abs(f_numeric(b))
               solution = a + (1/3) *(b-a);
          else
               solution = b - (1/3)*(b-a);
           end
           if sign(f_numeric(solution)) == sign(f_numeric(a))
              a = solution;
           else
             b = solution;
         end
      end
      fprintf('root: %.7f ' , solution);
      fprintf("Iterations: %d\n",iterations);
      changeOfStep = changeOfStep +0.005;
    else
     disp("Bolzano invalid.");
    end
end
```

Note: The above code was implemented without the assistance of language model.

In the first experimentation , for **root = 0.8410687** , a change **step** of **0.001** was used (in order to insure different initialization a was increased by that step and b was decreased by the same step).

```
Command Window
>> Bisection_modified_method
root: 0.8410687 Iterations: 17
root: 0.8410687 Iterations: 18
root: 0.8410686 Iterations: 19
root: 0.8410686 Iterations: 19
root: 0.8410687 Iterations: 17
root: 0.8410686 Iterations: 18
root: 0.8410687 Iterations: 18
root: 0.8410687 Iterations: 17
root: 0.8410687 Iterations: 18
root: 0.8410686 Iterations: 19
root: 0.8410687 Iterations: 19
root: 0.8410687 Iterations: 18
root: 0.8410687 Iterations: 19
root: 0.8410687 Iterations: 17
root: 0.8410687 Iterations: 16
root: 0.8410687 Iterations: 19
root: 0.8410687 Iterations: 16
root: 0.8410686 Iterations: 17
root: 0.8410687 Iterations: 19
root: 0.8410687 Iterations: 16
```

Note: The algorythm seems to converge for a different numbers of iterations.

In the second experimentation , for **root = 2.3005240** , a change **step** of **0.01** was used (in order to insure different initialization a was increased by that step and b was decreased by the same step).

```
Command Window
>> Bisection_modified_method
root: 2.3005240 Iterations: 23
root: 2.3005240 Iterations: 21
root: 2.3005240 Iterations: 26
root: 2.3005240 Iterations: 23
root: 2.3005240 Iterations: 25
root: 2.3005240 Iterations: 21
root: 2.3005240 Iterations: 24
root: 2.3005240 Iterations: 25
root: 2.3005240 Iterations: 24
root: 2.3005240 Iterations: 23
root: 2.3005240 Iterations: 24
root: 2.3005240 Iterations: 25
root: 2.3005240 Iterations: 23
root: 2.3005240 Iterations: 24
root: 2.3005240 Iterations: 21
root: 2.3005240 Iterations: 25
root: 2.3005240 Iterations: 25
root: 2.3005240 Iterations: 24
root: 2.3005240 Iterations: 23
root: 2.3005240 Iterations: 23
```

Note: The algorythm seems to converge for a different numbers of iterations.

2.3 Modified Secant method

2.3.a

1. For $x_n = 0.8$ and $x_{n+1} = 1.7$ and $x_{n+2} = 2.8$

   root = 0.8410687
   Iterations: 8

2. For $x_n = 0.8$ and $x_{n+1} = 1.7$ and $x_{n+2} = 2.8$

   root = 2.3005239
   Iterations: 5

3. For $x_n = 0.8$ and $x_{n+1} = 1.7$ and $x_{n+2} = 2.8$

   root = 1.0472017
   Iterations: 16

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Newton_Raphson_modified_method.m *

Newton_Raphson_modified_method.m * | Bisection_modified_method.m | Secant_modified_method.m

```matlab
%Τροποποιημένη μέθοδος Newton-Raphson
syms x
f_x = 94.*(cos(x).^3) - 24.*cos(x)+177.*(sin(x).^2) - 108*(sin(x).^4)-72.*(cos(x).^3 .* sin(x).^2) -65;
df = diff(f_x, x);

tolerance = 10^(-7);
f_numeric = matlabFunction(f_x);
df_numeric = matlabFunction(df);

xn_1 = 0.8;
xn =  xn_1 - f_numeric(xn_1) / df_numeric(xn_1) - (1./2)*f_numeric(xn_1).^2 * df_numeric(df_numeric(xn_1)) ./df_numeric(xn_1).^3;
iterations = 1;
while abs(xn - xn_1) > tolerance
    xn_1 = xn;
    xn =  xn_1 - (f_numeric(xn_1) ) / (df_numeric(xn_1)+eps) - (1./2)*(f_numeric(xn_1)).^2 * (df_numeric(eps+df_numeric(xn_1))) ./(df_numeric(xn_1)+eps).^3;
     iterations = iterations +1;
end
    disp(xn);
    disp(iterations);
```

Note: The above code was implemented without the assistance of language model.

2.4 Comparison of Modified methods with the initial methods

After experimenting 100 times for each initial method and its corresponding modified,for a variety of different initializations the results upon the convergence speed are the following based on an avarage value of iterations:

Method	**Average Iteration Speed**
Bisection Method	20.51
Modified Bisection Method	19.79
Newton-Raphson Method	19.29
Modified Newton-Raphson Method	19.51
Secant Method	39.85
Modified Secant Method	18.67

Note: Convergence speed comparison : Initial methods vs Modified methods

```
%Μέθοδος διχοτόμησης

TotalIterations = 0;
tolerance = 10^(-7);
c1 = -1.4;
c2 = -0.8;
for i=1:50
    a = c1 ;
    b = c2 ;
    iterations = 0;
    if f_x(a) * f_x(b)<0
        while (b - a)/2 > tolerance
            solution = ( b + a) / 2;
            iterations = iterations + 1;
            if abs(f_x(solution)) <eps
                break;
            elseif f_x(solution) * f_x(a) < 0
                b = solution;
            else
                a = solution;
            end
        end
        fprintf("Initialization: a= %.2f ",c1);
        fprintf("b= %.2f",c2);

        disp(solution);
        disp(iterations);
        TotalIterations = TotalIterations + iterations;
        c1 = c1 + 0.003;
        c2 = c2 - 0.003;
    else
        disp("Bolzano invalid.")
    end
end
```

```
tolerance = 10^(-7);
c1 = -1;
c2 = 2;
for i=1:40
    a = c1 ;
    b = c2 ;
    iterations = 0;
    if f_x(a) * f_x(b)<0
        while (b - a)/2 > tolerance
            solution = ( b + a) / 2;
            iterations = iterations + 1;
            if abs(f_x(solution)) <eps
                break;
            elseif f_x(solution) * f_x(a) < 0
                b = solution;
            else
                a = solution;
            end
        end
        fprintf("Initialization: a= %.2f ",c1);
        fprintf("b= %.2f",c2);

        disp(solution);
        disp(iterations);
        TotalIterations = TotalIterations + iterations;
        c1 = c1 + 0.003;
        c2 = c2 - 0.003;
    else
        disp("Bolzano invalid.")
    end
end
% As for root 0 every time the iterations are 1, i add 10 to the average (i opt for adding 10)
AverateIterations = (TotalIterations + 10) / 100;
disp(AverateIterations);
```

Note: Code for testing Bisection method 100 times and finding an average iteration speed.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_modified_method.m *

Newton_Raphson_method.m | Newton_Raphson_modified_method.m | Bisection_modified_metho

```matlab
%Τροποποιημένη μέθοδος διχοτόμησης | 100 δοκιμές

TotalIterations=0;



c1 = 0.86;
c2 = 1.5;
for i=1:33
    a = c1;
    b =c2;

    iterations = 0;
    solution = a;
    if f_numeric(a) * f_numeric(b) <0
        while abs((b - a)) > tolerance || abs(f_numeric(solution)) > tolerance
           iterations = iterations + 1;
           if abs(f_numeric(a)) <= abs(f_numeric(b))
               solution = a + (1/3) *(b-a);
           else
               solution = b - (1/3)*(b-a);
           end
           if sign(f_numeric(solution)) == sign(f_numeric(a))
               a = solution;
           else
               b = solution;
           end
        end
        fprintf('%.7f' , solution);
        disp(iterations);
        c1 = c1 + 0.005;
        c2 = c2 - 0.005;
        TotalIterations = TotalIterations + iterations;
    else
```

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_modified_method.m

Newton_Raphson_method.m | Newton_Raphson_modified_method.m | Bisection_modified_metho

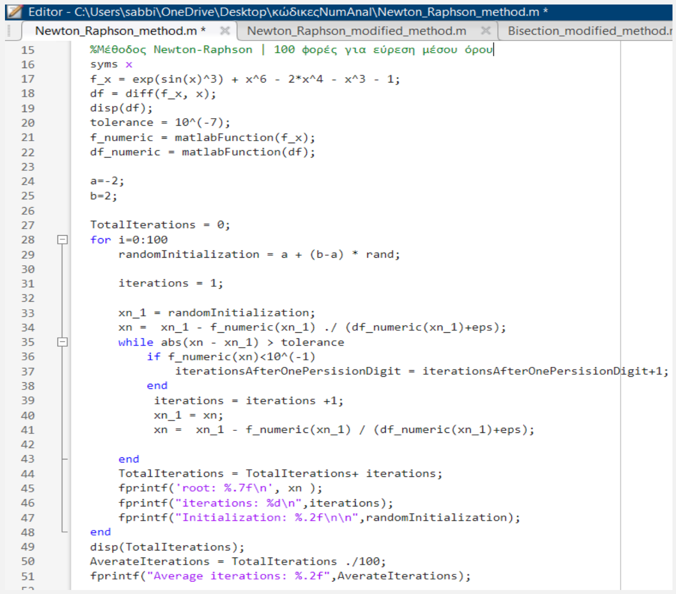
```matlab
    else
        disp("Bolzano invalid.");
    end
end

c1 = 0.5;
c2 = 1.047;
for i=1:33
    a = c1;
    b =c2;

    iterations = 0;
    solution = a;
    if f_numeric(a) * f_numeric(b) <0
        while abs((b - a)) > tolerance || abs(f_numeric(solution)) > tolerance
           iterations = iterations + 1;
           if abs(f_numeric(a)) <= abs(f_numeric(b))
               solution = a + (1/3) *(b-a);
           else
               solution = b - (1/3)*(b-a);
           end
           if sign(f_numeric(solution)) == sign(f_numeric(a))
               a = solution;
           else
               b = solution;
           end
        end
        fprintf('%.7f' , solution);
        disp(iterations);
        c1 = c1 + 0.005;
        c2 = c2 - 0.005;
        TotalIterations = TotalIterations + iterations;

    else
```

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Bisection_modified_method.m

Newton_Raphson_method.m | Newton_Raphson_modified_method.m | Bisection_modified_metho

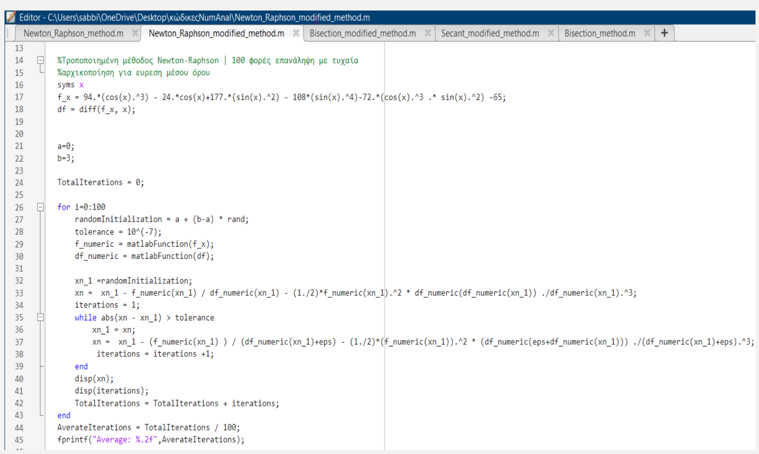
```matlab


c1 = 1.2;
c2 = 3;
for i=1:34
    a = c1;
    b =c2;

    iterations = 0;
    solution = a;
    if f_numeric(a) * f_numeric(b) <0
        while abs((b - a)) > tolerance || abs(f_numeric(solution)) > tolerance
           iterations = iterations + 1;
           if abs(f_numeric(a)) <= abs(f_numeric(b))
               solution = a + (1/3) *(b-a);
           else
               solution = b - (1/3)*(b-a);
           end
           if sign(f_numeric(solution)) == sign(f_numeric(a))
               a = solution;
           else
               b = solution;
           end
        end
        fprintf('%.7f' , solution);
        disp(iterations);
        c1 = c1 + 0.005;
        c2 = c2 - 0.005;
        TotalIterations = TotalIterations + iterations;

    else
        disp("Bolzano invalid.");
    end
end

AverageIterations = TotalIterations / 100;
disp(AverageIterations);
```

Note: Code for testing modified Bisection method 100 times and finding an average iteration speed.




```matlab
%Μέθοδος Newton-Raphson | 100 φορές για εύρεση μέσου όρου
syms x
f_x = exp(sin(x)^3) + x^6 - 2*x^4 - x^3 - 1;
df = diff(f_x, x);
disp(df);
tolerance = 10^(-7);
f_numeric = matlabFunction(f_x);
df_numeric = matlabFunction(df);

a=-2;
b=2;

TotalIterations = 0;
for i=0:100
    randomInitialization = a + (b-a) * rand;

    iterations = 1;

    xn_1 = randomInitialization;
    xn =  xn_1 - f_numeric(xn_1) ./ (df_numeric(xn_1)+eps);
    while abs(xn - xn_1) > tolerance
        if f_numeric(xn)<10^(-1)
            iterationsAfterOnePersisionDigit = iterationsAfterOnePersisionDigit+1;
        end
         iterations = iterations +1;
         xn_1 = xn;
         xn =  xn_1 - f_numeric(xn_1) / (df_numeric(xn_1)+eps);

    end
    TotalIterations = TotalIterations+ iterations;
    fprintf('root: %.7f\n', xn );
    fprintf("iterations: %d\n",iterations);
    fprintf("Initialization: %.2f\n\n",randomInitialization);
end
disp(TotalIterations);
AverateIterations = TotalIterations ./100;
fprintf("Average iterations: %.2f",AverateIterations);
```

Note: Code for testing Newton Raphson method 100 times to test the average convergence speed.




```matlab

%Τροποποιημένη μέθοδος Newton-Raphson | 100 φορές επανάληψη με τυχαία
%αρχικοποίηση για ευρεση μέσου όρου
syms x
f_x = 94.*(cos(x).^3) - 24.*cos(x)+177.*(sin(x).^2) - 108*(sin(x).^4)-72.*(cos(x).^3 .* sin(x).^2) -65;
df = diff(f_x, x);


a=0;
b=3;

TotalIterations = 0;

for i=0:100
    randomInitialization = a + (b-a) * rand;
    tolerance = 10^(-7);
    f_numeric = matlabFunction(f_x);
    df_numeric = matlabFunction(df);

    xn_1 =randomInitialization;
    xn =  xn_1 - f_numeric(xn_1) / df_numeric(xn_1) - (1./2)*f_numeric(xn_1).^2 * df_numeric(df_numeric(xn_1)) ./df_numeric(xn_1).^3;
    iterations = 1;
    while abs(xn - xn_1) > tolerance
        xn_1 = xn;
        xn =  xn_1 - (f_numeric(xn_1) ) / (df_numeric(xn_1)+eps) - (1./2)*(f_numeric(xn_1)).^2 * (df_numeric(eps+df_numeric(xn_1))) ./(df_numeric(xn_1)+eps).^3;
         iterations = iterations +1;
    end
    disp(xn);
    disp(iterations);
    TotalIterations = TotalIterations + iterations;
end
AverateIterations = TotalIterations / 100;
fprintf("Average: %.2f",AverateIterations);
```

Note: Code for testing modified Newton-Raphson method 100 times to test the average convergence speed.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Secant_method.m *

Secant_method.m * | Secant_modified_method.m | Bisection_method.m | Bisection_modified_method.m

```matlab

%Μέθοδος Τέμνουσας | 100 επαναλήψεις για εύρεση μέσης ταχύτητας σύγκλισης


TotalIterations = 0;

a=-2;
b=2;

for i=0:100
    iterations =1;
    randomInitialization1 = a + (b-a) * rand;
    randomInitialization2 = a + (b-a) * rand;
    xn_1 = randomInitialization1;
    xn_2 = randomInitialization2;
    xn =  xn_1 - f_numeric(xn_1) *(xn_1 - xn_2) / (f_numeric(xn_1)-f_numeric(xn_2)+eps);
    while abs(xn - xn_1)  > tolerance
        iterations = iterations +1;
        xn_2 = xn_1;
        xn_1 = xn;
        xn =  xn_1 - f_numeric(xn_1) .*(xn_1 - xn_2) / (f_numeric(xn_1)-f_numeric(xn_2)+eps);
    end
    fprintf('%.7f', xn);
    disp(iterations);
    TotalIterations = TotalIterations + iterations;
end
AverateIterations = TotalIterations/100;
disp(AverateIterations);

```

Note: Code for testing Secant method 100 times to test the average convergence speed.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Secant_modified_method.m

Secant_method.m | Secant_modified_method.m

```matlab

%Μέθοδος Τέμνουσας Μετασχηματισμένης | 100 επαναλήψεις για εύρεση μέσης ταχύτητας σύγκλισης
syms x
f_x = 94.*(cos(x).^3) - 24.*cos(x)+177.*(sin(x).^2) - 108*(sin(x).^4)-72.*(cos(x).^3 .* sin(x).^2) -65;
tolerance = 10^(-7);
f_numeric = matlabFunction(f_x);
MAX_ITERATIONS = 200;
a = 0;
b = 3;
TotalIterations = 0;
succesfulIterations =0;
while succesfulIterations<100
    randomInitialization1 = a + (b-a) * rand;
    randomInitialization2 = a + (b-a) * rand;
    randomInitialization3 = a + (b-a) * rand;

    xn = randomInitialization1;
    xn1 = randomInitialization2;
    xn2 = randomInitialization3;
    iterations = 1;
    q = f_numeric(xn)/f_numeric(xn1);
    r = f_numeric(xn2)/f_numeric(xn1);
    s = f_numeric(xn2)/f_numeric(xn);
    xn3 = xn2 - (( r.*(r-q).*(xn2-xn1) + (1-r).*s.*(xn2-xn) )/ (((q-1).*(r-1).*(s-1)+eps)));

    while abs(f_numeric(xn3)) > tolerance || abs(xn3 - xn2) > tolerance && abs(xn3 - xn1) > tolerance && abs(xn3 - xn) > tolerance
```

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Secant_modified_method.m

Secant_method.m | Secant_modified_method.m

while abs(f_numeric(xn3)) > tolerance || abs(xn3 - xn2) > tolerance && abs(xn3 - xn1) > tolerance && abs(xn3 - xn) > tolerance

```matlab

        xn=xn1;
        xn1=xn2;
        xn2=xn3;
        q = f_numeric(xn)/f_numeric(xn1);
        r = f_numeric(xn2)/f_numeric(xn1);
        s = f_numeric(xn2)/f_numeric(xn);
        xn3 = xn2 - (( r.*(r-q).*(xn2-xn1) + (1-r).*s.*(xn2-xn) )/ (((q-1).*(r-1).*(s-1)+eps)));
        iterations = iterations +1;
        if iterations==200
            break;
        end
    end
    fprintf('%.7f', xn3);
    disp(iterations);
    if(iterations<200)
        TotalIterations = TotalIterations + iterations;
        succesfulIterations = succesfulIterations+1;
        disp(succesfulIterations);
    end
end

AverateIterations = TotalIterations/100;
disp(AverateIterations);
```

Note: Code for testing modified Secant method 100 times to test the average convergence speed.

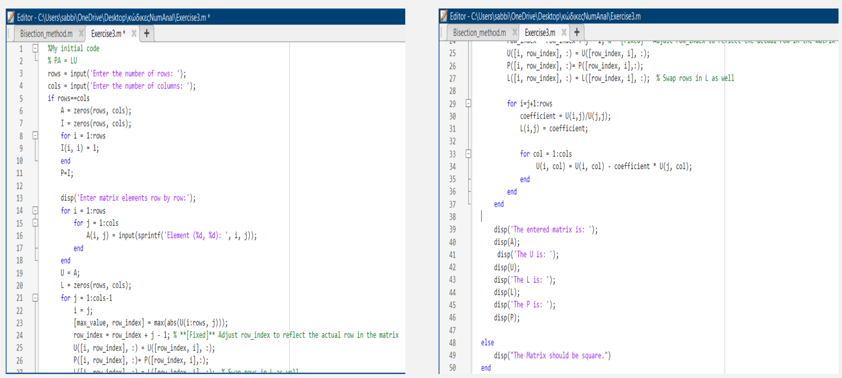
3 Exercise 3

3.1 LU decomposition

After writing the initial code that is displayed below, the only problem that occurred during the LU Decomposition was that the L Matrix was created but with its elements not being placed in the correct positions. Passing the code through Chat GPT, lead to the final code also being displayed below.The main changes conducted by GPT were:

1. the L Matrix updating

2. the partial pivoting implementation for the changing of the rows based on the *max* element.



Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Exercise3.m *

Bisection_method.m | Exercise3.m *

```
%My initial code
% PA = LU
rows = input('Enter the number of rows: ');
cols = input('Enter the number of columns: ');
if rows==cols
    A = zeros(rows, cols);
    I = zeros(rows, cols);
    for i = 1:rows
    I(i, i) = 1;
    end
    P=I;

    disp('Enter matrix elements row by row:');
    for i = 1:rows
        for j = 1:cols
            A(i, j) = input(sprintf('Element (%d, %d): ', i, j));
        end
    end
    U = A;
    L = zeros(rows, cols);
    for j = 1:cols-1
        i = j;
        [max_value, row_index] = max(abs(U(i:rows, j)));
        row_index = row_index + j - 1; % **[Fixed]** Adjust row_index to reflect the actual row in the matrix
        U([i, row_index], :) = U([row_index, i], :);
        P([i, row_index], :)= P([row_index, i],:);
```

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Exercise3.m

Bisection_method.m | Exercise3.m

```
        U([i, row_index], :) = U([row_index, i], :);
        P([i, row_index], :)= P([row_index, i],:);
        L([i, row_index], :) = L([row_index, i], :);  % Swap rows in L as well

        for i=j+1:rows
            coefficient = U(i,j)/U(j,j);
            L(i,j) = coefficient;

            for col = 1:cols
                U(i, col) = U(i, col) - coefficient * U(j, col);
            end
        end
    end

    disp('The entered matrix is: ');
    disp(A);
     disp('The U is: ');
    disp(U);
    disp('The L is: ');
    disp(L);
    disp('The P is: ');
    disp(P);

else
    disp("The Matrix should be square.")
end
```

Note: The initial code.

```
% LU Decomposition with Partial Pivoting
for j = 1:cols-1
    % Partial pivoting: find row with max value in column
    [~, row_index] = max(abs(U(j:rows, j)));
    row_index = row_index + j - 1;

    % Row swapping for P, Matrix (U), and L
    if row_index ~= j
        % Swap rows in Matrix (U)
        U([j, row_index], :) = U([row_index, j], :);

        % Swap rows in P
        P([j, row_index], :) = P([row_index, j], :);

        % Swap rows in L below the pivot column only
        if j > 1
            L([j, row_index], 1:j-1) = L([row_index, j], 1:j-1);
        end
    end

    % Elimination to form U and update L
    for i = j+1:rows
        coefficient = U(i, j) / U(j, j);
        L(i, j) = coefficient;

        % Update the row in Matrix
        U(i, j:cols) = U(i, j:cols) - coefficient * U(j, j:cols);
    end
end
```

Note: Final code, after Chat GPT's changes to the initial one.

After finding the $PA = LU$ decomposition and having to solve the equation $Ax = b$ the following is true:

$$Ax = b$$
$$PAx = Pb$$
$$LUx = Pb$$

First, considering that $Ly = Pb$ $y = L\ Pb$; in Matlab code, forward substitution is used for solving, as Matlab documentation suggests. Then what remains is to consider $Ux = y$ and solve it using back substitution with the Matlab code $x = U\ y$; The code is displayed below:

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Exercise3.m

Bisection_method.m | Exercise3.m | Newton_Raphson_method.m

```
else
    disp("The Matrix should be square.");
end

b = zeros(rows, 1);
for i = 1:rows
    b(i) = input(sprintf('Enter b%d: ', i));
end

b_transformed = P * b;

y = L \ b_transformed;

x = U \ y;

disp('The solution x is:');
disp(x);
```

Note: The code was created using no language model , but Matlab documentation.

3.2 Cholesky

For the Cholesky function, Chat GPT was used in order to find the Equation that produces the L matrix. The code implemented was based on the answers that are displayed below.

Αποσύνθεση Cholesky: Ο αλγόριθμος αποσύνθεσης Cholesky υπολογίζει τα στοιχεία του L χρησιμοποιώντας τις σχέσεις:

- **Για τα διαγώνια στοιχεία l_{ii}:** Το πρώτο βήμα για κάθε i είναι να υπολογιστεί το στοιχείο στην διαγώνιο:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

Για τα στοιχεία κάτω από την διαγώνιο l_{ji} όπου $j > i$: Τα υπόλοιπα στοιχεία του πίνακα L υπολογίζονται με την εξής σχέση:

$$l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} \cdot l_{ik}}{l_{ii}}$$

Note: The equations were provided using Chat GPT.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Choleski.m

Exercise3.m | Choleski.m

```
%Choleski
dimention = input('Enter the dimention of the square symmetric, positive definite matrix A: ');
disp('Enter matrix elements row by row:');
A = zeros(dimention, dimention);
for i = 1:dimention
    for j = 1:dimention
        A(i, j) = input(sprintf('Element (%d, %d): ', i, j));
    end
end
L=zeros(dimention,dimention);
for i=1:dimention
   L(i, i) = sqrt(A(i, i) - L(i, :)*L(i, :)');
   for j=(i + 1):dimention
      L(j, i) = (A(j, i) - L(i,:)*L(j ,:)')/L(i, i);
   end
end
disp(L);
```

Note:No language model was used to implement the code itself except the equations.

3.3 Gauss-Seidel

The Gauss-Seidel method for the given system, both for $n = 10$ and for $n = 5000$ produces the solution $x = (1, 1, 1, ..., 1)$.

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Gauss_Seidel.m

Exercise3.m | Choleski.m | Gauss_Seidel.m

```
%Gauss-Seidel
n = 10;

A = zeros(n,n);
for i=1:n
    if(i<=n-1)
        A(i+1,i) = -2;
        A(i,i+1) = -2;
    end
    A(i,i) = 5;
end

b(1) = 3;
b(n) = 3;
for i=2:n-1
    b(i) = 1;
end

m=1;
tolerance = 10^(-5);
x = zeros(n,1);
error = 100;
while error>tolerance
    new_x = x;
    for i=1:n
        Sum1=0;
        for j=1:i-1
            Sum1 =Sum1+A(i,j)*new_x(j);
        end
        Sum2=0;
```

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\Gauss_Seidel.m

Exercise3.m | Choleski.m | Gauss_Seidel.m

```
error = 100;
while error>tolerance
    new_x = x;
    for i=1:n
        Sum1=0;
        for j=1:i-1
            Sum1 =Sum1+A(i,j)*new_x(j);
        end
        Sum2=0;
        for j=i+1:n
            Sum2 =Sum2+A(i,j)*x(j);
        end
        new_x(i) = (1./A(i,i)) * (b(i) - Sum1 -Sum2);
    end
    max = 0;
    for k=1:n
        if abs(new_x(k)-x(k))>max
            max = abs(new_x(k)-x(k));
        end
    end
    error = max;
    x = new_x;
end
disp (new_x);
```

Note:The above code was created without the assistance of a language model.

4 Exercise 4

4.1 Stochastic G Matrix

In order for G to be a stochastic matrix, each of each columns should have a summary of 1 (its maximum eigenvalue should be equal to 1). Each of G matrix's element is given as follows.

$$G(i,j) = \frac{q}{n} + \frac{A(j,i)(1-q)}{n_i}$$

For each i from 1 to n, the following relationship must be true:

$$\begin{aligned}
&\sum_{i=1}^{n}\left(\frac{q}{n} + \frac{A(j,i)(1-q)}{n_j}\right) = 1 \\
\iff\ & n \cdot \frac{q}{n} + \sum_{i=1}^{n}\left(\frac{A(j,i)(1-q)}{n_j}\right) = 1 \\
\iff\ & \sum_{i=1}^{n}\left(\frac{A(j,i)(1-q)}{n_j}\right) = 1-q \\
\iff\ & \sum_{i=1}^{n}\left(\frac{A(j,i)}{n_j}\right) = 1 \\
\iff\ & \frac{\sum_{i=1}^{n} A(j,i)}{n_j} = 1
\end{aligned}$$

Which is indeed true, as for i from 1 to n, $\sum_{i=1}^{n} A(j,i)$ gives the summary of the column j, which is equal to n_j.

4.2 Creation of G

The vertex that corresponds to the eigenvector of the maximum eigenvalue is:

$$\mathbf{p} = (0.0268246, 0.0298611, 0.0298611, 0.0268246, 0.0395872, 0.0395872, 0.0395872, 0.0395872, 0.0745644, 0.1063200, 0.1063200, 0.0745644, 0.1250916, 0.1163279, 0.1250916)$$

Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\All_Codes_Matlab\Exe

Exercise4.m

```matlab
%Page Rank

n = 15;
q = 0.15;
A = [
    0 1 0 0 0 0 0 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 1 0 0 0 0 0 0 0 0;
    0 1 0 0 0 1 0 1 0 0 0 0 0 0 0;
    0 0 1 0 0 0 0 0 0 0 0 1 0 0 0;
    1 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 1 1 0 0 0 0;
    0 0 0 0 0 0 0 0 0 1 1 0 0 0 0;
    0 0 0 1 0 0 0 0 0 0 1 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 1 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
    0 0 0 0 0 0 1 1 0 0 1 0 0 0 0;
    0 0 0 0 0 0 0 0 1 0 0 0 0 1 0;
    0 0 0 0 0 0 0 0 0 1 1 0 1 0 1;
    0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
];
Google = zeros(n,n);
for j=1:n
    n_j = 0;
    for i =1:n
        n_j = n_j + A(j,i);
    end
    for i =1:n
        Google(i,j) = q/n + (A(j,i)*(1-q)) /n_j;
    end
end

bn_1 = A(:,1);
for k =1:10000
    bn = Google * bn_1;
    %bno = (1/bn(1,1)) *bn;
    bn_1 = bn;

end
fprintf('%.7f ',bn);
```

Note:The code implementing the G matrix and the vertex that corresponds to the eigenvector of the maximum eigenvalue. No language model was used.

4.3 Adding edges

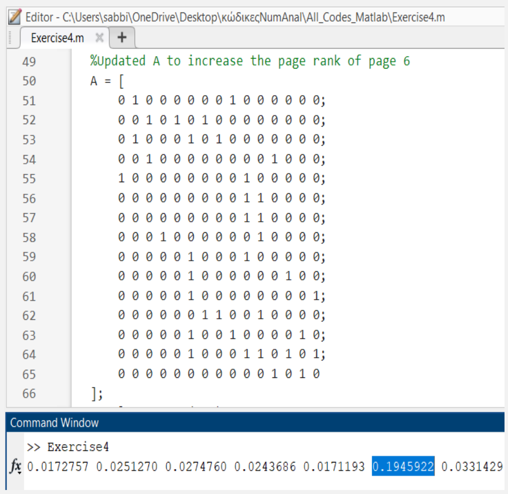
Having in mind that the needed page rank to be increased is that of **page 6** after **adding** the edges:

$$10 \to 6, \quad 11 \to 6, \quad 13 \to 6, \quad 14 \to 6,$$

and **removing** the already existing edge:

$$9 \to 5,$$

the result is as follows: The page rank of page 6 is **increased** from **0.0395872 to 0.1945922** That happens because the 4 new edges that lead to page 6 are beginning from pages that already have an increased page rank compared to the other pages. Also the removed edge is not directly affecting the page rank of page 6 so it is preferred for its irrelevancy.



Editor - C:\Users\sabbi\OneDrive\Desktop\κώδικεςNumAnal\All_Codes_Matlab\Exercise4.m

Exercise4.m

```
%Updated A to increase the page rank of page 6
A = [
    0 1 0 0 0 0 0 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 1 0 0 0 0 0 0 0 0;
    0 1 0 0 0 1 0 1 0 0 0 0 0 0 0;
    0 0 1 0 0 0 0 0 0 0 0 1 0 0 0;
    1 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 1 1 0 0 0 0;
    0 0 0 0 0 0 0 0 0 1 1 0 0 0 0;
    0 0 0 1 0 0 0 0 0 0 1 0 0 0 0;
    0 0 0 0 0 1 0 0 0 1 0 0 0 0 0;
    0 0 0 0 0 1 0 0 0 0 0 0 1 0 0;
    0 0 0 0 0 1 0 0 0 0 0 0 0 0 1;
    0 0 0 0 0 0 1 1 0 0 1 0 0 0 0;
    0 0 0 0 0 1 0 0 1 0 0 0 0 1 0;
    0 0 0 0 0 1 0 0 0 1 1 0 1 0 1;
    0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
];
```

Command Window

```
>> Exercise4
0.0172757 0.0251270 0.0274760 0.0243686 0.0171193 0.1945922 0.0331429
```

Note: Code displaying updated matrix A targeting to increase the page rank of page 6

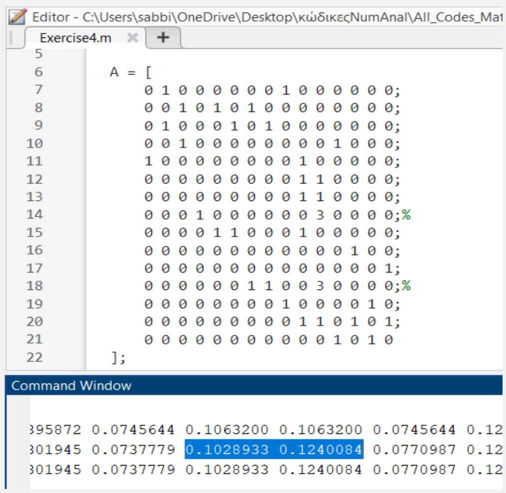
4.4 Experimenting with q

- After **decreasing q** to 0.02 in the new Graph the page rank of page 6 that was previously increased now seems to be **increased** even more from 0.1945922 to 0.2279782.
- Also **increasing q** to 0.6 **decreases the page rank of page 6** from 0.1945922 to 0.1126697.

It is evident that when the **q variable is notably increased**, being the probability of a user moving to a random page , that means that **it is more likely for a user to move to a random page** which leads to the connection - references - edges between pages playing a less important role. The opposite is happening when **q is decreased** , meaning that a user is not likely to move to random pages but **very much likely to move to pages referenced in the page he is already in**. That is the reason that page 6 page rank is notably increased when q has a small value: because many other pages with high page ranks are pointing - including a reference to page 6.

4.5 Page 11 tries to increase its page rank

The result shows that the page rank of page 11 **is increased from 0.1063200 to 0.1240084** which indicates that the strategy of changing the strength of $A(12, 11)$ and $A(8, 11)$ links to page 11 is indeed working.



Note: Code displaying updated matrix A targeting to increase the page rank of page 11 by changing the strength at which pages 8 and 12 point at it.

4.6 Deletion of page 10

Page	**Initial Page Rank**	Page Rank After Deletion	**Change**
Page 1	0.0268246	0.0470950	↑ **Increase**
Page 2	0.0298611	0.0409114	↑ **Increase**
Page 3	0.0298611	0.0359356	↑ **Increase**
Page 4	0.0268246	0.0320700	↑ **Increase**
Page 5	0.0395872	0.0428008	↑ **Increase**
Page 6	0.0395872	0.0413910	↑ **Increase**
Page 7	0.0395872	0.0516587	↑ **Increase**
Page 8	0.0395872	0.0502489	↑ **Increase**
Page 9	0.0745644	0.0482235	↓ **Decrease**
Page 10	0.1063200	-	-
Page 11	0.1063200	0.1709627	↑ **Increase**
Page 12	0.0745644	0.1035981	↑ **Increase**
Page 13	0.1250916	0.0411619	↓ **Decrease**
Page 14	0.1163279	0.1074622	↓ **Decrease**
Page 15	0.1250916	0.1864802	↑ **Increase**