**PHYS 6350 - Computational Physics**
**FALL 2020**
**Solution to Assignment 1 and 2**
September 16, 2020
*Submitted by* : Showren Datta, *Student ID.* : 1906712

## Answer to Question no. 1

*Language* : C++
*Source code* : **problem_1.cpp** and **plot_cppdata.m**
*Produced figure* :

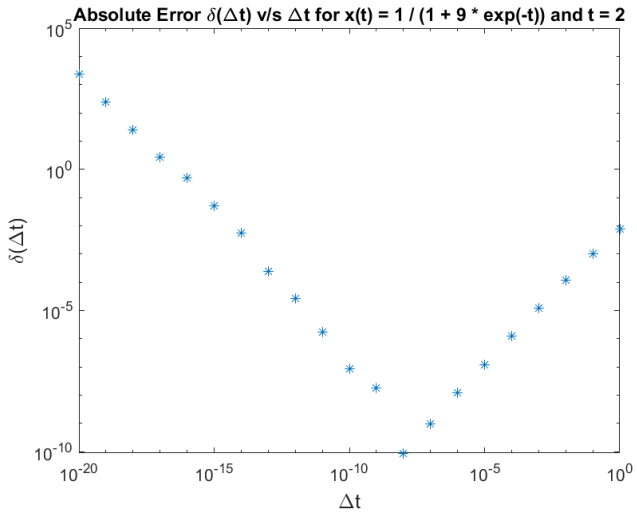

Figure 1:

*Language* : Python
*Source code* : **problem_1.py**
*Produced figure* :

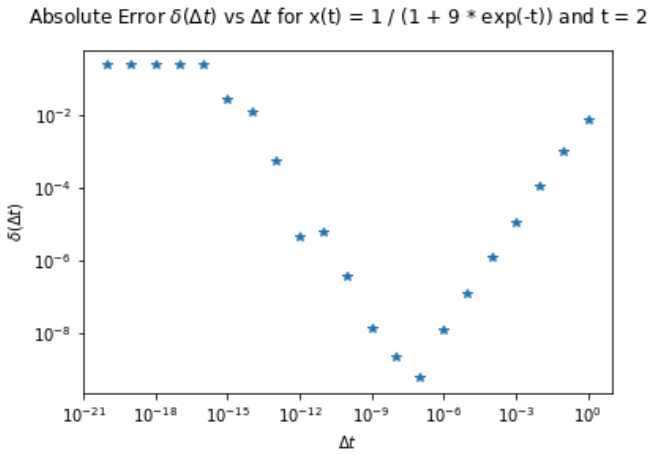

Figure 2:

*Language* : MATLAB
*Source code* : **problem_1.m**
*Produced figure* :

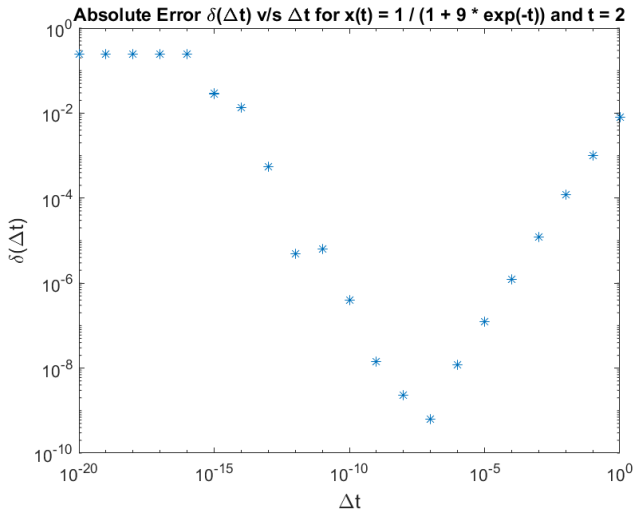

Figure 3:

## Answer to Question no. 2

(a) In C++, the largest power of 2 that can be represented of int type is 30. On the 31st iteration the product becomes negative, which is not supposed to happen. [*Source Code:* problem_2a.cpp]

(b) On the 1024-th iteration the product becomes Inf. i.e. infinity, because it surpasses the largest finite floating-point number *doublemax*(=1.7977 * $10^{324}$). Until this happens 2 can be multiplied 1023 times. [*Source Code:* problem_2b.cpp]

(c) On the 1024-th iteration the product becomes Inf. i.e. infinity, because it surpasses the largest finite floating-point number *realmax*(=1.79769 * $10^{308}$). Until this happens 2 can be multiplied 1023 times. [*Source Code:* problem_2c.m]

(d) On the 1024-th iteration the product becomes Inf. i.e. infinity, because it surpasses the largest finite floating-point number *realmax*(=1.79769 * $10^{308}$). Until this happens 2.0 can be multiplied 1023 times, which means it failed at the same power of 2 as in (c). [*Source Code:* problem_2d.m]

(e) No; Integer iterations didn't fail at same points as (c) and (d). Using data type *int64*, the integer iteration fails at the 63rd step. Therefore 2 can be multiplied 62 times. However, in case of float iteration, using data type *float64*, it fails on the 1024-th step, just like (c) and (d), which means 2.0 can be multiplied 1023 times. [*Source Code:* problem_2e.py]

## Answer to Question no. 3

*Language* : C++
*Source code* : problem_3.cpp

*Language* : Python
*Source code* : problem_3.py

*Language* : MATLAB
*Source code* : problem_3.m

## Answer to Question no. 4

*Language* : C++

*Corrections:* Missing header files $\#include < math.h >$ and $\#include < stdlib.h >$ were added. Variable *value* was initialized and declared *double* outside first nested *for-loop*. Within first nested *for-loop* increment of j was changed into "j++" from "i++". And finally in the function definition, type of input variable "x" was changed from *int* to *double*

*Source code* : corrected_c_errors.cpp

*Language* : Python

*Corrections:* Within the first nested *for-loop* "$power(abs(i-j), 2)$" was changed to "$math.pow(abs(i-j), 2)$". And in the second nested *for-loop* "$print(str(matrix(i, j)) + "", end = "")$" was changed into "$print(str(matrix[i, j]) + "", end = "")$"

*Source code* : corrected_py_error.py

## Answer to Question no. 5

*Language* : MATLAB
*Source code* : problem_5.m

For $r = 2.0$ the fixed point approached reached on the 10th iteration and for the first 9 iterations there was no oscillation either. It was just approaching towards the stable fixed value of 0.500.
On the other hand, for $r = 2.99$ the values have been oscillating from the second iteration. And after 50 iterations the fixed point was not reached yet, but surely was converging towards some point in between 0.6399 and 0.6890

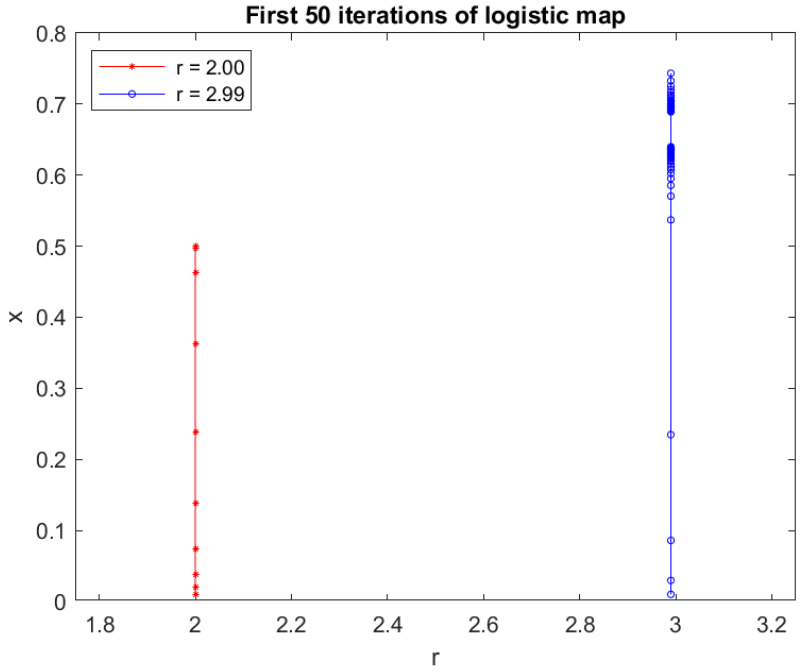


Figure 4:

## Answer to Question no. 6

(a) To converge on a stable fixed point, we must have $x_{n+1} = x_n$; which eventually produces,

$$x_\infty = r * \frac{x_\infty}{1 + x_\infty^2}$$
$$=> x_\infty^2 = (r - 1)$$

Again, for a stable fixed point $|f'(x^*)| < 1$. Here, $|f'(x)| = r * \frac{(1 - x^2)}{(1 + x^2)^2}$

Therefore,

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

And, so

$$|\frac{2 - r}{r}| < 1$$
$$=> \text{-1} < \frac{2 - r}{r} < 1$$
$$=> 0 < \frac{2}{r} < 2$$
$$=> 0 < \frac{1}{r} < 1$$
$$=> r > 1$$

That means the map $x_{n+1} = r * \frac{x_n}{1 + x_n^2}$ has a stable fixed point for all $r > 1$

(b) *Language* : MATLAB
*Source code* : problem_6b.m

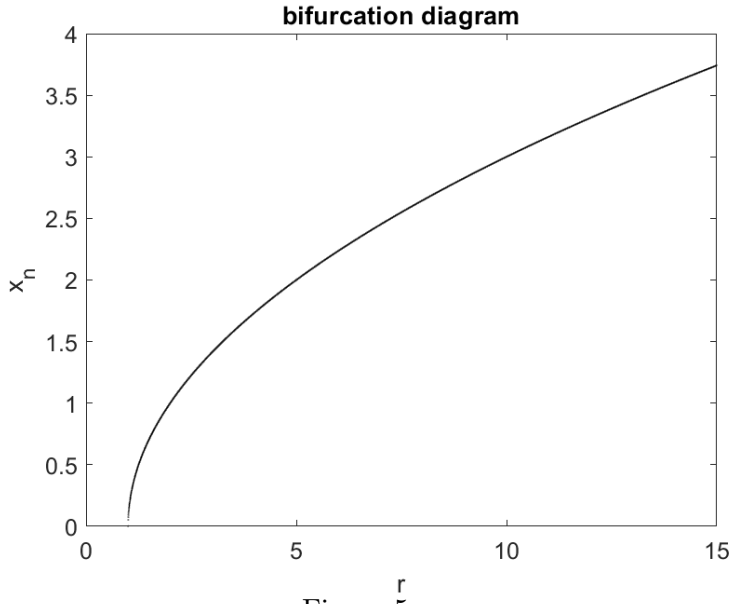


Figure 5:

And the figure shows that, for $r > 1$, there are stable fixed points for the map $x_{n+1} = r * \frac{x_n}{1 + x_n^2}$. But we observe no period doubling for this map within $1 \leq r \leq 15$.

(c) *Language* : MATLAB
*Source code* : problem_6c.m

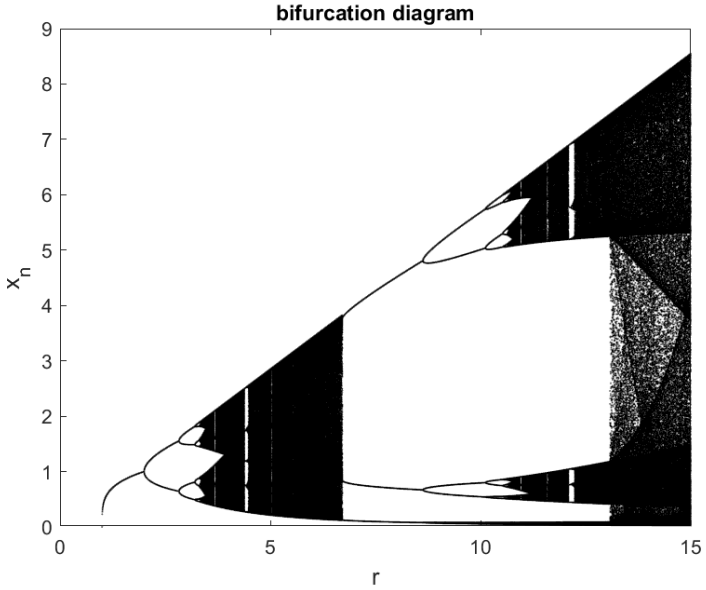


Figure 6:

From the figure, we can say that both (b) and (c) have fixed stable point at $r > 1$. But in case of (c) the logistic map oscillates or so to say, produces period $2^k$ flows within the range

$1 \leq r \leq 15$

## Answer to Question no. 7

If we express all physical quantities in terms of $m_e$, $\hbar$, and $eV$, then we are using *natural units.* using the basic conversion factors between natural and *MKS system*, we get-

$$1\ m_e = 9.10938356\ *\ 10^{-31}\ \text{kg}$$
$$1\ \hbar = 1.05457266\ *\ 10^{-34}\ \text{kg}m^2s^{-1}$$
$$1\ eV = 1.60217733\ *\ 10^{-19}\ \text{kg}m^2s^{-2}$$

now,

$$\frac{1\hbar}{1eV} = \frac{1.05457266 * 10^{-34}kgm^2s^{-1}}{1.60217733 * 10^{-19}kgm^2s^{-2}}$$
$$=>\ \mathbf{1}\ s\ \mathbf{=\ 1.519266894\ *}\ 10^{15}\hbar eV^{-1}\ \textit{(Ans.)}$$

Again,

$$\frac{1\hbar}{1m_e} = \frac{1.05457266 * 10^{-34}kgm^2s^{-1}}{9.1038356 * 10^{-31}kg}$$
$$=>\ 1\ s\ *\ 1\ \hbar m_e^{-1} = 1.157677304\ *\ 10^{-4}m^2$$
$$=>\ 1.519266894\ *\ 10^{15}\hbar eV^{-1}\ *\ 1\ \hbar m_e^{-1} = 1.157677304\ *\ 10^{-4}m^2$$
$$=>\ \mathbf{1}\ m\ \mathbf{=\ 3.622624147\ *}\ 10^{9}\hbar(m_eeV)^{-1/2}\ \textit{(Ans.)}$$