Homework 1: Computational Physics

Pablo Lopez Duque

September 2020

1 Problem 1

In this problem, you should use C++, Matlab,and Python, producing source code and 3 figures on log-log axes. For $a = b = 1$, and with $x_0 = 0.1$, the exact solution to the logistic equation is $x(t) = 1/(1 + 9e^{-t})$. Compute $\delta(\Delta t) = |x'(t) - \frac{x(t+\Delta t)-x(t)}{|\Delta t}|$ for $t = 2$ and for $\Delta t = 10^{-n}$.
Source codes are included in the GitHub, whereas the plots are included in figure 1. As it can be seen, the error decreases until a certain value and then start increasing again. This is due to the finite size of a float point variable, which will be analyzed in problem 2. For Matlab and C++, the behavior is identical, with the minimum at 10^{-8}, whereas for python, the minimum is around 10^{-7}. This difference is most likely system dependent, so running it in different computers will render different results.

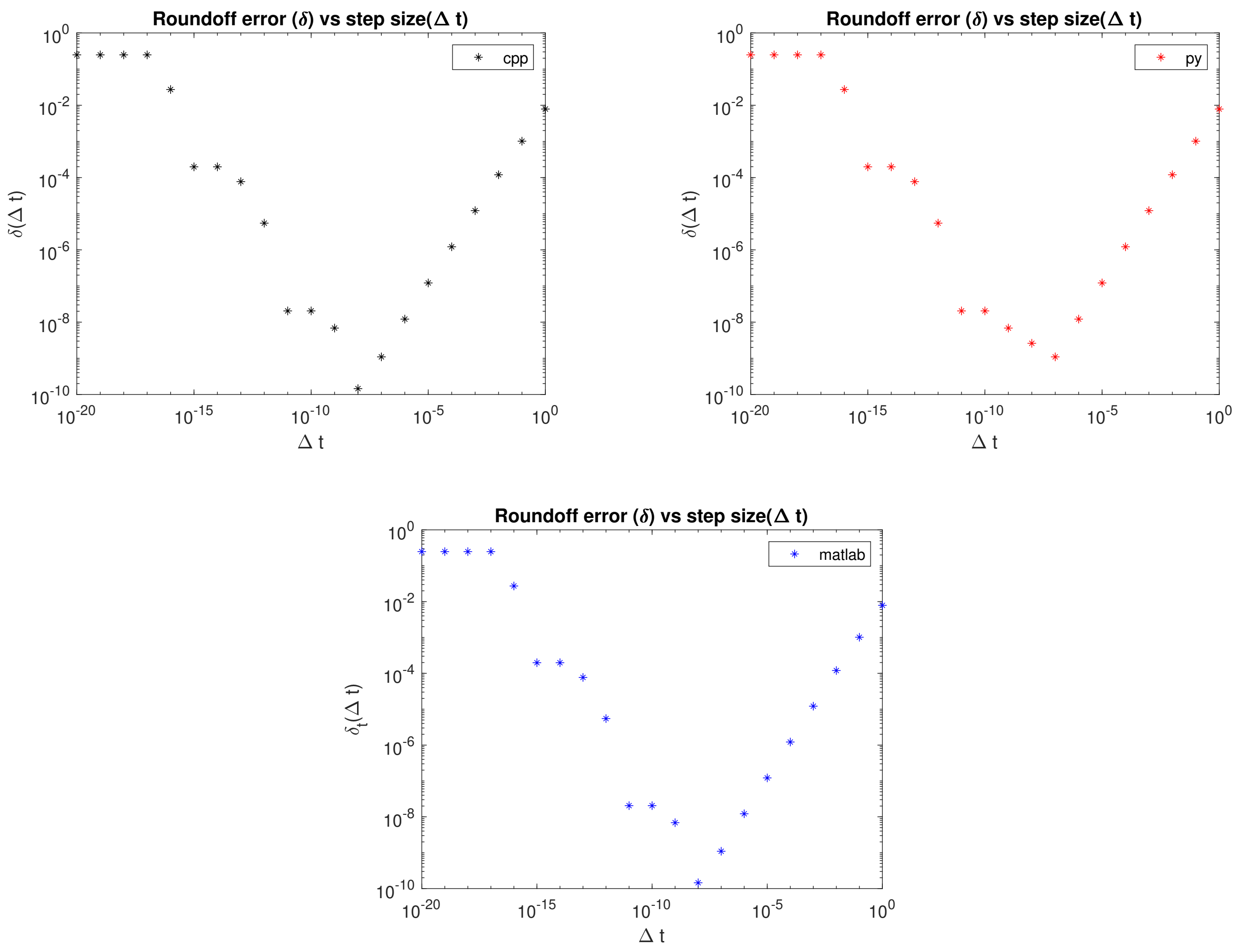


Figure 1: Absolute error vs step size(Δt). The minimum error is attained for a step size of 10^{-8}, but then increases due to round off effects.

2 Problem 2

Machine numbers have finite size, and not all languages have the same max sizes.

(a) **In C++, determine the largest power of 2 that can be represented of *int* type. Do so by defining *int* $x = 1$, and iteratively multiplying by 2 until something goes wrong. Describe what happened to indicate a problem.**

Due to C++ architecture, the maximum *int* type variable that can be represented is $1073741824 = 2^{30}$. In this case, after the maximum integer, all the following numbers appear as 0 in the console generating an infinite loop. The code includes an easy fix for this issue. The fix was devised after seeing the results when entering the infinite loop.

(b) **In C++, determine the largest power of 2 that can be represented of *double* type. Do so by defining *double* $x = 1.0$, and iteratively multiplying by 2.0 until something goes wrong. Describe what happened.**

In C++, the maximum *double* type variable that can be represented is $8.98847e + 307 = 2^{1023}$.In this case, after the maximum double, all the following numbers appear as "inf" in the console generating an infinite loop. The code includes an easy fix for this issue. The fix was devised after seeing the results when entering the infinite loop.

(c) **In Matlab, define $x = 1$ and iteratively multiply by 2 until something goes wrong. Describe what happened.**

In Matlab, the largest *int* variable that can be represented is $8.9885e+307 = 2^{1023}$. After that, the console returns "inf". Again, an easy fix to avoid entering an infinite loop was set up after looking at the results when entering in the infinite loop.

(d) **In Matlab, define $x = 1.0$ and iteratively multiply by 2.0 until something goes wrong. Did this fail at the same power of 2 as in (c)?**

In Matlab, as in C++, the largest *double* variable that can be represented is $8.9885e + 307 = 2^{1023}$. After that, the console returns "inf". Again, an easy fix to avoid entering an infinite loop was set up as in (c).

(e) **In Python, repeat the calculation in (c-d). Since both use weak typing, is the behavior in Python the same as in Matlab?**

In Python, there was no maximum reached even after looping until 2^{8*10^6} for a variable of *int* type. This is expected in Python3, since the limit present in Python2 was removed; now the maximum size is system dependent. However, for a *double* type the maximum was exactly the same as in Matlab and C++, $8.9885e + 307 = 2^{1023}$.

Notice: source codes for each part of the problem are uploaded to the GitHub.

3 Problem 3

Gram-Shmidt orthogonalization (GSO) produces two orthogonal vectors $\vec{v1}$ and $\vec{v2}$ from two arbitrary vectors $\vec{a}$ and $\vec{b}$, by setting $\vec{v1} = \vec{a}$ and $\vec{v2} = (\vec{b} - \frac{\vec{a}.\vec{b}}{a^2}\vec{a})c$, where c ensures $|\vec{v2}| = |\vec{b}|$. Implement GSO for two 3-dimensional vectors as a function in C++, Matlab, and Python. You may implement your data structures as arrays or vectors in C++, lists or numpy arrays in Python, and as arrays in Matlab.

Source code is included in the GitHub.

4 Problem 4

The scripts *cerrors.cpp* and *pyerrors.py* (uploaded to Teams) are intended to produce a 10×10 matrix $m_{ij} = |i - j|^2$ and print the elements to the terminal. However, both scripts suffer from common programming errors. Debug both scripts so that they produce the correct output.

```
#include <iostream>
//Some systems or IDEs may require "#include <cstdlib>" to define abs()

double squared(double x);

#define N 10

int main(){

        double matrix[10][10]={0};

        for(int i=0;i<N;i++){
                for(int j=0;j<N;i++){  //INCORRECTLY INCREASING i INSTEAD OF j
                        double value=((double)(abs(i-j)));  //VALUE WAS NOT INITIALIZED
                        matrix[i][j]=squared(value);
                }
        }

        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                        std::cout<<matrix[i][j]<<" ";
                }
                std::cout<<"\n";
        }
        return 0;
}
```

```
import numpy as np
import math

n=10

matrix=np.zeros((n,n))

for i in range(n):
        for j in range(n):
                matrix[i,j]=power(abs(i-j),2)  #pow instead of power

for i in range(n):              #whole loop can be replaced with print(matrix)
        for j in range(n):
                print(str(matrix(i,j))+" ",end="")      #() instead of [] for the matrix index
        print("")
```

Notice: source codes with the correct implementation for each language are uploaded to the GitHub.

5 Problem 5

The dynamics of the approach of the logistic map to a fixed point or attractor may be of interest in many contexts. In this problem, you may use your preferred language for this problem (you don't need to use all 3), and are free to use any library or modify any source code you wish. If you are using the code from class, $logistic0.x$(where $.x$ is your preferred language's extension) might be the most natural place to start.Plot the first 50 iterations of the logistic map, starting at $x = 0.01$, for $r = 2.00$ and 2.99 on the same figure. The data from each value of r should be connected with a line, be marked by a different symbol, and have a different color. These values of r have a single stable fixed point. Is the fixed point approached reached immediately or does the logistic map oscillate around the fixed point?

For $r = 2.00$, the fixed point is approached without oscillation, whereas for $r = 2.99$ the approach is oscillatory as depicted in figure 2. One must remember that we showed the fixed point is stable for $1 < r < 3$, so in this case just by plotting more iterations, this becomes clear as fhown in figure 2b.

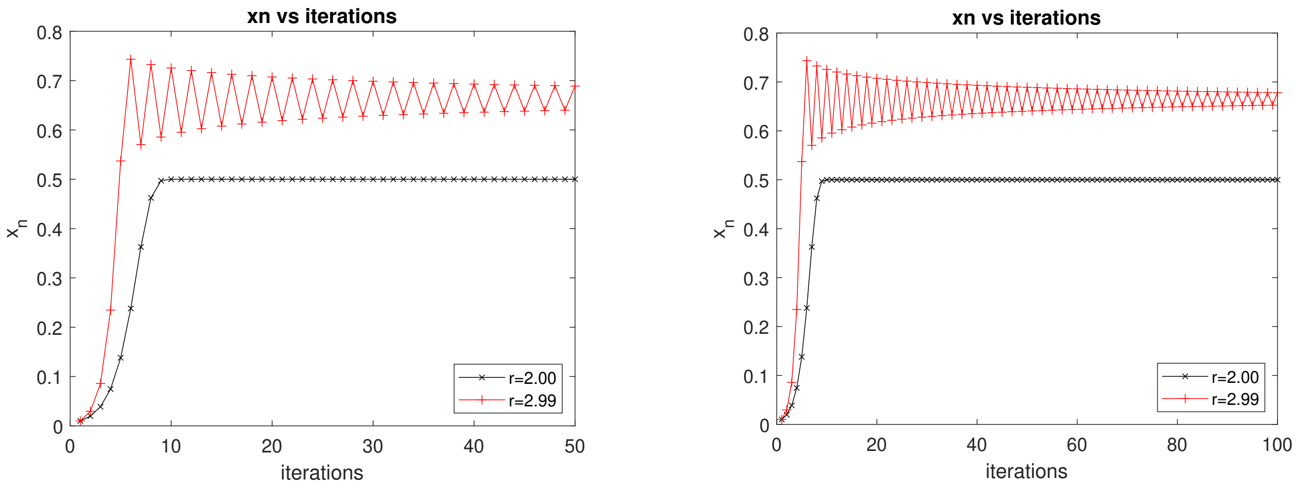


Figure 2: Logistic map, starting at $x = 0.01$, for $r = 2.00$ and 2.99.

6 Problem 6

It is not necessarily obvious if a nonlinear map will exhibit chaos, as you will demonstrate in this problem. You may use your preferred language (you don't need to use all 3), and can use any code from class or from a library you are familiar with. If you use code from class, $logistic1.x$ is the most natural place to start.

(a) **Show analytically that the map $x_{n+1} = r\frac{x_n}{1+x_n^2}$ has a stable fixed point for all $r > 1$.**

In order to show stability, we first must find the fixed point for this particular system: After a "long enough" time, when we reach a steady state, clearly, the value for $x_{n+1} = x_n$. Thus, we reach a fixed point.

$$x_\infty = r\frac{x_\infty}{2 + x_\infty}$$

$$\Leftrightarrow x_\infty^2 = r - 1 \Rightarrow x^* = \sqrt{r-1}$$

It is clear that there will be a fixed point as long as $r > 1$. Now, let's prove that this fixed point is indeed stable for any $r > 1$. We know that:

$$x_{n+1} = f(x_n)$$

So, for a fixed point, $x^* = f(x^*)$. So allowing an infinitesimal variation;

$$x_{n+1} = x^* + \epsilon_{n+1} = f(x_n) = f(x^* + \epsilon_n)$$

$$\Rightarrow x_{n+1} = f(x^*) + f'(x^*)\epsilon_n + \mathcal{O}(\epsilon_n^2)$$

$$\Rightarrow x_{n+1} \approx x^* + f'(x^*)\epsilon_n$$

So, in order to have stability, we must impose $|f'(x^*)| < 1$.

$$f'(x^*) = \frac{r(1 + x_n^2 - x_n * 2x_n)}{(1 + x_n^2)^2}$$

By replacing the value of the fixed point:

$$\Leftrightarrow f'(x^*) = \frac{r(1 - x_n^2)}{(1 + x_n^2)^2} = \frac{2}{r} - 1 \Rightarrow -1 < \frac{2}{r} - 1 < 1$$

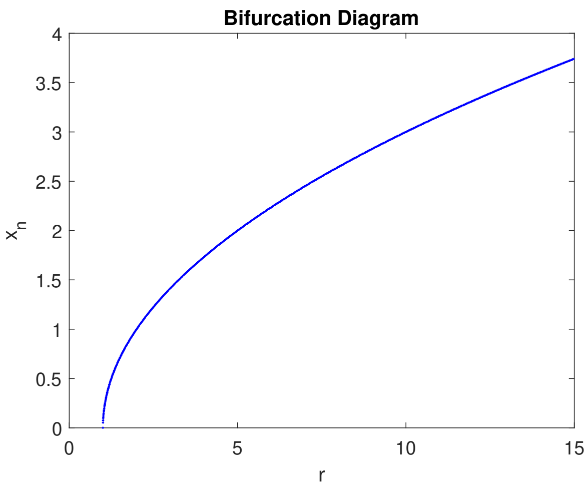
which leads to $0 < \frac{2}{r} < 2$.

$$\Rightarrow \frac{2}{r} < 2 \Rightarrow r > 1, and, 0 < \frac{2}{r} \Rightarrow r > 0$$

Hence, we have stability for $r > 1$.

(b) **Compute the bifurcation diagram for the map $x_{n+1} = r\frac{x_n}{1+x_n^2}$ for $1 \leq r \leq 15$ using any numerical method you like. Comment on whether your numerical results are consistent with (a). You should provide source code and a figure of the bifurcation diagram with labeled axes in the solutions.**

The numerical results show the expected behavior for this map. As it can be seen on figure 3(left), all fixed points are stable.



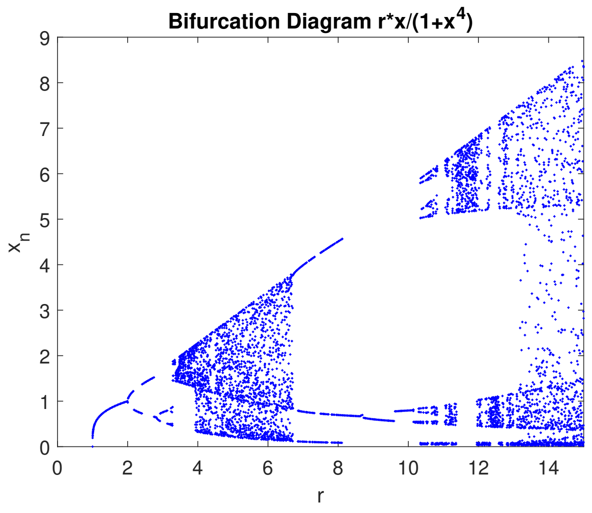


Figure 3: LEFT: Bifurcation diagram for the map $x_{n+1} = r\frac{x_n}{1+x_n^2}$ that shows stability for any value of $r > 1$ up to $r = 15$. RIGHT: Bifurcation diagram for the map $x_{n+1} = r\frac{x_n}{1+x_n^2}$ that shows stability for $1 < r < 2$ and bifurcations for $r > 2$.

(c) **Compute the bifurcation diagram for the map $x_{n+1} = r\frac{x_n}{1+x_n^4}$ for $1 \leq r \leq 15$. You do not need to analytically compute stability for this map, as it is quite tedious. Comment on any similarities or differences you see for the logistic bifurcation diagram or the results from (b).**

In this case, the map shows that, for a particular r, there are more than 1 fixed points for $r > 2$. Thus, bifurcations with different periods appear(figure 3(right)). We can also show this mathematically as follows:

Again, we must impose $|f'(x^*)| < 1$.

$$f'(x^*) = \frac{r(1 + x_n^4 - x_n * 4x_n^3)}{(1 + x_n^4)^2} = \frac{r(1 - 3x_n^4)}{(1 + x_n^4)^2}$$

By replacing the value of the fixed point:

$$\Leftrightarrow f'(x^*) = \frac{4}{r} - 3 \Rightarrow -1 < \frac{4}{r} - 3 < 1$$

which leads to $2 < \frac{2}{r} < 4$.

$$\Rightarrow \frac{4}{r} < 4 \Rightarrow r > 1, and, 2 < \frac{4}{r} \Rightarrow r < 2$$

Hence, we have stability for $1 < r < 2$. One could determine the r for the period 2 bifurcations as well, but this becomes more mathematically cumbersome.

7 Problem 7

Natural units for solving Schrodinger's equation for an electron in a potential are units of eV or energy, the electron mass m_e for the unit mass, and taking $\hbar \approx 1$. In this case, we would simply solve the differential equation $-\frac{1}{2}\partial_x^2\Psi + V(x)\Psi = i\partial_t\Psi$. What are the units of distance and time in this simulation? Put another way, what does the dimensionless position $x = 1$ or dimensionless time $t = 1$ correspond to in the physical units of m or s?

The standard form of the Schrödinger equation (SE) is:

$$\left(-\frac{\hbar^2}{2m}\partial_x^2 + V(x)\right)\Psi = i\hbar\partial_t\Psi$$

In order to have an adimensional equation, we must absorb the constants $-\frac{\hbar^2}{m}$ and $\hbar$ inside x and t respectively. First, lets rewrite the SE in the most conventional fashion:

$$\left(-\frac{\hbar^2}{2m}\frac{\partial}{\partial x} + V(x)\right)\Psi = i\hbar\frac{\partial}{\partial t}\Psi \Leftrightarrow \left(-\frac{1}{\frac{2m}{\hbar^2}}\frac{\partial}{\partial x} + V(x)\right)\Psi = i\frac{1}{\frac{1}{\hbar}}\frac{\partial}{\partial t}\Psi$$

Hence, how to absorb the constants becomes clear if we write:

$$\left(-\frac{1}{2}\frac{\partial}{\partial(\frac{m}{\hbar^2}x)} + V(x)\right)\Psi = i\frac{\partial}{\partial(\frac{1}{\hbar}t)}\Psi$$

Let's introduce two variables $x_c = \frac{m}{\hbar^2}x$ and $t_c = \frac{1}{\hbar}t$:

$$\left(-\frac{1}{2}\frac{\partial}{\partial x_c} + V(x)\right)\Psi = i\frac{\partial}{\partial t_c}\Psi$$

Which is exactly the form we wanted to achieve for computational calculations. Hence, in order to convert back from this units to conventional SI units, one just needs the definitions of x_c and t_c. The corresponding SI units for each are:

$$[x_c] = \frac{kg}{J^2s^2}m = \frac{kg}{\frac{kg^2m^4}{s^4}s^2}m = \frac{s^2}{kgm^3} = s^2kg^{-1}m^{-3}$$

and,

$$[t_c] = \frac{1}{Js}s = \frac{1}{\frac{kgm^2}{s^2}}s = \frac{s^3}{kgm^2} = s^3kg^{-1}m^{-2}$$