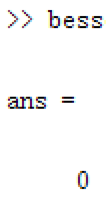
Q1

a. Finished in code
b. 3 questions
    - 1st solution: 0.6708 2nd solution: 2.4204
        i. The algorithm works well for this equation because when we choose the interval, we guarantee that there is a root in it. Therefore in a few iterations, we can quickly find the solution.
    - Solution is pi. But this is not the theoretical right one. There is no solution for this equation in the interval[0,5]. The algorithm is not reliable in this case. Therefore the algorithm just keep shrinking the interval until the interval is smaller than the tolerance, then algorithm stops and return the last x we iterated, evident by 64 iterations program outputted.
    - Not able to find the solution because the initial and end value has the same sign and there is no change of sign in the interval [-1,1], evident by the plot of this function. Therefore the algoirthm is not able to run

c. There are two stopping criterions for fzerotx:
    - When we find the root of the equation
    - When the searching-interval is relatively very small to b, there is no point to keep searching for the root given we have test so many values in [a,b]

d. When we evaluate f(b) at the beginning of program, we use command fb = F(b, varargin{;}). However, in this case, varargin is the extra output we put into the function, which is 0. So what actually happened to fb is that we call fb = besselj(pi,0); and if we set 0 to be the Z value in the Bessel's equation, the solution is always 0. So at the star of program, we have set fb = 0 and b = pi. When fb =0 , the program meets its stopping criteria, so it just output that solution is pi.
   This is really just a bug in calling the function.



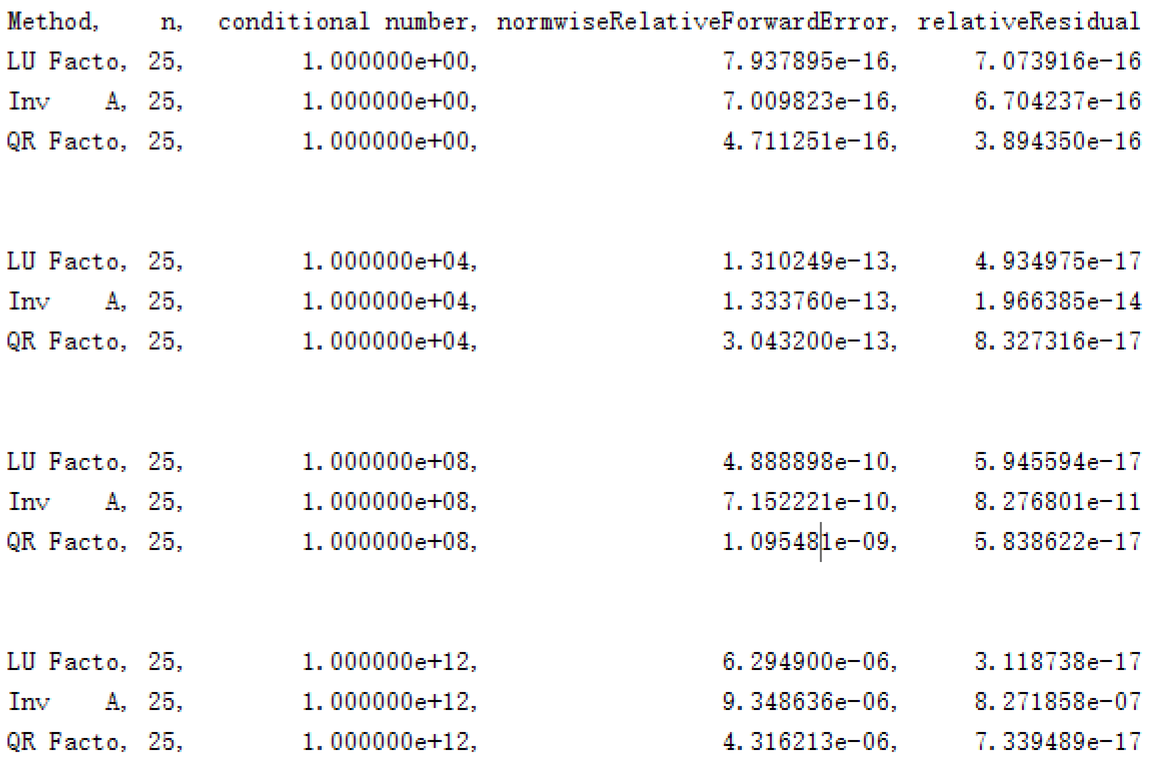
```
>> besselj(pi,0)

ans =

     0
```

Q2

Using gallery to generate matrix then solve

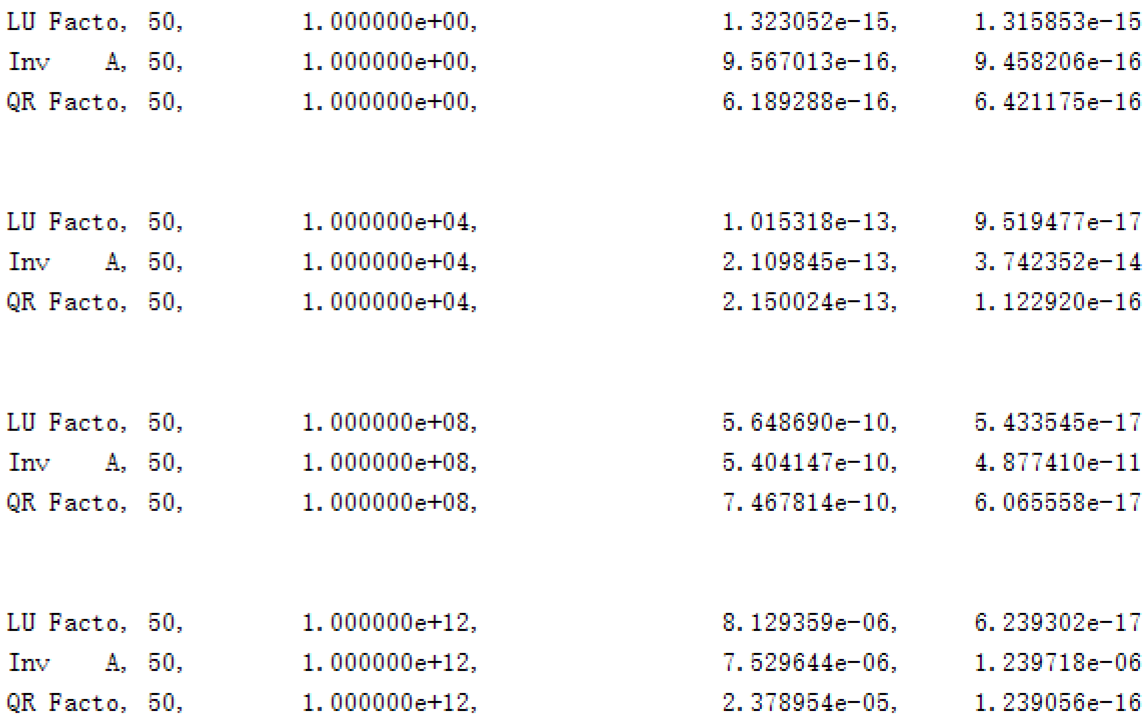


```
Method,     n,   conditional number, normwiseRelativeForwardError, relativeResidual
LU Facto,  25,         1.000000e+00,                  7.937895e-16,     7.073916e-16
Inv     A,  25,         1.000000e+00,                  7.009823e-16,     6.704237e-16
QR Facto,  25,         1.000000e+00,                  4.711251e-16,     3.894350e-16


LU Facto,  25,         1.000000e+04,                  1.310249e-13,     4.934975e-17
Inv     A,  25,         1.000000e+04,                  1.333760e-13,     1.966385e-14
QR Facto,  25,         1.000000e+04,                  3.043200e-13,     8.327316e-17


LU Facto,  25,         1.000000e+08,                  4.888898e-10,     5.945594e-17
Inv     A,  25,         1.000000e+08,                  7.152221e-10,     8.276801e-11
QR Facto,  25,         1.000000e+08,                  1.095481e-09,     5.838622e-17


LU Facto,  25,         1.000000e+12,                  6.294900e-06,     3.118738e-17
Inv     A,  25,         1.000000e+12,                  9.348636e-06,     8.271858e-07
QR Facto,  25,         1.000000e+12,                  4.316213e-06,     7.339489e-17
```



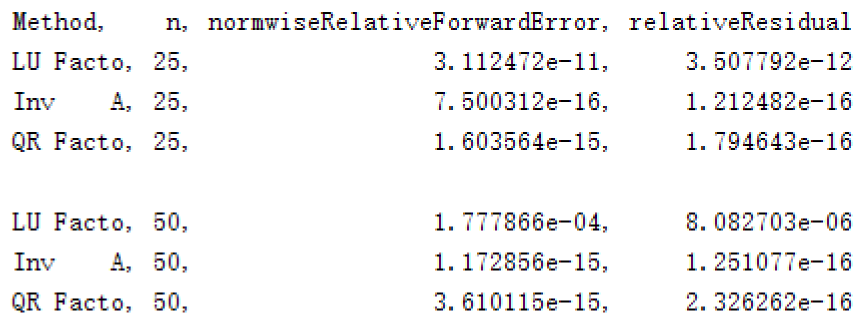
```
LU Facto,  50,         1.000000e+00,                  1.323052e-15,     1.315853e-15
Inv     A,  50,         1.000000e+00,                  9.567013e-16,     9.458206e-16
QR Facto,  50,         1.000000e+00,                  6.189288e-16,     6.421175e-16


LU Facto,  50,         1.000000e+04,                  1.015318e-13,     9.519477e-17
Inv     A,  50,         1.000000e+04,                  2.109845e-13,     3.742352e-14
QR Facto,  50,         1.000000e+04,                  2.150024e-13,     1.122920e-16


LU Facto,  50,         1.000000e+08,                  5.648690e-10,     5.433545e-17
Inv     A,  50,         1.000000e+08,                  5.404147e-10,     4.877410e-11
QR Facto,  50,         1.000000e+08,                  7.467814e-10,     6.065558e-17


LU Facto,  50,         1.000000e+12,                  8.129359e-06,     6.239302e-17
Inv     A,  50,         1.000000e+12,                  7.529644e-06,     1.239718e-06
QR Facto,  50,         1.000000e+12,                  2.378954e-05,     1.239056e-16
```

Using Gfpp to generate matrix A then solve



```
Method,     n, normwiseRelativeForwardError, relativeResidual
LU Facto, 25,                  3.112472e-11,     3.507792e-12
Inv    A, 25,                  7.500312e-16,     1.212482e-16
QR Facto, 25,                  1.603564e-15,     1.794643e-16

LU Facto, 50,                  1.777866e-04,     8.082703e-06
Inv    A, 50,                  1.172856e-15,     1.251077e-16
QR Facto, 50,                  3.610115e-15,     2.326262e-16
```

Knowing that backward stability is measure by the relative residuals. A stable method produces small relative residuals. And Accuracy is measure by the normal wise relative forward error.

For matrix generated by $gallery$:

For all three methods, the relative forward error increase as the conditional number increase. This is evident by theorem, $relative\ forward\ error\ \lesssim condition\ number$ So accuracy decrease as condition number increase, as forward error ~ condition number * backward error

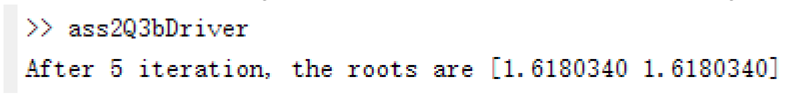
As the condition number increase, using Inv(A) to solve the equation will decrease the backward stability. This is because when condition number is large, A is close to singular, thus result in large relative residuals. And by theorem, large relative residuals will result in large relative backward error. Therefore, Inv(A) method is not backward stable as conditional number getting large. Whereas LU factorize and QR factorize method is more backward stable even conditional number is big,

For matrix generated by gallery, we can see LU factorization performs poorly on this matrix, whereas the other two methods are more backward stable and accurate.

The LU factorization method works backward unstable because Matrix generated by GFPP have maximal growth factors. By theorem, the backward error is determined by the growth factor. Bigger growth factors, higher backward error, therefore LU Factorization is not backward stable. And since forward error ~ condition number * backward error, the forward error also increases as growth factor increase. Therefore LU factorization is also not accurate for matrix generate by GFPP. Therefore, this method has higher forward error and backward error than the other two methods.

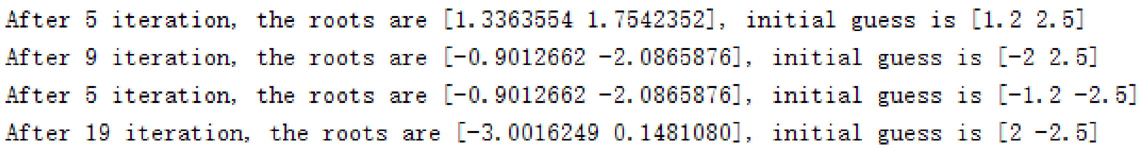
Q3

a. Written in the ass2Q3.m file
b. Written in the ass2Q3bDriver file with function in funcQ3b.m



```
>> ass2Q3bDriver
After 5 iteration, the roots are [1.6180340 1.6180340]
```

c. The Solution is as followed



```
After 5 iteration, the roots are [1.3363554 1.7542352], initial guess is [1.2 2.5]
After 9 iteration, the roots are [-0.9012662 -2.0865876], initial guess is [-2 2.5]
After 5 iteration, the roots are [-0.9012662 -2.0865876], initial guess is [-1.2 -2.5]
After 19 iteration, the roots are [-3.0016249 0.1481080], initial guess is [2 -2.5]
```

d. See the sufficient output section for the graph of iteration history and residual.
From the diagram Q3c6, we can see that there is 4 solutions to the system of equations. But our program only finds 3 sets of solutions.
The initial guess with (-1.2,-2.5) and (1.2, 2.5) works well with this algorithm. Evident by diagram Q3c1 and Q3c3, they find the solution in quickly with residual decreased every iteration.
Problem occurs when we use initial guess (-2, 2.5) and initial guess (2, -2.5)

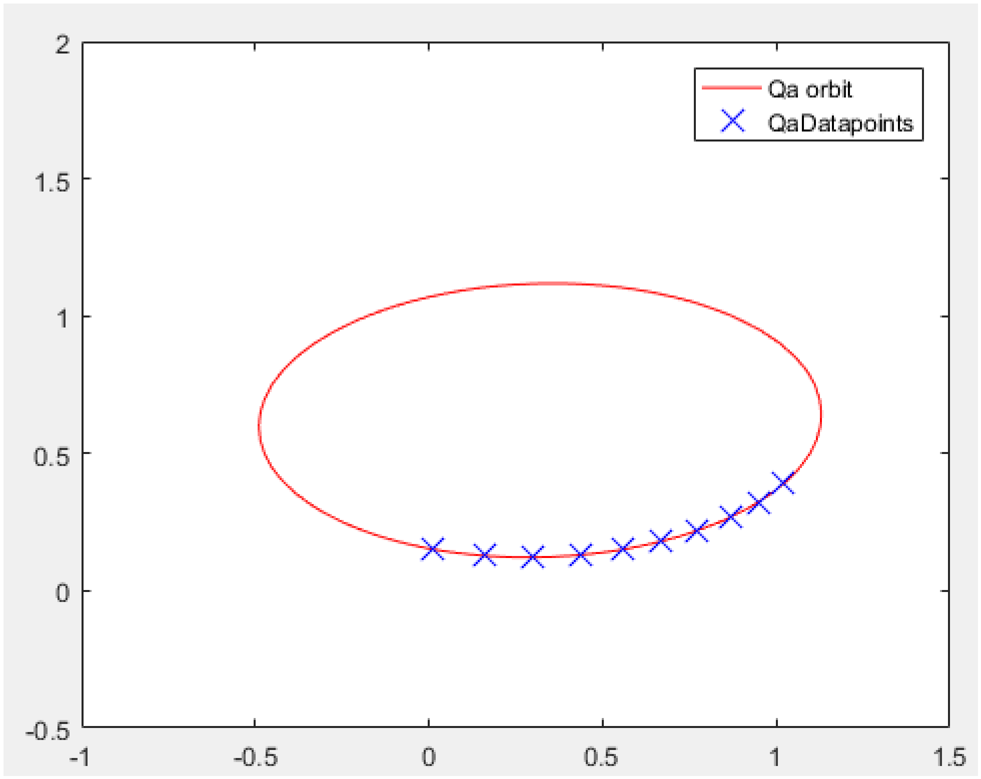
In the case (-2,2.5) , illustrated in the graph Q3c2, the algorithm has passed the nearest solution at its 3rd iteration due to a large step size result from small Jarcobian, therefore Xn has made a big job. After this iteration, the closet soution to xn+1 become (-0.9125522, -2.865876), and the algorithm then works as normal, end up with the same solution using initial guess(-2, 2.5)

The same problem also occurs with initial guess (2, -2.5), evident by graph Q3c4. At 3rd iteration, the xn+1 has made a big jump which is far far away from any solution. Therefore the residual has largely increased, then xn just follows the jarcobin and return back to the nearest solution step by step. Eventually it ends up with the solution that initial guess(-2, 2.5) should find.

From this two cases we can find that the value of jarcobian has great impact on the iteration process. When det(J(xn)) is relative close to 0, by solving the equation, the step size will be relative bigger, as sn = J (Xn) \ f(Xn). Therefore it is more likely to make a big jump from xn to xn+1, and missed the solution, thus result in possible deviation from the true solution that near the initial guess. In this case, Newton’s method would fail to find the nearby solutions around initial guess, as well as wasting many iterations to find another solution.

Q4

a. The constants for equation z = ax^2 + by^2 + cxy + dx + ey + f are:
a = -1, b = -2.6356, c = 0.1438, d = 0.5514, e = 3.2229, f = -0.43



b. To find the constant in the equation $x^2 = by^2 + cxy + dx + ey + f$, we set up equation $Ax = b$, with $A = [y^2\ xy\ x\ y\ f]$ and $b = x^2$
Therefore x = [b c d e f], which corresponds to the constant in equation
The singular values for our matrix A is (3.7860,0.9449,0.2089,0.0230,0.0055).
By theorem, the condition number for this matrix is 3.7860/0.0055 = 688.3636, which means this matrix is ill-conditioned.
Therefore, the solution, which is the value of the constant in the orbit equation, is sensitive to the input error in A and b of the equation in the system. So a small change in A and b might cause a relative big change in the orbit.

In this case, A and b are generated by using x and y. There we create small error in A and b by adding a random number in interval [-0.005, 0.005] to each element in given vector x, y, then plot the new orbit.

Since the Matrix A is ill-conditioned, we are expecting to get different orbits deviating from original orbit after perturbing the data. And from the Sufficient Ouput.pdf, Q4b section, we get different new orbits that are from smaller than original one to much bigger than the original one, whereas the data points are close to each other in each case. This evident that the equation is ill-conditioned. And we should find a new way to solve the constant in orbit.