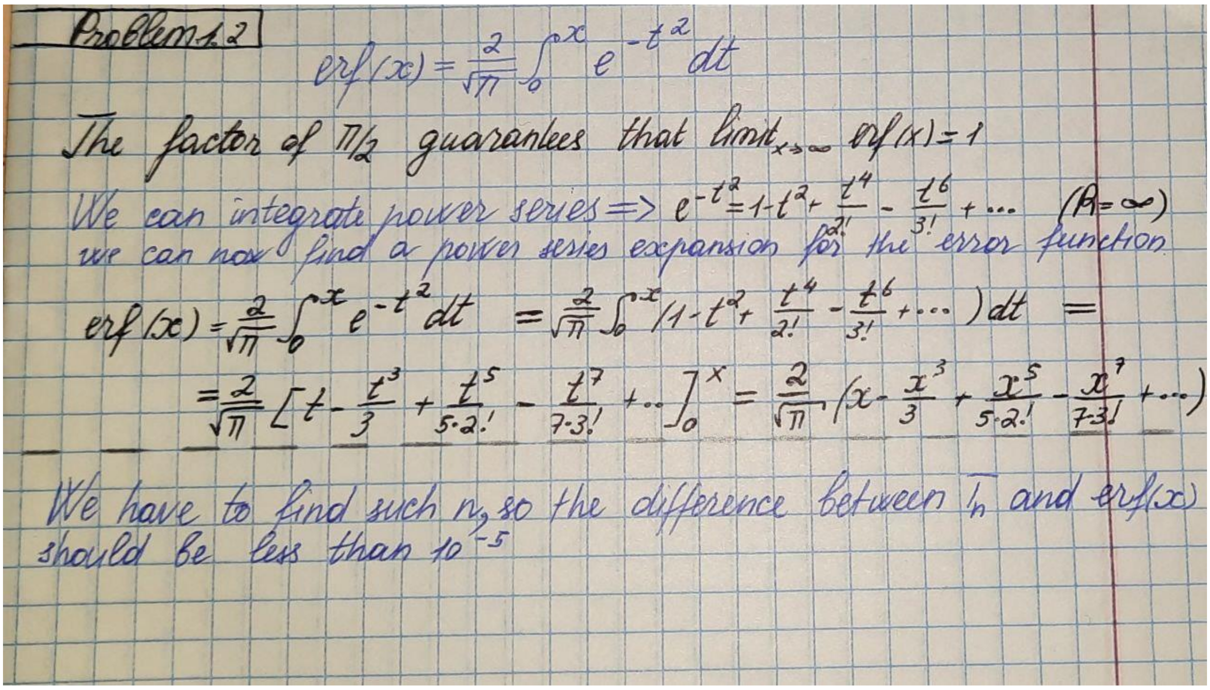
Lupascu Felicia

Homework 2

Ex.1



Problem 1.2

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

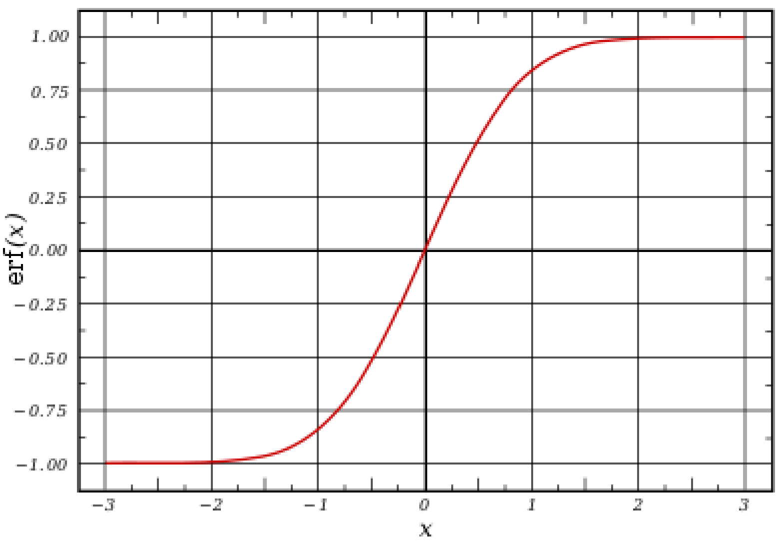
The factor of $\pi/2$ guarantees that $\lim_{x \to \infty} erf(x) = 1$

We can integrate power series $\Rightarrow e^{-t^2} = 1 - t^2 + \frac{t^4}{2!} - \frac{t^6}{3!} + \dots \quad (R = \infty)$
we can now find a power series expansion for the error function

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_0^x \left(1 - t^2 + \frac{t^4}{2!} - \frac{t^6}{3!} + \dots\right) dt =$$

$$= \frac{2}{\sqrt{\pi}} \left[t - \frac{t^3}{3} + \frac{t^5}{5 \cdot 2!} - \frac{t^7}{7 \cdot 3!} + \dots\right]_0^x = \frac{2}{\sqrt{\pi}} \left(x - \frac{x^3}{3} + \frac{x^5}{5 \cdot 2!} - \frac{x^7}{7 \cdot 3!} + \dots\right)$$

We have to find such n, so the difference between T_n and $erf(x)$ should be less than 10^{-5}



Lupascu Felicia

Ex.2

```python
memoized = {0: 1, 1: 1}

def fib(n):
    if (n in memoized):
        return(memoized[n])
    else:
        memoized[n] = fib(n-1) + fib(n-2)
        return(memoized[n])

for i in range(40):
    print(i+1,fib(i))
```

Program that generates Fibonacci numbers

```python
memoized = {0: 1, 1: 1}

def fib(n):
    if (n in memoized):
        return(memoized[n])
    else:
        memoized[n] = fib(n-1) + fib(n-2)
        print("  Golden ratio=",fib(n-1)/fib(n-2))
        return(memoized[n])

for i in range(40):
    print(i+1,fib(i))
```

The ratio of successive Fibonacci numbers approaches the golden ratio. Here's a program to explore this connection.

```
1 1
2 1
3 2
4 3
5 5
6 8
7 13
8 21
9 34
10 55
11 89
12 144
13 233
14 377
15 610
16 987
17 1597
18 2584
19 4181
20 6765
21 10946
22 17711
23 28657
24 46368
25 75025
26 121393
27 196418
28 317811
29 514229
30 832040
31 1346269
32 2178309
33 3524578
34 5702887
35 9227465
36 14930352
37 24157817
38 39088169
39 63245986
40 102334155
```

```
1 1
2 1
     Golden ratio= 1.0
3 2
     Golden ratio= 2.0
4 3
     Golden ratio= 1.5
5 5
     Golden ratio= 1.6666666666666667
6 8
Golden ratio= 1.6
7 13
     Golden ratio= 1.625
8 21
     Golden ratio= 1.6153846153846154
9 34
     Golden ratio= 1.619047619047619
10 55
     Golden ratio= 1.6176470588235294
11 89
     Golden ratio= 1.6181818181818182
12 144
     Golden ratio= 1.6179775280898876
13 233
     Golden ratio= 1.6180555555555556
14 377
     Golden ratio= 1.6180257510729614
15 610
     Golden ratio= 1.6180371352785146
16 987
     Golden ratio= 1.618032786885246
17 1597
     Golden ratio= 1.618034447821682
18 2584
     Golden ratio= 1.6180338134001253
19 4181
     Golden ratio= 1.618034055727554
20 6765
     Golden ratio= 1.6180339631667064
21 10946
     Golden ratio= 1.6180339985218033
22 17711
     Golden ratio= 1.618033985017358
23 28657
     Golden ratio= 1.6180339901755971
24 46368
     Golden ratio= 1.618033988205325
25 75025
     Golden ratio= 1.618033988957902
26 121393
     Golden ratio= 1.6180339886704431
27 196418
     Golden ratio= 1.6180339887802426
28 317811
     Golden ratio= 1.618033988738303
29 514229
     Golden ratio= 1.6180339887543225
30 832040
     Golden ratio= 1.6180339887482036
31 1346269
     Golden ratio= 1.6180339887505408
32 2178309
     Golden ratio= 1.6180339887496482
33 3524578
     Golden ratio= 1.618033988749989
34 5702887
     Golden ratio= 1.618033988749859
35 9227465
     Golden ratio= 1.6180339887499087
36 14930352
     Golden ratio= 1.6180339887498896
37 24157817
     Golden ratio= 1.618033988749897
38 39088169
     Golden ratio= 1.618033988749894
39 63245986
     Golden ratio= 1.6180339887498951
40 102334155
```

Thus we have found that the ratio of successive terms of a Fibonacci sequence a_{n+1}/a_n, which is equal to b_n/a_n, converges to the Golden Ratio.

Lupascu Felicia

Ex.3

the resistance R of the thermistor and the temperature T is given by

$$\frac{1}{T} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3$$

$$\frac{1}{19.01 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3 \quad (1)$$

$$\frac{1}{18.99 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3 \quad (2)$$

$$\frac{1}{18.99+273.15} = 1.129241 \cdot 10^{-3} + 2.341077 \cdot 10^{-4} \ln(R) + 8.775468 \cdot 10^{-8} \{\ln(R)\}^3$$

$f(R) = 2.341077 \cdot 10^{-4} \ln(R) = 0$, as $R_{-1} = 14000$, $R_0 = 15000$

Iteration 1

The estimate of the root is

$$R_1 = R_0 - \frac{f(R_0)(R_0 - R_{-1})}{f(R_0) - f(R_{-1})}$$

$$f(R_0) = 2.341077 \cdot 10^{-4} \ln(R_0) + 8.775468 \cdot 10^{-8} \{\ln(R_0)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(15000) + 8.775468 \cdot 10^{-8} (\ln(15000))^3 - 2.293775 \cdot 10^{-3} =$$
$$= \underline{3.5383 \cdot 10^{-5}}$$

$$f(R_{-1}) = 2.341077 \cdot 10^{-4} \ln(R_1) + 8.775468 \cdot 10^{-8} \{\ln(R_1)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(14000) + 8.775468 \cdot 10^{-8} \{\ln(14000)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= \underline{1.7563 \cdot 10^{-5}}$$

$$R_1 = 1500 - \frac{(3.5383 \cdot 10^{-5})(15000 - 14000)}{(3.5383 \cdot 10^{-5})(-1.7563 \cdot 10^{-5})} = \underline{13014}$$

The absolute relative approximate error $|\epsilon_a|$ at the end of iteration 1

$$|\epsilon_a| = \left| \frac{R_1 - R_0}{R_1} \right| \cdot 100\% = \left| \frac{13014 - 15000}{13014} \right| \cdot 100 = \underline{15.257\%}$$

The number of significant digits at least correct is zero as we need an absolute relative approximate error of less than 5% for one significant digit to be correct in our result.

Lupascu Felicia

Iteration 2

The estimate of the root is

$$R_2 = R_1 - \frac{f(R_1)(R_1 - R_0)}{f(R_1) - f(R_0)}$$

$$f(R_1) = 2.341077 \cdot 10^{-4} \ln(R_1) + 8.775468 \cdot 10^{-5} \{\ln(R_1)\}^3 - 2.293775 \cdot 10^{-3} =$$

$$= 2.341077 \cdot 10^{-4} \{\ln(13014)\}^3 - 2.293775 \cdot 10^{-3} = \underline{13083}$$

The absolute relative approximate error $|\epsilon_a|$ at the end of iteration 2

$$|\epsilon_a| = \left|\frac{R_2 - R_1}{R_2}\right| \cdot 100 = \left|\frac{13083 - 13014}{13083}\right| \cdot 100\% = \underline{0{,}52422\%}$$

The number of significant digits at least correct is 1, because the absolute relative approximate error is less than 5%

Iteration 3

The estimate of the root is

$$R_3 = R_2 - \frac{f(R_2)(R_2 - R_1)}{f(R_2) - f(R_1)}$$

$$f(R_2) = 2.341077 \cdot 10^{-4} \ln(R_2) + 8.775468 \cdot 10^{-8} \{\ln(R_2)\}^3 - 2.293775 \cdot 10^{-3} =$$

$$= 2.341077 \cdot 10^{-4} \ln(13083) + 8.775468 \cdot 10^{-8} \{\ln(13083)\}^3 - 2.293775 \cdot 10^{-3} =$$

$$= \underline{8.8907 \cdot 10^{-8}}$$

$$R_3 = 13083 - \frac{(8.8911 \cdot 10^{-8})(13083 - 13014)}{(8.8911 \cdot 10^{-8}) - (-1.2658 \cdot 10^{-6})} = \underline{13078}$$

The absolute relative approximate error $|\epsilon_a|$ at the end of iteration 3

$$|\epsilon_a| = \left|\frac{R_3 - R_2}{R_1}\right| \cdot 100 = \left|\frac{13078 - 13083}{13078}\right| \cdot 100\% = 0.034415\%$$

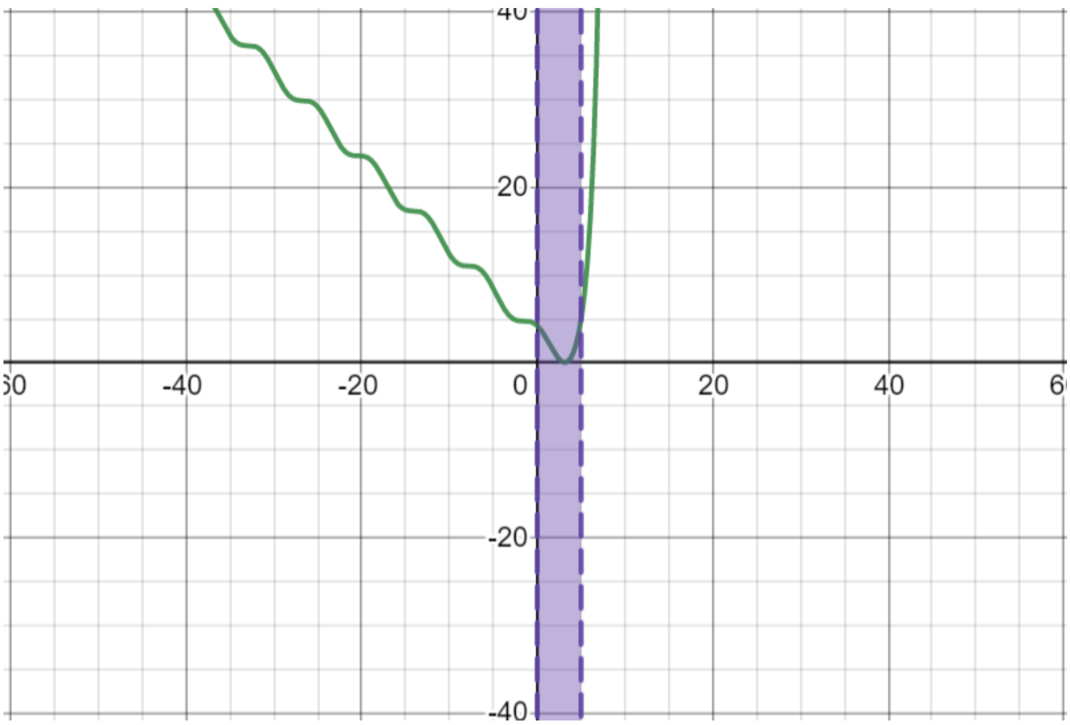
The number of significant digits at least correct is 3, because the absolute relative approximate error is less than 0.05%

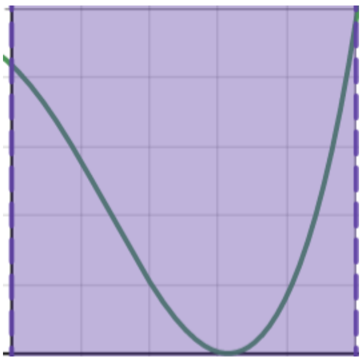
```matlab
clear all
clc
syms R
% write the equation to be solved
f = log(R)*2.341077*10^(-4)+ log(R)^3*8.775468*10^(-8)-2.293775*10^(-3);
% derivative of the function
g = diff(f);
n = 3; % number of decimal places
epsilon = 0.5*10^(2-n);
R0 = 15000; % initial guess
%%
for i=1:100
f0 = vpa(subs(f,R,R0)); %function value at R0
f0_der = vpa(subs(g,R,R0));%function derivative at R0
y = R0 - f0/f0_der; % update the next guess
err = abs(y-R0);
if err<epsilon
break
end
R0 = y;
end
%%
y = y - rem(y,10^-n)
%fprintf('The Root is : %f ',y);
```

Ex.3 Matlab

Lupascu Felicia

Ex.4





Order of convergence= 1.9617575539126844

It tends to 2 . It is quadratic.

c) $f(x) = e^{x-\pi} + \cos x - x + \pi$

$f'(x) = -\sin(x) + e^{x-\pi} - 1$

If the function has a multiplicity 2 or higher then it will converge slower than bisection, but if we take the derivative $f'(x)$ or higher, it will have the same solution as $f(x)$ and it will converge quadratically

```
function dummy=bisc_newton_secant()
clc;
clear all;

f=@(e^(x-pi)+cos(x)-x+pi); %function
fp=@(-sin(x)+e^(x-pi)-1); % derivative of unction

tol=1e-8;

a=0;
b=5; % interval
x0=5;
disp('Root by Bisection method')
y=bisecion(f,a,b,tol) % function calling

a1=0;
b1=5; % interval
disp('Root by fixed point method')
C1=fixedp(g,x0,tol) % function calling

disp('Root by Newton method')
y2=newt(f,x0,tol)% function calling

a2=0;
b2=5; % interval

disp('Root by Secant method')
y1=secn(f,a2,b2,tol)% function calling
```

Ex.5

a) $x_{n+1} = \cos x_n - 1 + x_n$

The formula gives $x_1 = \cos x_0 - 1 + x_0$

Plug in $x_0 = 0.1$ to find x_1

$x_1 = \cos(0.1) - 1 + 0.1 = 0.999984769$

$x_2 = \cos(0.999984769) - 1 + 0.999984769 = 0.9998324688$

$x_3 = \cos(0.9998324688) - 1 + 0.9998324688 = \dots$

- - - - - -

$x = f(x)$

$x = \cos(x) - 1 + x$

It gives 0 as a fixed point.

```cpp
// C++ program for implementation of Newton Raphson Method for
// solving equations
#include<bits/stdc++.h>
#define EPSILON 0.001
using namespace std;

// An example function whose solution is determined using
// Bisection Method. The function is xn+1 = cos xn − 1 + xn
double func(double x)
{
    return cos(x)-1+x;
}

// Derivative of the above function which is 1-sin(x)
double derivFunc(double x)
{
    return  1-sin(x);
}

// Function to find the root
void newtonRaphson(double x)
{
    double h = func(x) / derivFunc(x);
    while (abs(h) >= EPSILON)
    {
        h = func(x)/derivFunc(x);

        // x(i+1) = x(i) - f(x) / f'(x)
        x = x - h;
    }

    cout << "The value of the root is : " << x;
}

// Driver program to test above
int main()
{
    double x0 = -0.1; // Initial values assumed
    newtonRaphson(x0);
    return 0;
}
```

Ex.6

$$\lambda n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$$

n	x_n	$xn - xn-1$	λn
0	2.0		
1	2.1248	0.124834	
2	2.2148	0.089944	0,720705
3	2.2805	0.065698	0.72997777
4	2.3289	0.048386	0.736468797
5	2.3647	0.035827	0,74022727272
6	2.3913	0.026624	0.74567669172
7	2.4111	0.019835	0.74567669172
8	2.4260	0.014803	0,747626262
9	2.4370	0.011062	0,74224161073
10	2.4453	0.0082745	0,752227

$$\lambda_{n_3} = \frac{0{,}065698}{2.2148 - 2.1248} = 0{,}72997777$$

$$\lambda_{n_4} = \frac{0{,}048386}{2.2805 - 2.2148} = 0{,}73646879756$$

$$\lambda_{n_5} = \frac{0{,}035827}{2.3289 - 2.2805} = 0{,}740227272$$

$$\lambda_{n_6} = \frac{0{,}026624}{2{,}3647 - 2.3289} = 0{,}743687150838$$

$$\lambda_{n_7} = \frac{0{,}019835}{2{,}3913 - 2.3647} = 0{,}745676691 72$$

$$\lambda_{n_8} = \frac{0{,}014803}{2.4111 - 2.3913} = 0{,}747626262 62$$

$$\lambda_9 = \frac{0{,}011062}{2.4260 - 2.4111} = 0{,}742241610738$$

$$\lambda_{10} = \frac{0{,}0082745}{2.4370 - 2.4260} = 0{,}752227$$

a)An iterative method is called convergent if the corresponding sequence converges for given initial approximations. Out function is convergent because it tends to the initial value 2.

b) Yes.

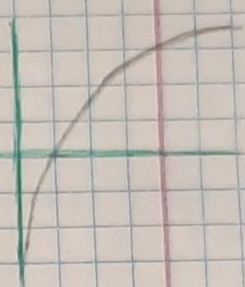
c) When the condition is satisfied, Newton's method converges, and it also converges faster than almost any other alternative iteration scheme based on other methods of coverting the original f(x) to a function with a fixed point.

Lupascu Felicia

Ex.7

Given function is $f(x) = x + \ln x = 0$.

Graph of $f(x) = x + \ln(x)$



domain $(0, \infty)$
Range $(-\infty, +\infty)$

① i) $x_1 = -\ln(x) = \varphi(x) \Rightarrow \varphi(x) = -\ln(x)$

$|\varphi'(x)| = |-\frac{1}{x}|$.

Therefore $|\varphi'(x)| = |\frac{1}{x}| \not< 1$ in its domain.

$\Rightarrow x$ can not converge

③ iii) $x = \frac{x + e^{-x}}{2}$

$\varphi(x) = \frac{x + e^{-x}}{2}$; $|\varphi'(x)| = \frac{1}{2}|1 - e^{-x}| < 1$ $\forall x$ in the domain of f.

$\Rightarrow x = \frac{x + e^{-x}}{2}$ will converge.

② ii) $x = e^{-x} = \varphi(x) = \varphi(x)$

$\varphi'(x) = |-e^{-x}| = |e^{-x}| < 1$ $\forall x$ in the domain of f

$\Rightarrow \varphi(x) = x = e^{-x}$ will converge definitely

② and ③ can be used to find the root of f.

A better formula to solve this equation by Newton-Raphson meth.

Newton-Raphson method is given by:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

We have $f(x) = x + \ln(x)$
$f'(x) = 1 + \frac{1}{x}$
$a_0 = 0.5$

Putting $n = 0$ in the above iterative formula we get

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.5 - \frac{0.5 + \ln(0.5)}{1 + \frac{1}{0.5}} = 0.5 - \frac{-0.1931478}{3}$$

i.e. x_1 = first iteration $= 0.564382393$

Second iteration

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.567138987 - \frac{0.567138987 + \ln(0.567138987)}{1 + \frac{1}{0.567178987}} =$$

$$= 0.567145831$$

Rellative error $= |x_3 - x_2| = 0.000006844543222$

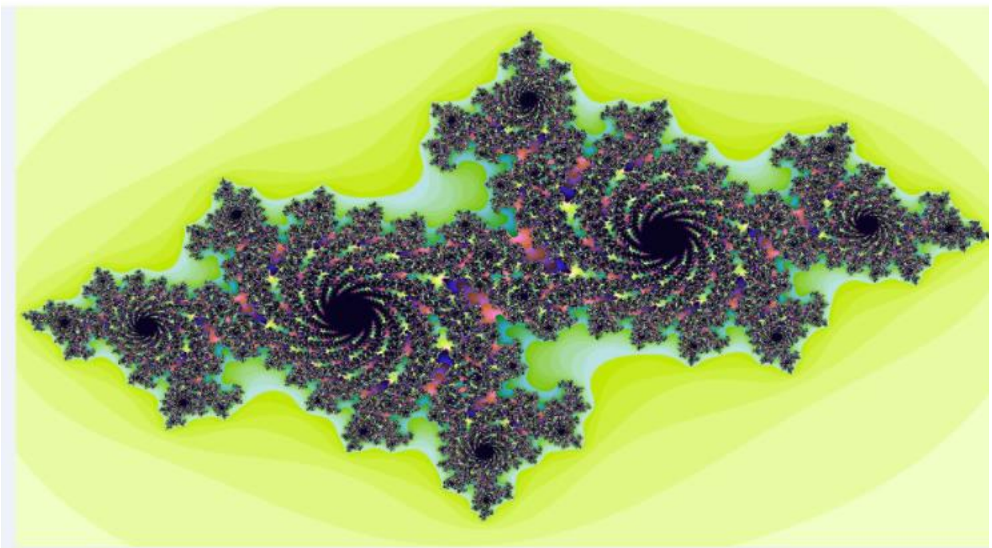
$x_3 = 0.567145831$ is correct up to 5 digits.

x_3 is a root of the eq. $x + \ln(x) = 0$.

Lupascu Felicia

Ex.9

```python
# Python code for Julia Fractal
from PIL import Image

# driver function
if __name__ == "__main__":

    # setting the width, height and zoom
    # of the image to be created
    w, h, zoom = 1920,1080,1

    # creating the new image in RGB mode
    bitmap = Image.new("RGB", (w, h), "white")

    # Allocating the storage for the image and
    # loading the pixel data.
    pix = bitmap.load()

    # setting up the variables according to
    # the equation to create the fractal
    cX, cY = -0.7, 0.27015
    moveX, moveY = 0.0, 0.0
    maxIter = 255

    for x in range(w):
        for y in range(h):
            zx = 1.5*(x - w/2)/(0.5*zoom*w) + moveX
            zy = 1.0*(y - h/2)/(0.5*zoom*h) + moveY
            i = maxIter
            while zx*zx + zy*zy < 4 and i > 1:
                tmp = zx*zx - zy*zy + cX
                zy,zx = 2.0*zx*zy + cY, tmp
                i -= 1

            # convert byte to RGB (3 bytes), kinda
            # magic to get nice colors
            pix[x,y] = (i << 21) + (i << 10) + i*8

    # to display the created fractal
    bitmap.show()

```



Ex.8

n	x_n	$x_n - x_{n-1}$
0	1.00	
1	0.36788	$-6.3212E-01$
2	0.69220	$3.2432E-01$
3	0.50047	$-1.9173E-01$
4	0.60624	$1.0577E-01$
5	0.54540	$-6.0848E-02$
6	0.57961	$3.4217E-02$

8.

$$\lambda_2 = \frac{-6.3212E-01}{0{,}36788-1} =$$

$$\lambda_3 = \frac{-1{,}9173E-01}{0{,}69220-0{,}36788} =$$

$$\lambda_4 = \frac{1{,}0577E-01}{0{,}50047-0{,}69220} = 1{,}0577e + \frac{100000}{19173}$$

$$\lambda_5 = \frac{-6{,}0848E-02}{0{,}60624-0{,}50047} = -6{,}048e - \frac{100000}{5291}$$

$$\lambda_6 = \frac{3{,}4217E-02}{0{,}54540-0{,}600624} \approx 3{,}4217e + \frac{250000}{6903}$$

$f'(R)=0$, so the sequence converges linearly to the fixed point.

We can also use Aitken extrapolation formula