Muhideen Ogunlowo HW7

Exercise 3.4.2

Symmetry: $\mathbf{P}^T = (\mathbf{I} - \rho\mathbf{v}\mathbf{v}^T)^T = I^T - \rho(v^T)^T v^T = I - \rho\mathbf{v}\mathbf{v}^T = \mathbf{P}$

Orthogonality:

$$
\begin{aligned}
&\mathbf{P}\mathbf{P}^T = \mathbf{P}\mathbf{P} \\
&= \left(I - \frac{2\mathbf{v}\mathbf{v}^T}{v^T v}\right)\left(I - \frac{2\mathbf{v}\mathbf{v}^T}{v^T v}\right) \\
&= \mathbf{I} - 4\frac{\mathbf{v}\mathbf{v}^T}{v^T v} + 4\frac{v(v^T v)v^T}{(v^T v)^2} \\
&= \mathbf{I} - 4\frac{\mathbf{v}\mathbf{v}^T}{v^T v} + 4\frac{\mathbf{v}\mathbf{v}^T}{v^T v} \\
&= \mathbf{I}
\end{aligned}
$$

Since $\mathbf{P}\mathbf{P}^T = \mathbf{I}$ we can conclude that the matrix P is orthogonal

Exercise 3.4.5

a) The asymptotic flop count of the QR procedure is given as $2\mathrm{mn}^2 - \frac{n^3}{3}$ while the asymptotic flop count for the LU factorization is given as $\frac{2}{3}n^3$. By comparison, we can only compare when m=n, which makes the flop count of the QR procedure $\frac{5}{3}n^3$. This is 2.5 times larger than the asymptotic flop count for the LU factorization.This means that the QR factorization takes more floating point operations. Which means that it takes longer to compute and the computational time is 2.5 times slower than that of the LU factorization method.

b)

$K_2(A) = ||A||_2 * ||A^{-1}||_2$ and $K_2(R) = ||R||_2 * ||R^{-1}||_2$

Given $A = \mathrm{QR}$ and knowing that Q is orthogonal we have

$||A||_2 = ||\mathrm{QR}||_2 = ||Q||_2 * ||R||_2 = ||R||_2$, because the two norm of an orthogonal matrix is 1

Using the property of the 2 norm of the inverse of an orthogonal matrix is also 1 and that $(\mathrm{AB})^{-1} = B^{-1} * A^{-1}$

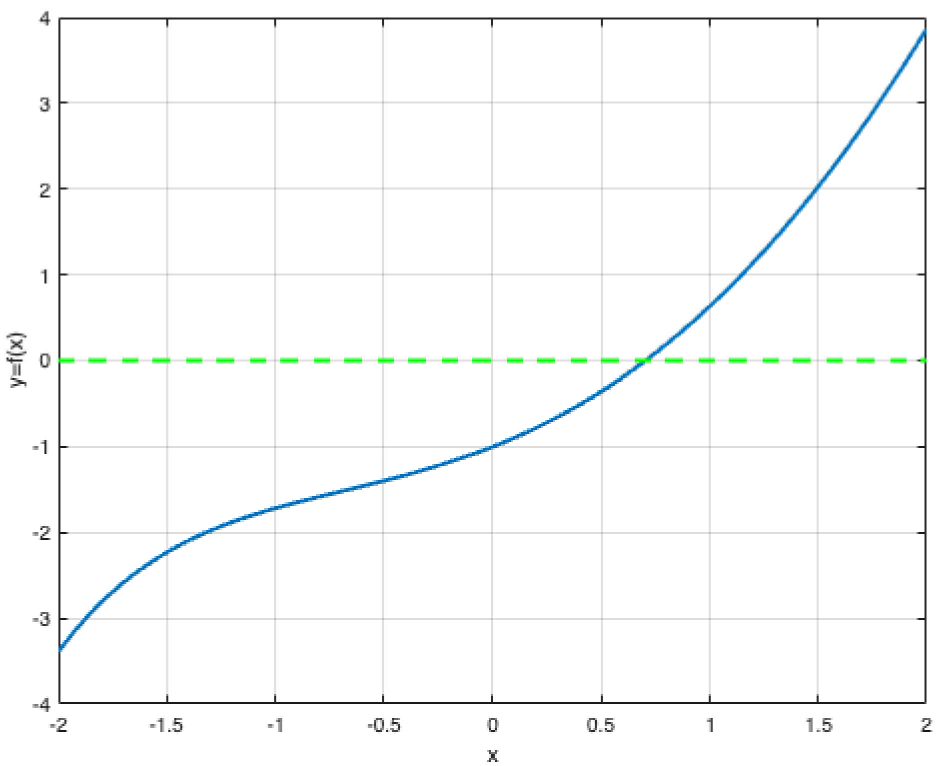
$||A^{-1}||_2 = ||R^{-1}Q^T||_2 = ||Q^T||_2 * ||R^{-1}||_2 = ||R^{-1}||_2$

Plug these values into the formula for the condition number :

$K_2(A) = ||R||_2 * ||R^{-1}||_2 = K_2(R)$

```
format long
%Exercise 4.1.1 a
f= @(x) (x.^2)- exp(-x); %function in rootfinding form

xx= linspace(-2,2,101); %Graph details, with interval of -2,2 and 101 points
figure
plot(xx, f(xx), 'linewidth', 2), grid on %command to plot graph
hold on
plot([-2 2], [0,0], 'g--', 'linewidth', 2) %details of graph with interval
and color plus starting point (0,0)
xlabel('x'), ylabel('y=f(x)') %label
```



Only one root lies in this interval

```
%Exercise 4.1.1 b
x0=1 %initial guess
```

```
x0 =
     1
```

```
x=fzero(f,x0) %Using Fzero
```

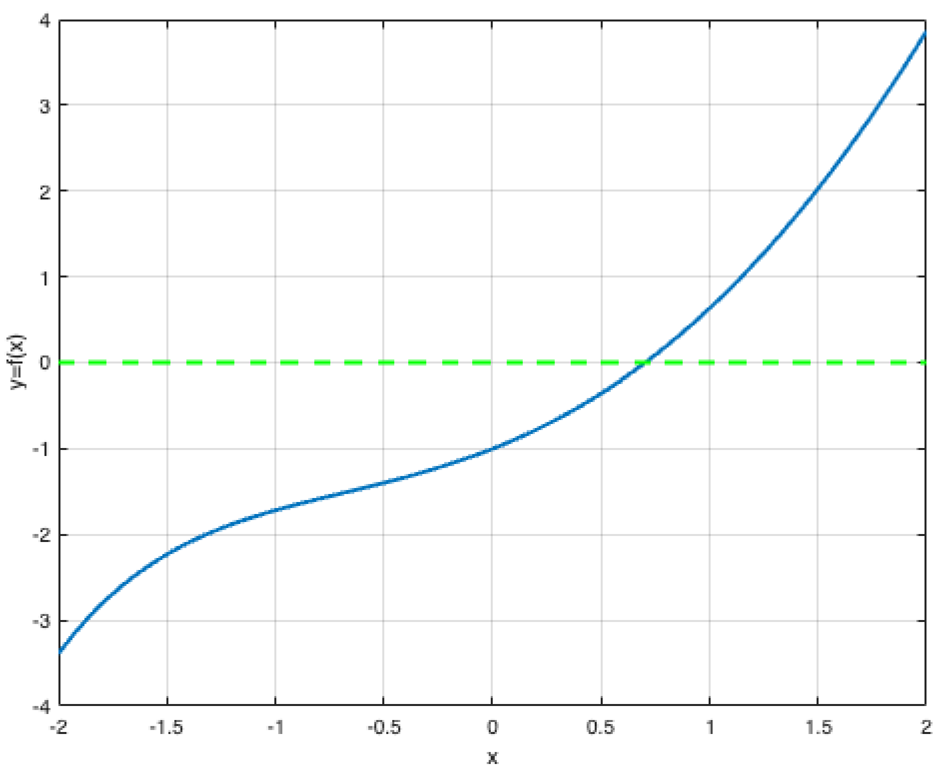
```
x =
   0.703467422498392
```

```
% Exercise 4.1.1 c
```

```
dfdx1= @(x) 2*x+exp(-x);
condition_number = abs(1/dfdx1(x));
disp(['Condition number of the root: ', num2str(condition_number)]);
```

```
Condition number of the root: 0.52582
```

```
format long
%Exercise 4.3.1 a
f= @(x) (x.^2)- exp(-x);%f(x) written in root finding form
dfdx= @(x) 2*x+ exp(-x); %dfdx
%Exercise 4.3.1 b
xx= linspace(-2,2,101); % Graph details with interval
figure
plot(xx, f(xx), 'linewidth', 2), grid on  %command to plot graph
hold on
plot([-2 2], [0,0], 'g--', 'linewidth', 2) %details of graph with interval
and color plus starting point (0,0)
xlabel('x'), ylabel('y=f(x)') %label
```



Only one root lies in this interval

```
%Exercise 4.3.1 c
x0=1 %Initial guess
```

```
x0 =
     1
```

```
a=fzero(f,x0)% Using Fzero to solve for root
```

```
a =
   0.703467422498392
```

```
%Exercise 4.3.1 d
g=newton(f,dfdx,1) %Iterates for Newton's method
```

```
g = 1×5
   1.000000000000000   0.733043605245445   0.703807786324133   0.703467468331798 ···
```

```
%Exercise 4.3.1 e
r=a %set r to original root
```

```
r =
   0.703467422498392
```

```
e = abs(g − r) %Definition of e as a vector of the errors in the Newton
Sequence
```

```
e = 1×5
   0.296532577501608   0.029576182747054   0.000340363825741   0.000000045833406 ···
```

```
ratios = e(2:end−1)./e(1:end−2).^2 %Ratios to determine quadratic
convergence
```

```
ratios = 1×3
   0.336354541475706   0.389098139781343   0.395635576485777
```

Since all ratios approach a constant as k increases, it is determined to be quadratic convergent

```
function x = newton (f, dfdx, x1) %d
% NEWTON Newton's method for a scalar equation.
% Input :
% f: objective function
% dfdx: derivative function
% x1: initial root approximation
% Output:
% x vector of root approximations (last one is best)
% Operating parameters.
funtol = 100*eps; xtol = 100*eps; maxiter = 40;
x=x1;
y = f(x1);
dx = Inf; % for initial pass below
k=1;

while (abs(dx) > xtol ) && (abs(y)> funtol) && (k < maxiter)
    dydx = dfdx(x(k));
    dx = -y/dydx;   %Newton step
    x(k+1) =x(k) +dx;
    k = k+1;
    y = f(x(k)) ;
```

```
end
if k==maxiter, warning('Maximum number of iterations reached.'),
end
end
```