South China University of Technology
Experiment Report

Experiment Title: Programming Using Matlab
Name: 黄豪 Student ID: 201430540078
Class: 计创
Advisor: Yuhui Quan

Introduction

[The purposes]

The purposes:
Learn how to program using Matlab and do some basic image operations in Matlab
Experiment requirements:
Get familiar with the Matlab programming environment and try to do some basic matrix operations with it

[The principles]

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.
It's quite convenient for us to do the image processing works using Matlab.

[Experiment environment]

Ubuntu 16.04 x64, Matlab 9.1

The Procedure

[Procedure outline]
There are four sub experiments in this time of lab.
Exercise 1: In this experiment, we were asked to express the given function in the form of graph with Matlab. We were even required to implement the algorithm of bisection method to resolve the solution of the unknown x in the equation, without the help of Matlab internal library.
Exercise 2: In this experiment, we were given a polynomial function and we would want to resolve the derivative of it. But there was one more step until we could finish this one. Given a polynomial equation, I needed to get each unknown coefficient, where linear algebra knowledge was required.
Exercise 3: This experiment was all about harmonic series. We needed to try two methods to calculate the harmonic series result, with and without loops. What's more, we would want to plot out the figure between a given interval.

Exercise 4: The last experiment requested us to work out the maximum distance among all points randomly generated, with or without the help of looping. Also, we needed to compare the running time of both method given various data sets.

[Experiment procedure]
Experiment 1 Problem:

- 1.1 Implement a matlab function to calculate:

$$f(x) = 21x^9 + x^8 + 8x^7 + x^6 + 3x^5 + 25x^4 + 21x^3 + \frac{1}{6}x^2 + \frac{1}{2}x + 1$$

- 1.2 Write a script to draw the curve of $f(x)\ in\ [-100,100]$ using plot command

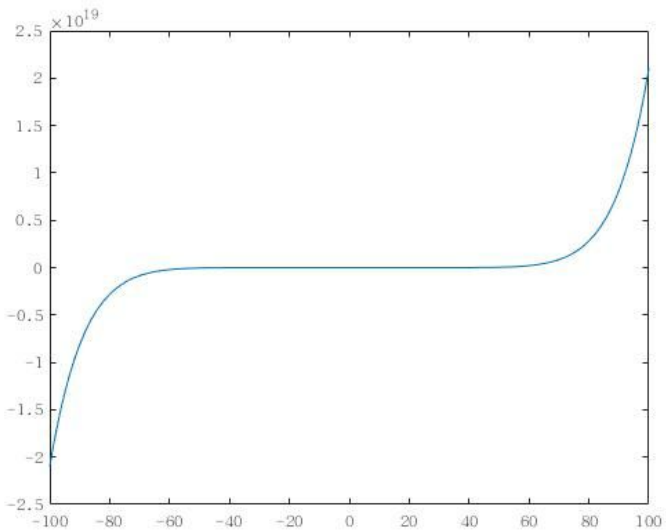
- 1.3 implement a matlab function solve $f(x) = k$ using the bisection method.

My Solution to Experiment 1:
To plot out the graph versus given function, I wrote the following code:
lab0_1_plot.m

```
clear;clc;
x = -100:0.5:100;
f = 21.*x.^9 + x.^8 + 8.*x.^7 + x.^6 + 3.*x.^5 + 25.*x.^4
+ 21.*x.^3 + (1/6).*x.^2 + 0.5.*x + 1;
plot(x,f);
```

There is not so much thing to explain in it. But I want to point out that I was using the jump degree of 0.5.

Here is the figure plotted:



Now in the following is my code to work out the solution of *f(x)=k* with k given:

lab0_1_solve.m:

```
clear; clc;
%params: they are:
%the left bound of solution
%the right bound of solution
%calculation generation number (To invoke the bisection
function, the
%generation number should not exceed a specified number,
otherwise the process
%will be stopped)
%The maximum error that can be made
%the k value of the to-solve-equation f(x)=k
bisection( -100, 100, 0, 0.0001, 3);
```

bisection.m:

```
function bisection( left, right, generation,
errorAllowed, k )
%solve a function using bisection method
%   find the middle point of left and right bound
    x = (left+right)/2;
    %if the middle point is what we are looking for
    if function_to_solve(k, x)==0
        output = x;
        disp(num2str(output))
        return;
    %find the next middle point
    elseif function_to_solve(k,
left)*function_to_solve(k, x)<0
        right = x;
    elseif function_to_solve(k, x)*function_to_solve(k,
right)<0
        left = x;
    end
    %there is no need to continue with the dividing
process when the possible error
    %becomes so slight
    if(abs(left-right)<=errorAllowed)
        output = left;
        disp(num2str(output))
        return;
    end
    %I defined the max generation to be 500
    if(generation>500)
        disp('Too many generations, quit!');
```

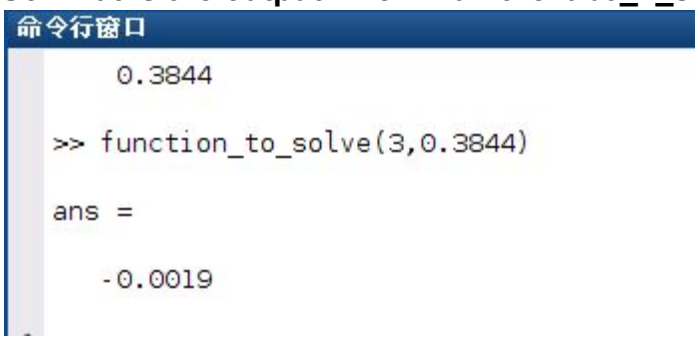
```
        return;
    end
    generation = generation+1;
    bisection(left, right, generation, errorAllowed, k);
end
```

The function above needs to invoke this script:

function_to_solve.m:

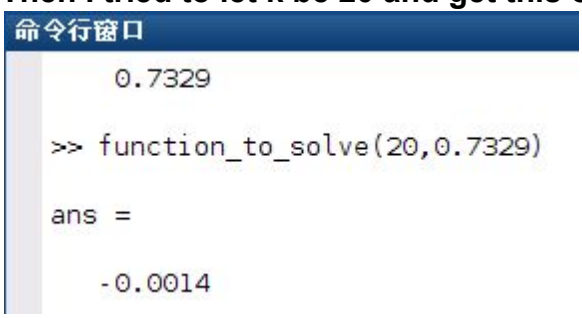
```
function output = function_to_solve(k, x)
f = 21.*x.^9 + x.^8 + 8.*x.^7 + x.^6 + 3.*x.^5 + 25.*x.^4
+ 21.*x.^3 + (1/6).*x.^2 + 0.5.*x + 1-k;
output = f;
end
```

So what is the output when I run the lab0_1_solve.m script?



When the k value in f(x)=k is 3, from my solution I get 0.3844 as the resulting x value. To verify its correctness, I substituted the 0.3844 into the original function and get the -0.0019. The nicest one should be 0 but this one can also be accepted.

Then I tried to let k be 20 and get this output as below:



Hmm.. Not so accurate. But what about modified the ‘Error Allowed’ number, which is the maximum gap between the left and right bound of the expected solution? That is, for example, make it be 0.00001?

After that, I get the following result:

lab0_1_solve.m | bisection.m | function_to_solve.m

```
1 -    clear; clc;
2      %params: they are:
3      %the left bound of solution
4      %the right bound of solution
5      %calculation generation number (To invoke the
6      %generation number should not exceed a specifi
7      %will be stopped)
8      %The maximum error that can be made
9      %the k value of the to-solve-equation f(x)=k
10 -   bisection( -100, 100, 0, 0.00001, 20);
```

命令行窗口

```
0.73291
>> function_to_solve(20,0.73291)

ans =

  -3.4991e-04
```

Experiment 2 Problem:

- 2.1 Given a polynomial function $f(x) = \sum_{n=1}^{N} a_n x^n$.
  Write a function to calculate the derivative function $f'(x)$
  - (HINT1: the input should be a sequence of factors $[a_0, a_1 \ldots, a_n]$)
  - (HINT2: Symbolic Math Toolbox™ provides functions for solving and manipulating symbolic math expressions, BUT you are required NOT TO USE any build-in functions in this exercise)

An example of input and output

```
>> ex2_2_derivative(1, [1 -2 1])

ans =

     0
```

- 2.2 Consider function $f(x) = ax^4 + bx^3 + cx^2 + dx + e$.
  $y = f(x)$ pass through ***points*** $(-2,53), (-1,2), (0,1), (1,20), (2,125)$.
  Write a matlab script to calculate the factors a, b, c, d, e
  (Hint: use inv() to calculate the inverse of a matrix)

My Solution to Experiment 2:
When it comes to the derivative of f(x) function:
lab0_2_1.m:

```
clear; clc;
%the first param is the  a_n series
%the second param is the x value in f'(x)
```

```
derivative([1,-2,1], 2);
```

derivative.m

```
function [ answer ] = derivative( params, target )
%the first param is the  a_n series
%the second param is the x value in f'(x)
   seriesNum = length(params);
   answer = 0;
   for n = 1:seriesNum
       answer = answer + n*params(n)*target^(n-1);
   end
   disp(answer);
end
```

When it comes to the calculation of the factors of the equation:

lab0_2_2.m:

```
x1 = -2;
y1 = 53;
x2 = -1;
y2 = 2;
x3 = 0;
y3 = 1;
x4 = 1;
y4 = 20;
x5 = 2;
y5 = 125;

A = [
    x1^4, x1^3, x1^2, x1, 1;
    x2^4, x2^3, x2^2, x2, 1;
    x3^4, x3^3, x3^2, x3, 1;
    x4^4, x4^3, x4^2, x4, 1;
    x5^4, x5^3, x5^2, x5, 1;
    ];
C = [y1;y2;y3;y4;y5];
answer = inv(A) * C;
disp(num2str(answer));
```

Experiment 3 Problem:

- 3.1 Calculate the value of harmonic series:

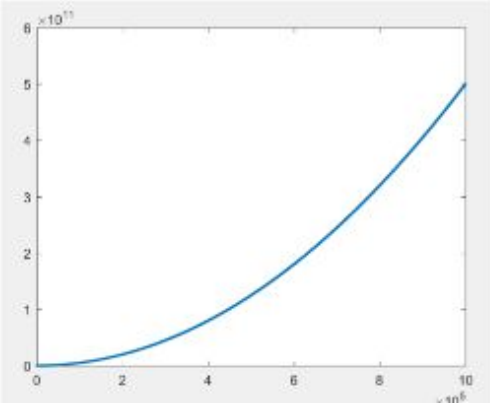
$$a_n = \sum_{i=1}^{n} \frac{1}{i}$$

You are require to use LOOP in here!!

  - Write a matlab function to calculate the harmonic series

- 3.2 Write a script to draw the points $(i, a_i), 1 \leq i \leq 10^6$ on a figure

Don't connect the points (a_n has no definition when n isn't a positive integer)



- 3.3 Calculate the value of harmonic series:

$$a_n = \sum_{i=1}^{n} \frac{1}{i}$$

  - Almost the same as ex3.1, except that you're required to write a function WITHOUT LOOP.
    (use cumsum() to calculate cumulative sum)

- 3.4 Compare the efficiencies of the functions you wrote in ex3.1 and ex3.3. Write down their elapsed time.
  (HINT: use TIC, TOC the get elapsed time)

	Loop(ex3.1)	Vectorization(ex3.3)
N=1000		
N=10000		
N=100000		
N=1000000		

Elapsed time (s)

My Solution to Experiment 3:
The implementation of the harmonic sum is as follows: (with loops, I also used tic and toc to record the running time of the function)
harmonicsum.m:

```
function output = harmonicsum( n )
tic;
output = 0;
for divider = 1:n
```

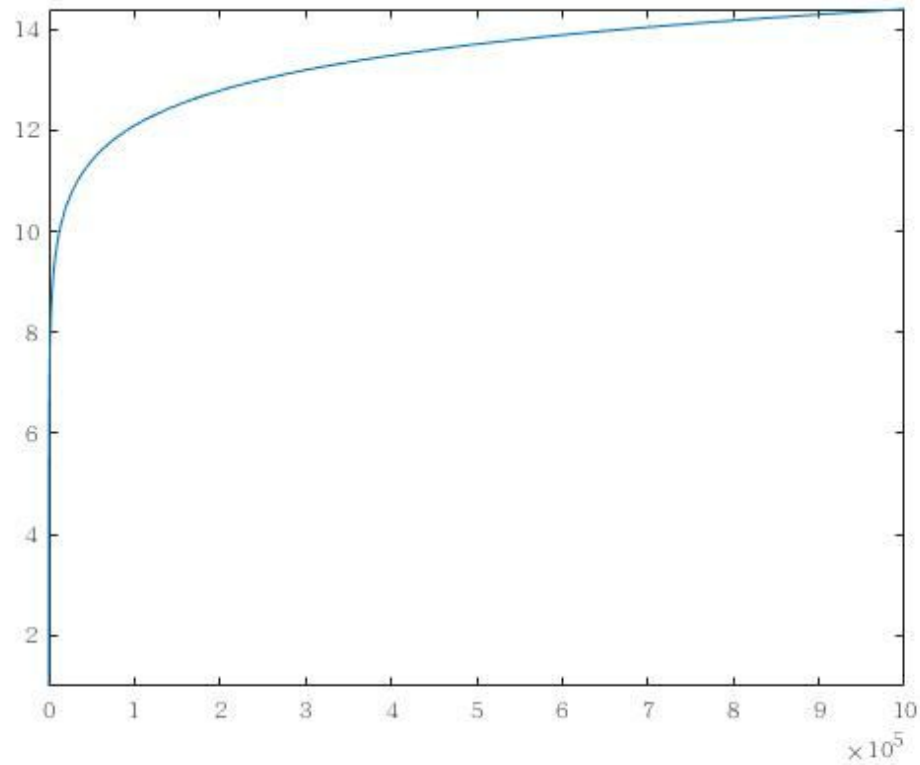
```
    output = output + 1 / divider;
end
t_loop = toc;
disp('harmonic sum with loop: ');
disp(t_loop);
end
```

Plotting the graph:

lab0_3_1.m:

```
clear;clc;
fplot(@(x) harmonicsum(x), [1 1000000]);
```

The output graph is as follows:



The non-loop implementation of harmonic series:

harmonicsum_cs.m:

```
function output = harmonicsum_cs( n )
tic;
nums = 1:n;
nums = 1./nums;
output = cumsum(nums);
output = output(n);
t_nonloop = toc;
disp('harmonic sum without loop');
disp(t_nonloop);
end
```

The result of the comparison between the loop method and non-loop method:

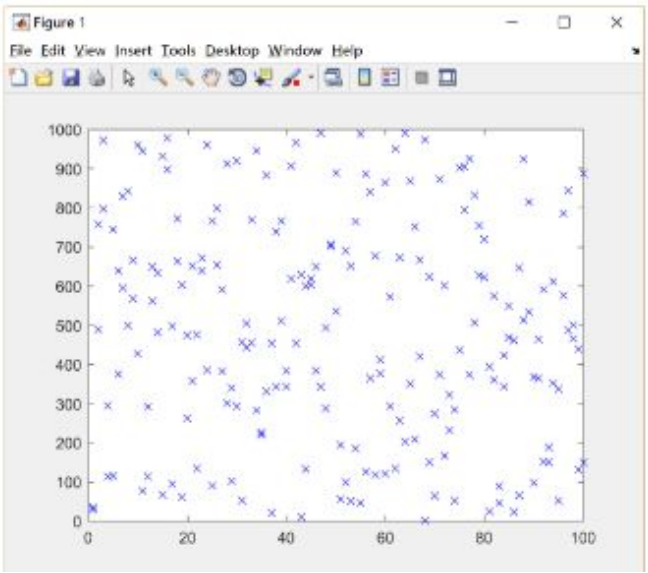
	Loop(ex3.1)	Vectorization(ex3.3)
N=1000	9.2000e-05	7.0000e-05
N=10000	1.0600e-04	5.0000e-05
N=100000	6.2500e-04	2.8800e-04
N=1000000	0.0141	0.0033

Elapsed time (s)

Looping takes time!

Experiment 4 Problem:

- 4.1 Use rand(N,2) to get N points, write a script to draw these points, each point should be represented as a blue cross



- 4.2 Given N points, write a function to find out maximum distance between all possible pairs of points. (Require to USE LOOP)
- 4.3 Almost the same as ex4.2, except that you're required to write a function WITHOUT LOOP.
- 4.4 Compare the efficiencies of the functions you wrote in ex4.2 and ex4.3. Write down their elapsed time.

	Loop(ex4.2)	Vectorization(ex4.3)
N=100		
N=500		
N=1000		
N=5000		

Elapsed time (s)

Drawing random points:

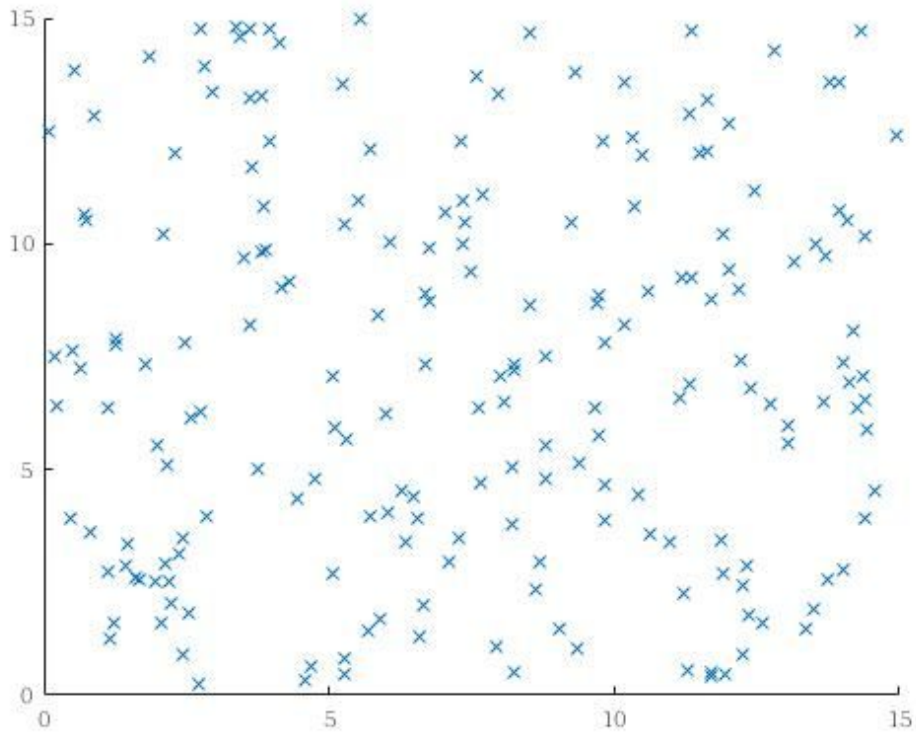
draw_points.m:

```
function draw_points( pointnum )
randnums = rand(pointnum, 2) * 15;
scatter(randnums(:,1), randnums(:,2), 'x');
end
```

lab0_4_1.m:

```
draw_points(200);
```

I wanted to draw out 200 random points. The result is shown below:



Finding maximum distance using loops:

lab0_4_2.m:

```
pointnum = 200;
randnums = rand(pointnum, 2) * 15;
scatter(randnums(:,1), randnums(:,2), 'x');
num = size(randnums, 1);
max_distance = 0;
for k = 1 : num
    for ki = k+1 : num
        this_distance = get_distance_sqr(randnums(k,:),
randnums(ki,:));
        if  this_distance > max_distance
            max_distance = this_distance;
        end
    end
end
max_distance = sqrt(max_distance);
disp('max distance using loop: ');
disp(max_distance);
```

Finding maximum distance without using loops:

lab0_4_3.m (part of it):

```
pointnum = 200;
randnums = rand(pointnum, 2)*100;
%randnumsx = randnums(:, 1);
%randnumsy = randnums(:, 2);
```

```
%scatter(randnumsx, randnumsy, 'x');
%plot(randnumsx(k), randnumsy(k), '-.*c', randnumsx,
randnumsy, 'x');
D = squareform(pdist(randnums,'euclidean'));
max_distence_noloop = max(max(D));
disp('without loop: ');
disp(max_distence_noloop);
```

Run the script lab0_4_3.m and the console window will show the result of the figured out maximum distance in both cases and the elapsed time.

Example running is like this:

命令行窗口

```
>> lab0_4_3
without loop the max distance:
  130.1222

time passing without loop:
    0.0094

max distance using loop the max distance:
  130.1222

time passing using loop:
    0.0286
```

Now I am going to perform the comparison between the looping method and the non-looping method in running time for different scale of input data:

	Loop(ex4.2)	Vectorization(ex4.3)
N=100	0.0101	9.6300e-04
N=500	0.2082	0.0037
N=1000	0.7050	0.0141
N=5000	16.6743	0.3511

Elapsed time (s)

Looping takes time!

[Experiment conclusion]

Matlab is really a good tool for us to perform matrix operations and graph plotting! But there is something we need to be concerned about, for example, sometimes there is more than one solution to a question. Often the method that requires looping will take greater time than the loop-free method, thus we may want to avoid using loops to get the result.

Summary

In lab 0, we used Matlab to solve some practical problems. Not only did we try to be a problem solver using Matlab, but we also did some comparisons about efficiency on Matlab. For-loos often are time-consuming so vectorization is often a better way when available. But why is that? This is something worth discovering in the future and I am sure I will learn more about it in class!

Comment of Advisor and Grade

Comment：

Grade： Sign of Advisor：

Date：