Solution of Week 6 Lab (Prepared by Yuan Yin)

December 22, 2019

1 Exercise 1: Newton's Method:

1.

By setting $f(x) = x^2 - a$ and plugging into $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, we get

$$x_{n+1} = x_n - \frac{{x_n}^2 - a}{2x_n} = \frac{2{x_n}^2 - {x_n}^2 + a}{2x_n} = \frac{{x_n}^2}{2x_n} = \frac{x_n}{2} + \frac{a}{2x_n}.$$

This gives the algorithm you coded for Tut2 Exercise1 (e).

2.

Try to understand and compare Newton's method and Bisection method. Note that Newton's method uses mixed tolerance as its stopping criterion.

3. AND 4.

[35]:
```
%%file driver_partCD.m

function driver_partCD

fprintf('Our function is cos(x) - x. Try initial guess: 1, 3, 6 respectively.
 ↪\n\n');

f1 = @(x) cos(x) - x;
f1_dash = @(x) -sin(x) - 1;
x0_1  = 1;
x0_11 = 3;
x0_111 = 6;

[sol_1, it_hist_1, ierr_1] = Newton(x0_1,f1,f1_dash);
fprintf('When x0 = 1, the solution is: %8.6f\nit_hist:\n', sol_1);
disp(it_hist_1');
fprintf('\n\n');

% figure(1);
% vizNewton(f1, f1_dash, x0_1, 50, sol_1 - 1, sol_1 + 1);
```

```
[sol_11, it_hist_11, ierr_11] = Newton(x0_11,f1,f1_dash);
fprintf('When x0 = 3, the solution is: %8.6f\nit_hist:\n', sol_11);
disp(it_hist_11');
fprintf('\n\n');

% figure(2);
% vizNewton(f1, f1_dash, x0_11, 50, sol_11 - 2, sol_11 + 2.5);

[sol_111, it_hist_111, ierr_111] = Newton(x0_111,f1,f1_dash);
fprintf('When x0 = 6, the solution is: %8.6f\nit_hist:\n', sol_111);
disp(it_hist_111');
fprintf('-------------------------------------------------------------------------\n\n');
 ↪

% figure(3);
% vizNewton(f1, f1_dash, x0_111, 50, sol_111 - 2, sol_111 + 3);



fprintf('\n\nOur function is log(x) - exp(-x). Try initial guess: 2\n\n');

f2 = @(x) log(x) - exp(-x);
f2_dash = @(x) 1 / x + exp(-x);
x0_2 = 2;

[sol_2, it_hist_2, ierr_2] = Newton(x0_2,f2,f2_dash);
fprintf('The solution is: %8.6f\nit_hist:\n', sol_2);
disp(it_hist_2');
fprintf('-------------------------------------------------------------------------\n\n');
 ↪

% figure(4);
% vizNewton(f2, f2_dash, x0_2, 50, sol_2 - 1, sol_2 + 1);



fprintf('\n\nOur function is x.^3 - 7x^2 + 14x - 8.\n\n');
f3 = @(x) x.^3 - 7 * x.^2 + 14 * x - 8;
f3_dash = @(x) 3 * x.^2 - 14 * x + 14;
i = 5;

for x0_3 = 1.1 : 0.1 : 1.9
    [sol_3, it_hist_3, ierr_3] = Newton(x0_3,f3,f3_dash);
    fprintf('When x0 = %3.1f, the solution is: %8.6f\nit_hist:\n', x0_3, sol_3);
    disp(it_hist_3');
    fprintf('\n\n');
```

```
    % figure(i);
    % vizNewton(f3, f3_dash ,x0_3, 50, sol_3 - 2, sol_3 + 2);

    i = i + 1;
end


end
```

```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
6/driver_partCD.m'.
```

[36]:
```
driver_partCD
```

```
Our function is cos(x) - x. Try initial guess: 1, 3, 6 respectively.

When x0 = 1, the solution is: 0.739085
it_hist:
    -4.596976941318602e-01
    -1.892347882211744e-02
    -4.645589899077152e-05
    -2.847205804457076e-10
                         0
                         0


When x0 = 3, the solution is: 0.739085
it_hist:
    -3.989992496600445e+00
     1.375785636177072e+00
    -2.662365851383397e+00
     8.179794111259786e-02
    -9.505036962773605e-04
    -1.190953643481762e-07
    -1.887379141862766e-15
                         0
                         0



When x0 = 6, the solution is: 0.739085
it_hist:
    -5.039829713349634e+00
     1.539355228724625e+00
    -9.143975876784287e+00
    -4.365574314937907e+00
     1.680950178528314e+00
```

```
    -2.707175006121862e+00
     9.885637176683726e-02
    -1.410272249024236e-03
    -2.620688250853931e-07
    -8.992806499463768e-15
                         0
                         0

--------------------------------------------------------------------------



Our function is log(x) - exp(-x). Try initial guess: 2.
The solution is: 1.309800
it_hist:
     5.578118973233326e-01
    -2.104911740247836e-01
    -1.539033840352810e-02
    -9.354555465146408e-05
    -3.494047977483818 7e-09
    -5.551115123125783e-17
    -5.551115123125783e-17

--------------------------------------------------------------------------


Our function is x.^3 - 7x^2 + 14x - 8.

When x0 = 1.1, the solution is: 1.000000
it_hist:
     2.610000000000028e-01
    -5.228751961190614e-02
    -1.120722552352404e-03
    -5.572245616036753e-07
    -1.385558334732195e-13
                         0
                         0



When x0 = 1.2, the solution is: 1.000000
it_hist:
     4.480000000000040e-01
    -3.209610730427297e-01
    -3.019968297417464e-02
    -3.865388461417041e-04
    -6.636412841487527e-08
```

```
    -2.664535259100376e-15
                         0
                         0



When x0 = 1.3, the solution is: 1.000000
it_hist:
     5.669999999999966e-01
    -1.593523637705494e+00
    -3.184679968646629e-01
    -2.981101275519293e-02
    -3.768783244595397e-04
    -6.308936839616308e-08
    -2.664535259100376e-15
                         0
                         0



When x0 = 1.4, the solution is: 1.000000
it_hist:
     6.239999999999988e-01
    -2.497455393586007e+00
    -6.997330150744817e+00
    -1.813971394432778e+00
    -3.756863447969554e-01
    -3.911142955156027e-02
    -6.395761116380072e-04
    -1.816162376044872e-07
    -1.421085471520200e-14
                         0
                         0



When x0 = 1.5, the solution is: 4.000000
it_hist:
     6.250000000000000e-01
                         0
                         0



When x0 = 1.6, the solution is: 2.000000
it_hist:
     5.760000000000005e-01
    -8.959999999999937e-01
```

```
    -2.777383246546350e-02
    -1.825441201610545e-04
    -8.327546652253659e-09
                         0
                         0



When x0 = 1.7, the solution is: 2.000000
it_hist:
     4.830000000000023e-01
    -2.690371495678505e-01
    -1.099967182808825e-02
    -2.959309844641211e-05
    -2.189288750287233e-10
                         0
                         0
```



```
When x0 = 1.8, the solution is: 2.000000
it_hist:
     3.519999999999968e-01
    -7.705320514085656e-02
    -1.278161472011874e-03
    -4.073820392136440e-07
    -4.263256414560601e-14
                         0
                         0
```



```
When x0 = 1.9, the solution is: 2.000000
it_hist:
     1.890000000000001e-01
    -1.360497421839568e-02
    -4.503883320694513e-05
    -5.070788233751955e-10
                         0
                         0
```

The aim of this exercise is to investigate how the initial guess influences the process of finding roots. That is, a GOOD initial guess will make the root finding process more efficient and vice versa. However, GOOD initial guess does not simply mean 'x_0 being close enough to the roots'. You have to take the overall shape/characteristics of the function into account.

5. Explain the M-file(Challenging):

This algorithm aims to find the square root of some number, A.

- If A is zero, then we know that $\sqrt{A} = 0$ (easy!).
- However, if A is a non-zero real number, rather than finding $\sqrt{A}$ directly, this algorithm tries to generate some other number, m, first. We can apply Newton's Method to get $\sqrt{m}$. After that, by using the relationship between $\sqrt{m}$ and $\sqrt{A}$, we can finally get a numerical result for $\sqrt{A}$. Now, let's investigate the codes more closely———

1). After going through and exiting the first two while loops, we have: $\sqrt{m} = 2^{-TwoPower} \times \sqrt{A}$ and $0.25 \leq m < 1$.

2). We apply Newton's Method to find $x = \sqrt{m}$:

Our initial guess is $x = \frac{1+2m}{3}$. This is a good guess since $|m - x^2| = \frac{9m-(1+2m)^2}{9} = \frac{m-(1-2m)^2}{9} \leq 0.0625$ as $0.25 \leq m < 1$, i.e. x is already close to $\sqrt{m}$!

Then, finding $\sqrt{m}$ is equivalent to finding the root of $f(x)$ where $f(x) = x^2 - m$. From Newton's method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{{x_n}^2 - m}{2x_n} = \frac{x_n + \frac{m}{x_n}}{2}.$$

One can see that this is basically what the for loop is doing in the MATLAB file.

Note that we only have 4 iterations in the for loop. This is because: From previous analysis, we have $|m - x^2| \leq 0.0625$. So, we can treat 0.0625 as an upper bound for the initial error, e_0. However, note that $f(x) = x^2 - m$ (with $m \neq 0$) has only simple roots, for which Newton's method is quadratically convergent, i.e. ${e_{n+1}} = C{e_n}^2$. Therefore, one can see that 4 iterations are enough to make

$$e_4 = (((0.0625^2)^2)^2)^2 = 5.42 \times 10^{-20} < eps$$

.

2 Exercise 2: Secant Method:

[37]:
```
%%file Newton_secant_PartA.m

function [sol, it_hist, ierr] = Newton_secant_PartA(x1, x2,f,tol,parms)

format long e
%
% set the debug parameter, 1 turns display on, otherwise off
%
debug = 1;
%
% initialize it_hist, ierr, and set the iteration parameters
%
ierr = 0;
MAXIT = 40;
```

```
if nargin == 5
    MAXIT=parms(1);
end

if nargin <= 3
    tol = [0 eps];
end

rtol = tol(2);
atol = tol(1);

if nargin < 3
    disp('Must specify 2 functions and an interval.');
    ierr = 1;
    return
end

it_hist = [];
itc=0;
xc1 = x1;
xc2 = x2;
fc = f(xc2);
fd = (f(x2) - f(x1)) / (x2 - x1);
it_hist=[it_hist,fc];
step = -fc / fd;

while abs(step) > atol + rtol * abs(xc2) && itc < MAXIT
   itc = itc + 1;
   step = -fc / fd;
   xc1 = xc2;
   xc2 = xc2 + step;
   fc   = f(xc2);
   fd   = (f(xc2) - f(xc1)) / (xc2 - xc1);
   it_hist=[it_hist,fc];

   % if debug == 1
        % disp([itc xc fc])
   % end
end

sol = xc2;

% on failure, set the error flag

if itc >= MAXIT
    ierr = 1;
end
```

```
end
```

```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
6/Newton_secant_PartA.m'.
```

[54]:
```
%%file secant_driver_PartBC.m

function secant_driver_PartBC
format long

%% PART B

fprintf('cos(x) - x with initial conditions 1 and 3:\n\n');
func = @(x) cos(x) - x;
init_cond1 = 1;
init_cond2 = 3;

[sol, it_hist, ierr] = Newton_secant_PartA(init_cond1, init_cond2,func);

fprintf('The computed root is %6.4f\n', sol);
fprintf('Check: cos(%6.4f) - %6.4f = %.8f\n\n', sol, sol, func(sol));
disp('it_hist:');
disp(it_hist);
fprintf('\n\n-----------------------------------------------------------------------------------------
 ↪
%% PART C


format short e;

fprintf('log(x) - exp(-x) with initial conditions 1 and 2:\n\n');

funcc = @(x) log(x) - exp(-x);
funcc_dash = @(x) 1 / x + exp(-x);
init_cond11 = 1;
init_cond22 = 2;

[soll, it_histt, ierrr] = Newton_secant_PartA(init_cond11, init_cond22,funcc);
[sol_b, it_hist_b, ierr_b] = Bisection([init_cond11, init_cond22],funcc);
[sol_n, it_hist_n, ierr_n] = Newton((init_cond11 + init_cond22) /␣
 ↪2,funcc,funcc_dash);

fprintf('The computed root is %6.4f\n', soll);
fprintf('Check: log(%6.4f) - exp(-%6.4f) = %.8f\n\n', soll, soll, funcc(soll));
```

```
fprintf('it_hist (Comparing with Newton"s method and Bisection):\n\n');
disp('For Newtons"s method: ');
disp(it_hist_n');
disp('For Bisection method: ');
disp(it_hist_b');
disp('For Secant method: ');
disp(it_histt');

end
```

```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
6/secant_driver_PartBC.m'.
```

[55]:
```
secant_driver_PartBC
```

```
cos(x) - x with initial conditions 1 and 3:

The computed root is 0.7391
Check: cos(0.7391) - 0.7391 = 0.00000000

it_hist:
  Columns 1 through 3

   -3.989992496600445e+00    -8.112277313536698e-04    -4.182491713222714e-05

  Columns 4 through 6

   -4.474739689896978e-09    -2.475797344914099e-14                         0

  Column 7

                         0



--------------------------------------------------------------------------------
---------



log(x) - exp(-x) with initial conditions 1 and 2:

The computed root is 1.3098
Check: log(1.3098) - exp(-1.3098) = -0.00000000

it_hist (Comparing with Newton"s method and Bisection):
```

```
For Newtons"s method:
     1.823349479597346e-01
    -1.530087688107434e-02
    -9.246706388221781e-05
    -3.413909821503580e-09
    -5.551115123125783e-17
    -5.551115123125783e-17

For Bisection method:
     1.823349479597346e-01
    -6.336124554598033e-02
     6.561413531378812e-02
     2.787366754457898e-03
    -2.985380704920870e-02
    -1.342726255921550e-02
    -5.293741207889502e-03
    -1.246670408426298e-03
     7.719730503288336e-04
    -2.369419201275202e-04
     2.676418729389655e-04
     1.536304757221441e-05
    -1.107830817261846e-04
    -4.770842857099167e-05
    -1.617229338934933e-05
    -4.045236331462476e-07
     7.479286788181216e-06
     3.537387782248658e-06
     1.566433625699304e-06
     5.809553841329418e-07
     8.821597241581713e-08
    -1.581538061623533e-07
    -3.496891076704145e-08
     2.662353237870008e-08
    -4.172688861103779e-09
     1.122542181430930e-08
     3.526366532113911e-09
    -3.231611644949339e-10
     1.601602683809489e-09
     6.392207874128530e-10
     1.580298114589596e-10
    -8.256567651798719e-11
     3.773203971491057e-11
    -2.241679064596269e-11
     7.657596778898323e-12
    -7.379541422380953e-12
     1.389999226830696e-12
    -3.620270749848942e-12
    -1.740663169158552e-12
```

```
    -8.008038676621254e-13
    -3.308464613382966e-13
    -9.592326932761353e-14
     2.153832667772804e-14
    -3.724798247617400e-14
    -7.771561172376096e-15
     6.883382752675971e-15
    -4.996003610813204e-16
     3.164135620181696e-15
     1.387778780781446e-15
     4.440892098500626e-16
    -5.551115123125783e-17
     1.665334536937735e-16

For Secant method:
     5.578118973233326e-01
     8.738450962148026e-02
    -2.538972482740143e-02
     9.060977840135709e-04
     9.106046938571207e-06
    -3.295761330512903e-09
     1.187938636348917e-14
    -5.551115123125783e-17
    -5.551115123125783e-17
```


From the output, we can see that Newton"s method and Secant method only need 6 and 9 iterations respectively to converge. However, for Bisection, we need 50. This is consistent with what we learnt in lectures! If you plot the graph of log(x) - exp(-x), there is a simple root near 1.3098. For simple root, Newton"s method exhibits quadratic convergence while Secant method is superlinear convergent. Bisection method converges only linearly.

3 Exercise 3: Matrices in MATLAB:

[60]:
```
%%file EX3.m

function EX3


fprintf('PART A: \n\n');

vector1 = rand(4, 1);
vector2 = rand(4, 1);

dot_product = vector1' * vector2

dot_product_1 = vector1' * vector1;
```

```
eucli_length_1 = sqrt(dot_product_1)

% Also, note that the Euclidean length of a vector is its two norm,
% so we can use the following MATLAB command:
norm(vector1)
fprintf('\n-------------------------------------------------\n\n\n\n');


fprintf('PART B: \n\n');


% Random symmetric matrix:
% Note that When the matrix A is square, (A + A')/2 is symmetric.
A = rand(4, 4);
rand_symm_matrix = (A + A') / 2

% Random skew-symmetric matrix:
% Note that when the matrix B is square, (-B + B') is skew_symmetric:
B = rand(4, 4);
rand_skew_symm_matrix = B - B'
fprintf('\n-------------------------------------------------\n\n\n\n');


fprintf('PART C: \n\n');

magic_size_2 = magic(2)
check_magic_matrix(magic_size_2); % This is a subfunction calculating the row &
 ↪column & diag sums.

magic_size_3 = magic(3)
check_magic_matrix(magic_size_3);

magic_size_4 = magic(4)
check_magic_matrix(magic_size_4);
fprintf('\n-------------------------------------------------\n\n\n\n');


fprintf('PART D: See the below diagram\n\n');

[X1, X2] = meshgrid(0 : 0.05 : 1, 0 : 0.05 : 1);
Z = min(1 - abs(X1), 1 - abs(X2));
surf(X1, X2, Z)
fprintf('\n-------------------------------------------------\n\n\n\n');


fprintf('PART E: \n\n');
```

```
fprintf('%d\n', eye(4, 2))
fprintf('\n\n');
fprintf('%d %6.2f\n', eye(4, 2))


end

function check_magic_matrix(matrix)

column_sum = sum(matrix)
row_sum = sum(matrix')
dig_sum = [trace(matrix), trace(rot90(matrix))]

fprintf('**********************************************\n\n');

end
```

```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
6/EX3.m'.
```


[61]:
```
EX3
```

```
PART A:

dot_product =

     1.41752789280135 6e+00


eucli_length_1 =

     1.207754412016482e+00


ans =

     1.207754412016482e+00


----------------------------------------------


PART B:


rand_symm_matrix =
```

```
  Columns 1 through 3

    3.804458469753567e-01     5.493095968670969e-01     3.223389752176282e-01
    5.493095968670969e-01     7.791672301020112e-01     7.017006626436944e-01
    3.223389752176282e-01     7.017006626436944e-01     1.190206950124140e-02
    1.080662134299250e-01     4.620953745788186e-01     3.241688432218432e-01

  Column 4

    1.080662134299250e-01
    4.620953745788186e-01
    3.241688432218432e-01
    5.285331355062127e-01


rand_skew_symm_matrix =

  Columns 1 through 3

                         0    -8.723256173837124e-02     3.399431582382546e-02
    8.723256173837124e-02                         0    -4.627957629991718e-01
   -3.399431582382546e-02     4.627957629991718e-01                         0
   -1.573666321672252e-01     9.123133386299529e-01    -7.476414487363637e-01

  Column 4

    1.573666321672252e-01
   -9.123133386299529e-01
    7.476414487363637e-01
                         0


------------------------------------------------



PART C:


magic_size_2 =

     1     3
     4     2


column_sum =
```

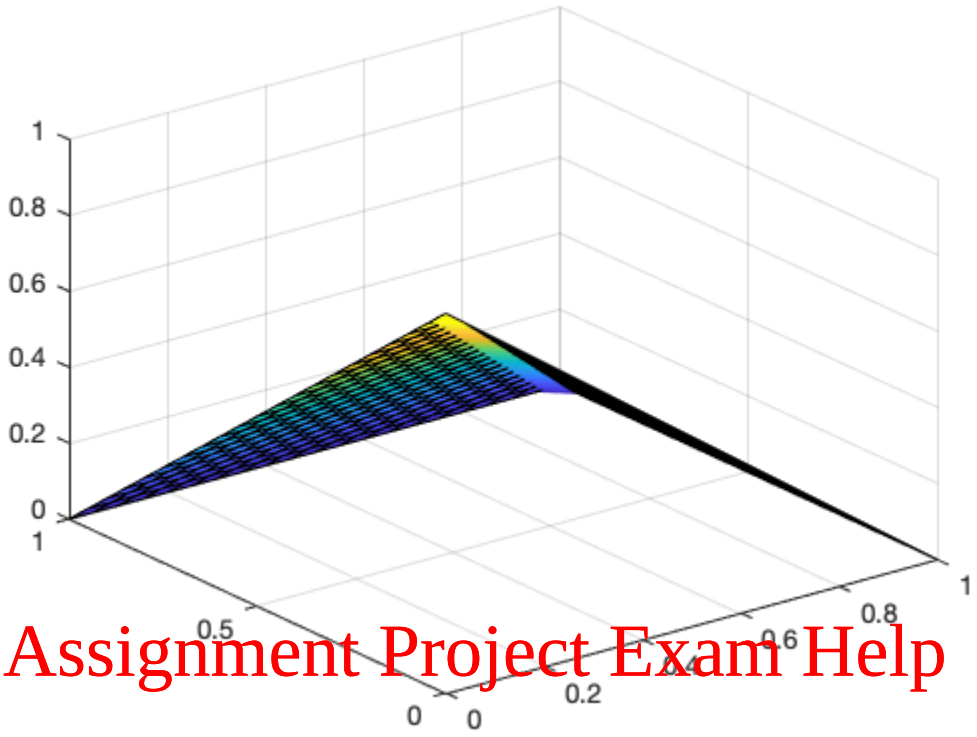
```
     5     5


row_sum =

     4     6


dig_sum =

     3     7

************************************************


magic_size_3 =

     8     1     6
     3     5     7
     4     9     2


column_sum =

    15    15    15


row_sum =

    15    15    15


dig_sum =

    15    15

************************************************


magic_size_4 =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1


column_sum =
```

```
    34    34    34    34

row_sum =

    34    34    34    34

dig_sum =

    34    34

**********************************************

-----------------------------------------------
```


PART D: See the below diagram

PART E:

```
1
0
0
0
0
1
0
0


1   0.00
0   0.00
0   1.00
0   0.00
```



For PART C: We can see that if the matrix is not of size 2-by-2, the matrix we generated satisfies the restrictions presented in the tutorial sheet. If you use the 'help' command, you can actually see that magic(N) produces valid magic squares for all N $>$ 0 except N $=$ 2.