Solution to Assignment 2: Simulation & errors

1 Root-finding

a. The modified code of fzerotx is as below.

```
function [b,iteration] = fzerotx(F,ab,varargin)
%FZEROTX  Textbook version of FZERO.
%   x = fzerotx(F,[a,b]) tries to find a zero of F(x)
between a and b.
%   F(a) and F(b) must have opposite signs.  fzerotx
returns one
%   end point of a small subinterval of [a,b] where F
changes sign.
%   Arguments beyond the first two,
fzerotx(F,[a,b],p1,p2,...),
%   are passed on, F(x,p1,p2,..).
%
%   Examples:
%      fzerotx(@sin,[1,4])
%      F = @(x) sin(x); fzerotx(F,[1,4])

%   Copyright 2014 Cleve Moler
%   Copyright 2014 The MathWorks, Inc.

% Initialize.
a = ab(1);
b = ab(2);
fa = F(a,varargin{:});
fb = F(b,varargin{:});
if sign(fa) == sign(fb)
   error('Function must change sign on the interval')
end
c = a;
fc = fa;
d = b - c;
e = d;
iteration = 0;
% Main loop, exit from middle of the loop
while fb ~= 0
   % The three current points, a, b, and c, satisfy:
   %    f(x) changes sign between a and b.
%   abs(f(b)) <= abs(f(a)).
%   c = previous b, so c might = a.
% The next point is chosen from
%   Bisection point, (a+b)/2.
%   Secant point determined by b and c.
%   Inverse quadratic interpolation point determined
%   by a, b, and c if they are distinct.

iteration = iteration + 1;
if sign(fa) == sign(fb)
   a = c;  fa = fc;
   d = b - c;  e = d;
end
if abs(fa) < abs(fb)
   c = b;    b = a;    a = c;
   fc = fb;  fb = fa;  fa = fc;
end

% Convergence test and possible exit
m = 0.5*(a - b);
tol = 2.0*eps*max(abs(b),1.0);
if (abs(m) <= tol) || (fb == 0.0)
   break
end

% Choose bisection or interpolation
if (abs(e) < tol) || (abs(fc) <= abs(fb))
   % Bisection
   d = m;
   e = m;
else
   % Interpolation
   s = fb/fc;
   if (a == c)
      % Linear interpolation (secant)
      p = 2.0*m*s;
      q = 1.0 - s;
   else
      % Inverse quadratic interpolation
      q = fc/fa;
      r = fb/fa;
      p = s*(2.0*m*q*(q - r) - (b - c)*(r - 1.0));
      q = (q - 1.0)*(r - 1.0)*(s - 1.0);
   end
```

```
      if p > 0
         q = -q;
      else
         p = -p;
      end
      % Is interpolated point acceptable
      if (2.0*p < 3.0*m*q - abs(tol*q)) && (p <
abs(0.5*e*q))
         e = d;
         d = p/q;
      else
         d = m;
         e = m;
      end
   end
   % Next point
   c = b;
   fc = fb;
   if abs(d) > tol
      b = b + d;
   else
      b = b - sign(b-a)*tol;
   end
   fb = F(b,varargin{:});
end
```

b. Using modified fzerotx, we have:

For equation $\sin x = \cos\left(2x^2\right)$, its equal function is $f\left(x\right) = \sin x - \cos\left(2x^2\right)$, it takes 9 iterations to obtain its first positive root as 0.6708, it takes 10 iterations to obtain its second positive root as 2.4204.

For function $f\left(x\right) = 1/\left(x-\pi\right)$, it takes 65 iterations to obtain its root as 3.1416 on interval [0,5], which obviously is a mistake. This indicates that fzerotx fail to guarantee its performance when dealing with functions like $f\left(x\right) = 1/\left(x-\pi\right)$.

For function $f\left(x\right) = 1-\left(1+3x\right)\exp\left(-3x\right)$, on interval [-1,1], fzerotx fail to work for that both $f\left(-1\right)$ and $f\left(1\right)$ are positive, not satisfying the requirement. This indicates that fzerotx has quite strict usage conditions.

c. The stopping criterion is that the function value of right endpoint of interval is equal to 0 or that the half length of interval is below given tolerance.

2 Backward stability vs. accuracy

For test matrices generated by function gallery, using three methods, we obtain their relative forward error and relative residual as below.

a.

Relative forward error

n\cond no	1	10^4	10^8	10^{12}
25	4.03943073125473e-16	1.68074748396999e-13	1.46245477698640e-09	1.78634479210557e-05
50	1.28697645796975e-15	1.91149766108004e-13	1.03240486129617e-09	2.69213319691295e-06

Relative residual

n\cond no	1	10^4	10^8	10^{12}
25	4.00895483180275e-16	1.22069158991018e-16	4.49960294442183e-17	4.20807675992911e-17
50	1.24318666039162e-15	7.77881388096141e-17	5.15529141043001e-17	4.50256440887642e-17

b.

Relative forward error

n\cond no	1	10^4	10^8	10^{12}
25	6.27111209146136e-16	2.30278450280580e-13	1.60249271083880e-09	1.73150745117739e-05
50	1.30371765322426e-15	2.10989364655776e-13	1.03473444095948e-09	5.42017757696663e-06

Relative residual

n\cond no	1	10^4	10^8	10^{12}
25	6.11014433017404e-16	2.26204441714784e-14	7.19841845346890e-11	9.07688936574592e-08
50	1.32515954270764e-15	2.09868697752542e-14	5.84317162775116e-11	4.28888693705745e-07

c.

Relative forward error

n\cond no	1	10^4	10^8	10^{12}
25	5.15524816643046e-16	4.77641305499262e-13	1.38035299006542e-09	8.71908747403233e-06
50	7.60264024139365e-16	2.87413044561116e-13	1.33228465990119e-09	1.33283603075361e-05

Relative residual

n\cond no	1	10^4	10^8	10^{12}
25	5.94385037681062e-16	1.71651201396264e-16	1.14752662786678e-16	6.40522421498100e-17
50	7.37105996719865e-16	1.35327136194111e-16	7.46869267731718e-17	8.06051360820707e-17

From data above, in terms of relative forward error, the three methods have close performance, however, in terms of relative residual, LU factorization and QR factorization have close and better performance, while the residual of method of multiplying inverse matrix increases as condno increases.

For test matrices generated by function gfpp, using three methods, we obtain their relative forward error and relative residual as below.

Relative forward error

n\method	a	b	c
25	1.28815982746777e-10	6.76379352156879e-16	1.63448375857963e-15
50	0.00169198854265177	1.18258220856521e-15	1.00051773871199e-14

Relative residual

n\method	a	b	c
25	2.05106769854316e-11	4.15577917121150e-17	1.90497525160770e-16
50	9.87307515216809e-05	1.25284801135605e-16	5.45827470443243e-16

From data above, LU factorization’s performance deteriorates as n increases while the other two methods’ performances are stable, indicating LU factorization is poor at handling such matrices.

3 Application: Nonlinear systems of equations

a. The code of MATLAB function to implement Newton's method is as below.

```
function [roots, count, resids, history] = ass2Q3(func,
x0, tol)
    if(~exist('tol', 'var'))
        tol = 1e-10;
    end
    count = 0;
    roots = x0;
    [f, J] = func(roots);
    resids = [norm(f, inf)];
    history = [roots];
    while norm(f, inf) >= tol && count < 50
        roots = roots - J^(-1) * f;
        [f, J] = func(roots);
        resids = [resids, norm(f, inf)];
        history = [history, roots];
        count = count + 1;
    end
end
```

b. The driver function code is as below.

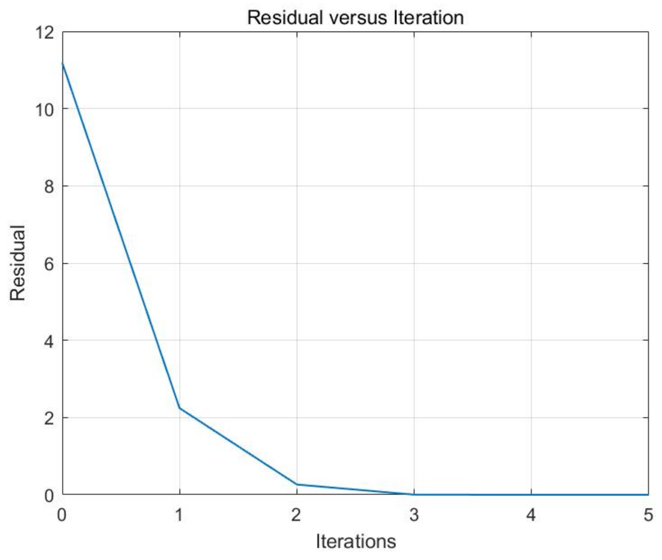
```
[roots, count, resids, history] = ass2Q3(@example1, [1;
2]);
fprintf("After %d iterations, the roots are
[%.7f %.7f]\n", count, roots(1), roots(2));
function [f, J] = example1(x)
    f = [x(1)^2 - x(2) - 1; -x(1) + x(2)^2 - 1];
    J = [2 * x(1), -1; -1, 2 * x(2)];
end
```

Using the code above, we obtain output as "After 5 iterations, the roots are [1.6180340 1.6180340]".

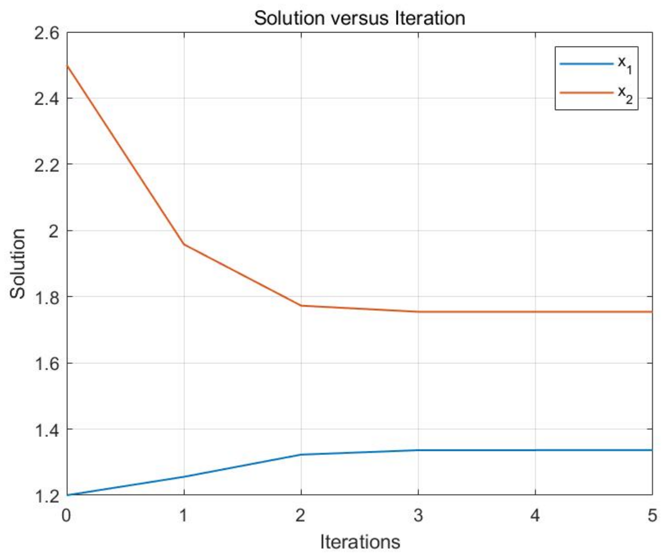
c. For initial guess $x_0 = (1.2, 2.5)$, we obtain its output as:

After 5 iterations, the roots are [1.3363554 1.7542352].

Its convergence history is plotted as below.



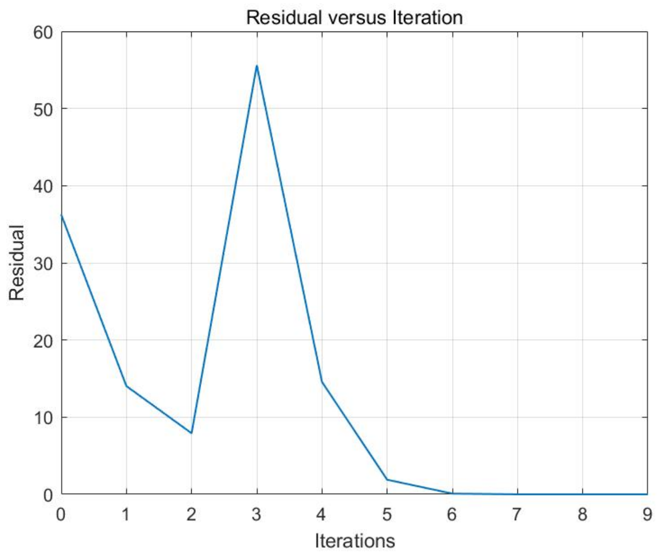
Its solution trajectory is plotted as below.



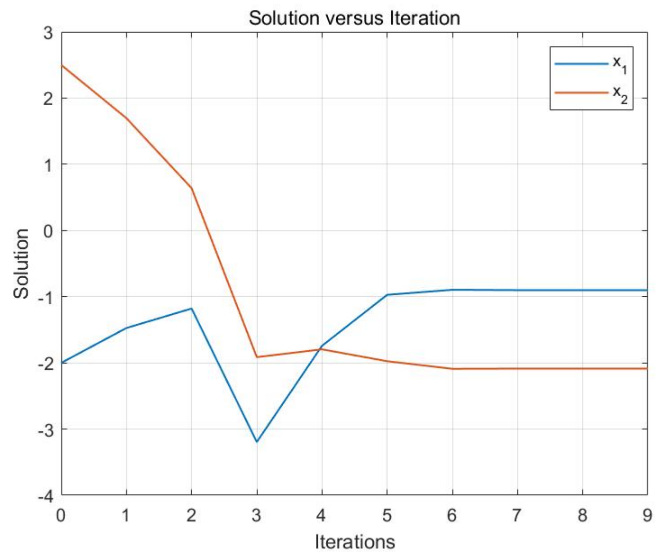
For initial guess $x_0 = (-2, 2.5)$, we obtain its output as:

After 9 iterations, the roots are [-0.9012662 -2.0865876].

Its convergence history is plotted as below.



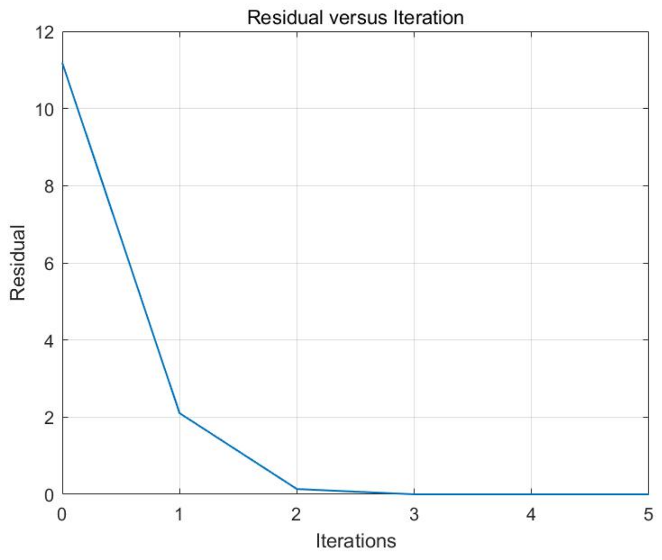
Its solution trajectory is plotted as below.



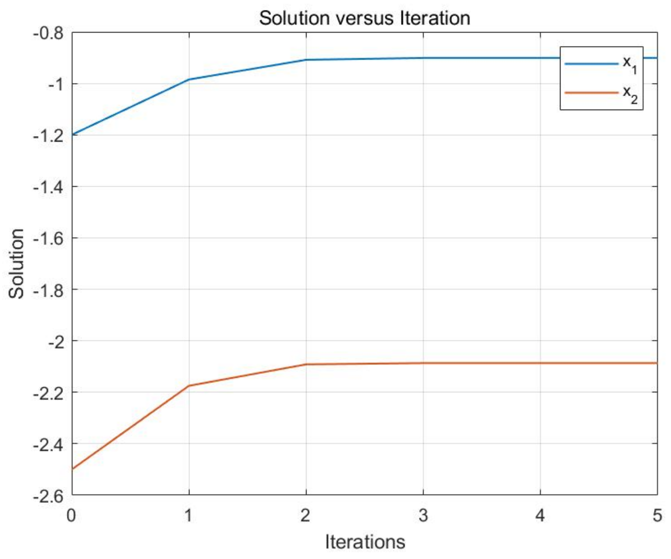
For initial guess $x_0 = (-1.2, -2.5)$, we obtain its output as:

After 5 iterations, the roots are [-0.9012662 -2.0865876].

Its convergence history is plotted as below.



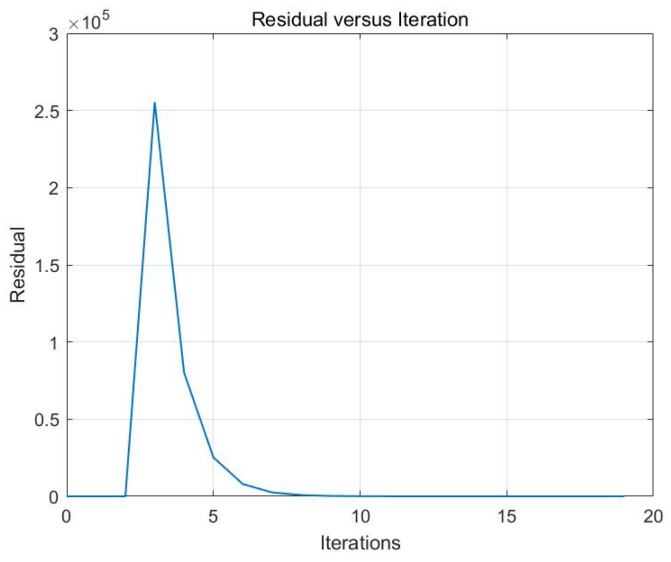
Its solution trajectory is plotted as below.



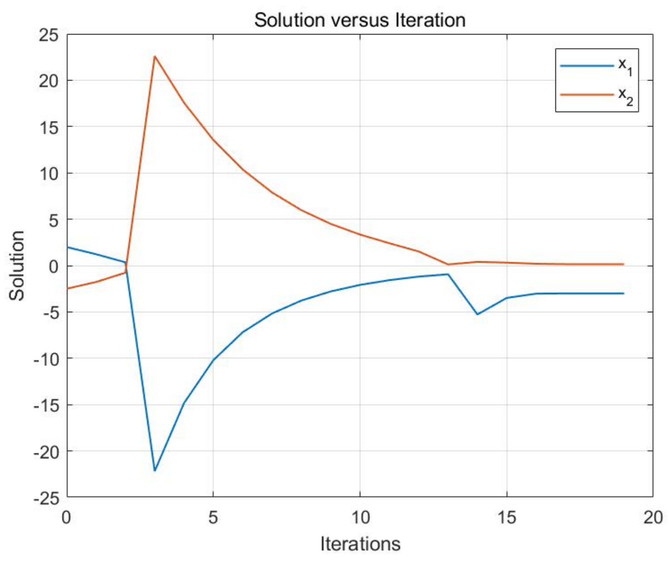
For initial guess $x_0 = (2, -2.5)$, we obtain its output as:

After 19 iterations, the roots are [-3.0016249 0.1481080].

Its convergence history is plotted as below.



Its solution trajectory is plotted as below.



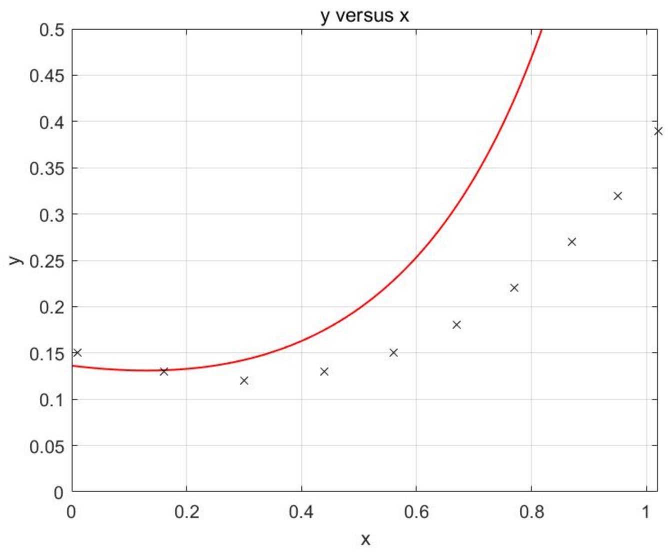
It is quite obvious that all four initial conditions succeed to converge to stable solutions. Also, we notice that the curves of convergence history and solution trajectory of initial guesses $x_0 = (-2, 2.5)$ and $x_0 = (2, -2.5)$ have waves instead of keeping decreasing. Besides, for a system of multiple solutions, different initial guesses may result in different results.

4 Least squares

a. In the least squares sense, we finally obtain

$$\begin{cases} b \approx -2.379302921992261 \\ c \approx 0.002246147555916 \\ d \approx 0.572331305957531 \\ e \approx 3.228180821080603 \\ f \approx -0.439929725572333 \end{cases}$$

With coefficients obtained above, we plot the orbit along with given data points as below.

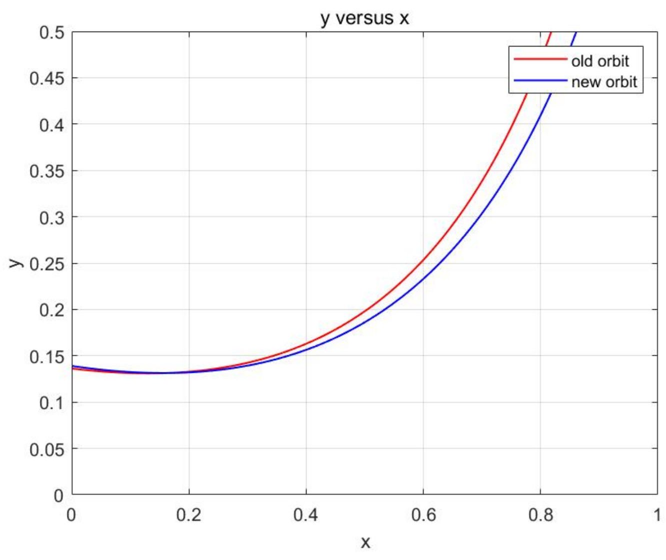


b. After perturbing the data, we obtain

$$\begin{cases} b \approx -2.102176329306665 \\ c \approx -0.145971832940527 \\ d \approx 0.589521120498403 \\ e \approx 3.247881535315491 \\ f \approx -0.448675089284515 \end{cases}$$

By comparing the new coefficients with the old ones, it can be observed that the coefficients are different but `still quite close, which is owning to that the perturbance is quite small.

The new orbit along with old orbit are plotted as below.



For the orbits, they are still quite close to each other.

As for singular values, the original matrix has singular values 3.786036378417616, 0.944922723795015, 0.208912985765180, 0.023043153753913 and 0.005499527069382, the perturbed matrix has singular values 3.783884272680697, 0.944316834477350, 0.206506122392295, 0.022907628046645 and 0.005339258589899. By observing their singular values, we can draw the same conclusion, the small perturbance only result in tiny change to singular values, that is why the coefficients and orbits are very close.