Homework 2

Due February 27, 19:00

Problem 2.1

The **error function** (also called Gauss error function) is defined by

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

Use Taylor series to approximate $\operatorname{erf}(x)$ with a polynomial $T_n(x)$. What is n, if the desired accuracy is 10^{-6}? Using this approximation, plot the graph of $\operatorname{erf}(x)$ on $[-3, 3]$.

Problem 2.2

Consider the sequence of Fibonacci numbers F_n:

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, \ldots$$

Let $R_n = \frac{F_{n+1}}{F_n}$. It can be shown that

$$\lim_{n\to\infty} R_n = \frac{1+\sqrt{5}}{2} \equiv \phi,$$

which is known as **golden ratio**. Write a code that will compute numerically the first 50 terms of the sequence R_n together with errors $\phi - R_n$. In computations make sure that you are using IEEE double precision. Comment your results. What can be said on the order of convergence?

Problem 2.3

Thermistors are temperature-measuring devices based on the principle that the thermistor material exhibits a change in electrical resistance with a change in temperature. By measuring the resistance of the thermistor material, one can then determine the temperature. For a 10K3A Betatherm thermistor, the relationship between

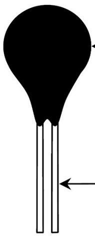

Figure 1: A typical thermistor

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

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

where T is in Kelvin and R is in Ohms, and log denotes the natural logarithm. A thermistor error of no more than $\pm 0.01^o C$ is acceptable. To find the range of the resistance that is within this acceptable limit at $19^o C$, we need to solve

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

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

Write a computer routine implementing Newton's method and solve equations (1) and (2) using Newton's method with initial guess $R_0 = 15000$ and error tolerance of 10^{-6}.
What is the obtained range for resistance values?

Problem 2.4

Consider the function $f(x) = e^{x-\pi} + \cos x - x + \pi$.

a) Plot its graph on interval $[0, 6]$.

b) Apply Newton's method routine you developed in **Problem 2.3** to solve equation $f(x) = 0$ on interval $[0, 6]$. What can be said about its order of convergence? Argue why this is happening? How its convergence order can be improved?

c) Write a modified routine that will ensure **quadratic** convergence and apply it.

d) Instead of solving $f(x) = 0$, try to apply the fixed point iterations $x_{n+1} = e^{x_n - \pi} + \cos x_n + \pi$. Comment on your results.

Problem 2.5

a) Compute the fixed point $x_{n+1} = \cos x_n - 1 + x_n$ with initial guess $x_0 = 0.1$.

b) What can be said about the speed of convergence? Compare it with bisection method.

c) Write a modified computer routine that will speed up the convergence.

Problem 2.6

Newton's method is used to find the root α of $f(x) = 0$. The first 10 iterates are shown in the table below.

(1) What can be said about the order of convergence? Is it slower or faster than bisection method?
(2) What can be said about the root α to explain this convergence?
(3) Knowing function $f(x)$, how would you speed up the convergence?

n	x_n	$x_n - x_{n-1}$
0	2.0	
1	2.1248	0.124834
2	2.2148	0.089944
3	2.2805	0.065698
4	2.3289	0.048386
5	2.3647	0.035827
6	2.3913	0.026624
7	2.4111	0.019835
8	2.4260	0.014803
9	2.4370	0.011062
10	2.4453	0.0082745

Problem 2.7

For solving the equation $x + \ln x = 0$, there were proposed three methods:

$$\begin{aligned} (a) \quad x &= -\ln x \\ (b) \quad x &= e^{-x} \\ (c) \quad x &= \frac{x + e^{-x}}{2} \end{aligned}$$

(1) Which of the formulas can be used?
(2) Which of the formulas should be used?
(3) Give an even better formula!

Problem 2.8

Consider the following table of iterates from an iteration method which is convergent to a fixed point α of the function $g(x)$:

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	$3.4217E-02$

(1) Show that this is a linearly convergent iteration method.
(2) Find its rate of linear convergence. Is this method faster or slower than bisection method?
(3) Propose a way to accelerate the convergence of this method?

Problem 2.9

BONUS. Benout B. Mandelbrot, a famous mathematician is known as the inventor of *fractals* and this problem is dedicated to him. In this problem you will generate the so-called **quadratic Julia Sets**, a well-known fractal example.

Given two complex numbers, c and z_0 the following recursion (it is similar to fixed point iterations) is defined

$$z_n = z_{n-1}^2 + c.$$

For an arbitrary given choice of c and z_0, this recursion leads to a sequence of complex numbers z_1, z_2, $z_3, \ldots$ called the **orbit** of z_0. Depending on the exact choice of c and z_0, a large range of orbit patterns are possible.

For a given fixed c, most choices of z_0 yield orbits that tend towards infinity. (That is, $|z_n| \to \infty$ as $n \to \infty$).

For some values of c certain choices of z_0 yield orbits that eventually go into a periodic loop. Finally, some starting values yield orbits that appear to dance around the complex plane, apparently at random (an example of chaos). These initial values of z_0 make up the Julia set of this recursion, denoted by J_c.

Write a MATLAB/GNU Octave/Python script that visualizes a slightly different set, called the filled-in Julia set denoted by K_c, which is the set of all z_0 with orbits which do not tend towards infinity. The "normal" Julia set J_c is the edge of the filled-in Julia set. The figure below illustrates a filled-in Julia Set for one particular value of c.

a) It has been shown that if $|z_n| > 2$ for some n, then it is guaranteed that the orbit will tend to infinity. The value of n for which this becomes true is called the **escape velocity** of a particular z_0. Write a function that returns the escape velocity of a given z_0 and c. The function declaration should be: $n =$`EscVel`(z_0, c, N), where N is the maximum allowed escape velocity (i.e. if $|z_n| \leq 2$ for $n < N$, return N as the escape velocity, so you will prevent infinite loops).

b) To generate the filled-in Julia Set, write the following function

$$M = \texttt{JuliaSet}(z_{Max}, c, N),$$

where z_{Max} will be the maximum of the real and imaginary parts of the various values of z_0 for which we will compute escape velocities, c and N are the same as defined above, and M is the matrix that contains the escape velocity of various z_0.

- In this function, you first want to make a 500×500 matrix that contains complex numbers with real part between $-z_{Max}$ and z_{Max}, and imaginary part between $-z_{Max}$ and z_{Max}. Call this matrix Z. Make the imaginary part vary along the y-axis of this matrix. You can most easily do this by using `linspace` and `meshgrid` commands from MATLAB/GNU Octave, but you can also do it with a loop.
- For each element of Z, compute the escape velocity (by calling your `EscVel` function) and store it in the same location in a matrix M. When done, the matrix M should be the same size as Z and contain escape velocities with values between 1 and N.
- Run your `JuliaSet` function with various z_{Max}, c and N values to generate various fractals. To display the fractal nicely, use `imagesc` command to visualize $\arctan(0.1 * M)$, (taking the arctangent of M makes the image look nicer, you also can use `axisxy` command so that y values aren't flipped).

The figure below shows a Julia Set for $c = -.297491 + i * 0.641051$.

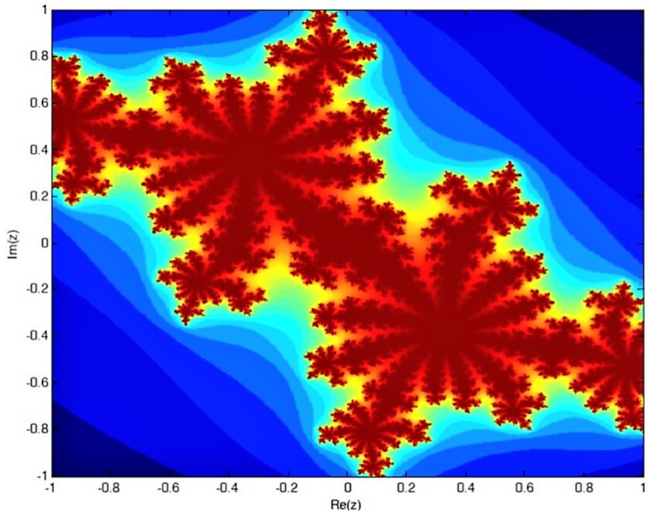

Figure 2: $M =$`JuliaSet`$(1, -.297491 + i * 0.641051, 100)$

Julia Sets are the boundaries of more general Mandelbrot sets. There is a chapter in Clive Moler textbook "Experiments with MATLAB" `www.mathworks.com/moler/exm/chapters.html` dedicated to Mandelbrot sets. The difference between Julia Sets and Mandelbrot set is presented in `www.karlsims.com/julia.html`, the figure below was taken from there.

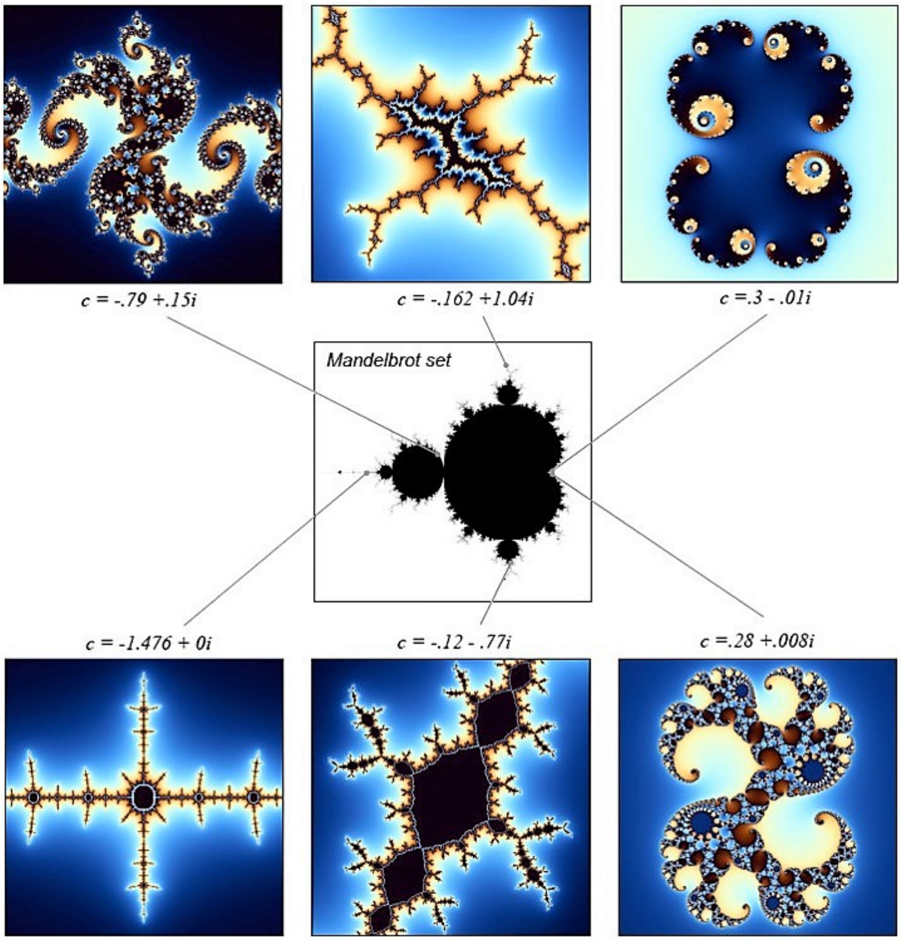

Figure 3: Different Julia Sets and their relation to Mandelbrot Set.