Minh-Thi Nguyen
MAT321: Numerical Methods
Professor Nicolas Boumal
Due: 16 November 2018

MAT-APC 321: Numerical Methods Homework 5

1 A Dash of Systems of Linear Equations

1.1 Suli and Mayers Exercise 4.7

We are given the vector function

$$f_1(x_1,x_2) = x_1^2 + x_2^2 - 2 \quad (1)$$
$$f_2(x_1,x_2) = x_1 - x_2 \quad (2)$$

We verify that $f_1(x_1,x_2)=0$ and $f_2(x_1,x_2)=0$ have the solutions:

$$f_1(x_1,x_2) = x_1^2 + x_2^2 - 2 = 0 \to x_1^2 + x_2^2 = 2 \quad (3)$$
$$f_2(x_1,x_2) = x_1 - x_2 = 0 \to x_1 = x_2 \quad (4)$$
$$2x_1^2 = 2 \to x_1 = x_2 = \pm 1 \quad (5)$$

Applying Newton's Method, we obtain the first iteration:

$$x^{(1)} = x^{(0)} - [J_f(x^{(0)}]^{-1} f(x^{(0)} \quad (6)$$

We see that the Jacobian of the system is:

$$J_f(x^{(0)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & -1 \end{bmatrix} \quad (7)$$

Thus, we have

$$J_f^{-1}(x^{(0)}) = \begin{bmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & -1 \end{bmatrix}^{-1} = \frac{1}{\begin{vmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & -1 \end{vmatrix}} \begin{bmatrix} -1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} = \frac{1}{-2x_1^{(0)} - 2x_2^{(0)}} \begin{bmatrix} -1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} \quad (8)$$

Therefore, we obtain the first iteration:

$$x^{(1)} = x^{(0)} - [J_f(x^{(0)}]^{-1} f(x^{(0)}) = \quad (9)$$
$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} + \frac{1}{2(x_1^{(0)} + x_2^{(0)})} \begin{bmatrix} -1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} \begin{bmatrix} (x_1^{(0)})^2 + (x_2^{(0)})^2 - 2 \\ x_1^{(0)} - x_2^{(0)} \end{bmatrix} = \quad (10)$$
$$\begin{bmatrix} \frac{(x_1^{(0)})^2 + (x_2^{(0)})^2 + 2}{2(x_1^{(0)} + x_2^{(0)})} \\ \frac{(x_1^{(0)})^2 + (x_2^{(0)})^2 + 2}{2(x_1^{(0)} + x_2^{(0)})} \end{bmatrix} \quad (11)$$

Case 1: $x_1^{(0)} + x_2^{(0)} > 0$:
Thus, we see that because $(x_1^{(0)})^2 + (x_2^{(0)})^2 + 2 > 0$ for all values of x_1, x_2, we see that $x_1^{(1)} = x_2^{(1)} > 0$. Based on the iteration of Newton's Method, because $x_1^{(1)} = x_2^{(1)}$, we see that $x_1^{(2)} = x_2^{(2)}, x_1^{(3)} = x_2^{(3)}, \ldots, x_1^{(n)} = x_2^{(n)} > 0$, for $n > 1$ and as $n \to \infty$, with $x_1^{(n+1)} = x_2^{(n+1)} = \frac{2(x_1^{(n)})^2+2}{4(x_1^{(n)})} = \frac{(x_1^{(n)})^2+1}{2(x_1^{(n)})} = \frac{(x_2^{(n)})^2+1}{2(x_2^{(n)})}$

Thus, we have our sequence:

$$x^{(n+1)} = \frac{(x^{(n)})^2 + 1}{2(x^{(n)})} \tag{12}$$

$$x^{(n+1)} - 1 = \frac{(x^{(n)})^2 + 1 - 2(x^{(n)})}{2(x^{(n)})} = \frac{((x^{(n)}) - 1)^2}{2(x^{(n)})} \to x^{(n+1)} = \frac{((x^{(n)}) - 1)^2}{2(x^{(n)})} + 1 \geq 1 \tag{13}$$

We see that:

$$x^{(n+1)} - 1 = \frac{((x^{(n)}) - 1)^2}{2(x^{(n)})} = (\frac{(x^{(n)} - 1)}{2x^{(n)}}(x^{(n)} - 1)) \to \|x^{(n+1)} - 1\|_\infty = (\frac{(x^{(n)} - 1)}{2x^{(n)}})\|x^{(n)} - 1\|_\infty \tag{14}$$

So, we can represent:

$$L^{(n)} = (\frac{(x^{(n)} - 1)}{2x^{(n)}}) \tag{15}$$

We see that because $x^{(n)} \geq 1$ in (13), we see that $2x^{(n)} \geq 1 + x^{(n)} \geq x^{(n)} - 1$; thus, $\frac{(x^{(n)}-1)}{2x^{(n)}}) = L^{(n)} < 1$ for all $n > 1$.

$$\|x^{(n+1)} - 1\|_\infty = L^{(n)}\|x^{(n)} - 1\|_\infty = L^{(n)}L^{(n-1)}\|x^{(n-1)} - 1\|_\infty = \ldots = L^{(n)}L^{(n-1)}\ldots L^{(1)}\|x^{(0)} - 1\|_\infty \tag{16}$$

Therefore, we see that $\lim_{n\to\infty} \|x^{(n+1)} - 1\|_\infty = 0$ because $L^{(i)} < 1$ for all $i > 1$. Thus, we have as $n \to \infty$, $x^{(n)} \to 1$, so $x_1^{(n)} \to 1$ and $x_2^{(n)} \to 1$. Therefore, the sequence converges to $(1, 1)^T$, as desired.

Furthermore, to demonstrate that this sequence converges quadratically, we define $\varepsilon_k = \|x^{(k)} - 1\|$. Thus we see that:

$$\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \frac{\|x^{(n+1)} - 1\|}{\|x^{(n)} - 1\|^2} = \frac{1}{2x^{(n)}} = \frac{1}{2} > 0 \tag{17}$$

Therefore, because the ratio converges to a real number $\mu = \frac{1}{2} > 0$, we see that the iteration converges quadratically.

Case 2: $x_1^{(0)} + x_2^{(0)} < 0$:
Thus, we see that because $(x_1^{(0)})^2 + (x_2^{(0)})^2 + 2 > 0$ for all values of x_1, x_2, we see that $x_1^{(1)} = x_2^{(1)} < 0$. Based on the iteration of Newton's Method, because $x_1^{(1)} = x_2^{(1)}$, we see that $x_1^{(2)} =$

$x_2^{(2)}$,$x_1^{(3)} = x_2^{(3)}$, ... , $x_1^{(n)} = x_2^{(n)} < 0$, for $n > 1$ and as $n \to \infty$ We see that:

$$x^{(n+1)} = \frac{(x^{(n)})^2 + 1}{2(x^{(n)})} \tag{18}$$

$$x^{(n+1)} + 1 = x^{(n+1)} - (-1) = \frac{(x^{(n)})^2 + 1 + 2(x^{(n)})}{2(x^{(n)})} = \frac{((x^{(n)}) + 1)^2}{2(x^{(n)})} \to x^{(n+1)} - (-1) = \frac{((x^{(n)}) + 1)^2}{2(x^{(n)})} = \tag{19}$$

$$\frac{((x^{(n)}) + 1)}{2(x^{(n)})}(x^{(n)} + 1) = \frac{(x^{(n)}) + 1)}{2(x^{(n)})}(x^{(n)} - (-1)) \tag{20}$$

We see that because $x^{(n)} < 0$, $|x^{(n)} + 1| < |x^{(n)}| < |2x^{(n)}|$, so $|\frac{(x^{(n)})+1)}{2(x^{(n)})}| < 1$. Thus, we see that:

$$\|x^{(n+1)} - (-1)\|_\infty = (\frac{(x^{(n)} + 1)}{2x^{(n)}})\|x^{(n)} - (-1)\|_\infty \tag{21}$$

So, we can represent:

$$L^{(n)} = (\frac{(x^{(n)} + 1)}{2x^{(n)}}) \tag{22}$$

We see that $|L^{(n)}| < 1$ for all $n > 1$. So,

$$\|x^{(n+1)} - (-1)\|_\infty = L^{(n)}\|x^{(n)} - 1\|_\infty = L^{(n)}L^{(n-1)}\|x^{(n-(-1))} - 1\|_\infty = ... = L^{(n)}L^{(n-1)}...L^{(1)}\|x^{(0)} + 1\|_\infty \tag{23}$$

Therefore, we see that $\lim_{n\to\infty} \|x^{(n+1)}+1\|_\infty = 0$ because $|L^{(i)}| < 1$ Thus, we have as $n \to \infty$, $x^{(n)} \to -1$, so $x_1^{(n)} \to -1$ and $x_2^{(n)} \to -1$. Therefore, the sequence converges to $(-1, -1)^T$, as desired. Similar to Case 1, to demonstrate that this sequence converges quadratically, we define $\varepsilon_k = \|x^{(k)} + 1\|$. Thus we see that:

$$\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \frac{\|x^{(n+1)} + 1\|}{\|x^{(n)} + 1\|^2} = \frac{1}{2x^{(n)}} = \frac{1}{2} > 0 \tag{24}$$

Therefore, because the ratio converges to a real number $\mu = \frac{1}{2} > 0$, we see that the iteration converges quadratically.

1.2 Suli and Meyers Exercise 4.8

We are given that $\xi = lim_{k\to\infty x^{(k)}}$. Using Definition 1.4, we see that the sequence $x^{(k)}$ converges to ξ at least linearly if there exists a sequence $(\varepsilon_k) \to 0$ and a constant $\mu \in (0, 1)$ such that

$$\lim_{k\to\infty} \frac{\varepsilon_{k+1}}{\varepsilon_k} = \mu \tag{25}$$

with $\varepsilon_k = \|x^{(k)} - \xi\|_\infty$ and $p = -log_{10}\mu =$ the asymptotic rate of convergence, which describes the rate at which the convergence behavior is observed when the discretization is very small. We have the two functions $f_1(x_1, x_2) = x_1^2 + x_2^2 - 2$ and $f_2(x_1, x_2) = x_1 + x_2 - 2$. We see that $f_2(x_1, x_2) = x_1 + x_2 - 2 = 0 \to x_1 + x_2 = 2$ and $f_1(x_1, x_2) = x_1^2 + x_2^2 - 2 = 0 \to x_1^2 + x_2^2 = 2 \to x_1 = 1, x_2 = 1$ is a solution for the functions, so $(1, 1)^T$ is a solution so $f(\xi) = 0$ when $\xi = (1, 1)^T$. We assume now that $x_1^{(0)} \neq x_2^{(0)}$. We see that the Jacobian gives us:

$$J_f(x^{(0)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & 1 \end{bmatrix} \tag{26}$$

Thus, we have

$$J_f^{-1}(x^{(0)}) = \begin{bmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & 1 \end{bmatrix}^{-1} = \frac{1}{\begin{vmatrix} 2x_1^{(0)} & 2x_2^{(0)} \\ 1 & 1 \end{vmatrix}} \begin{bmatrix} 1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} = \frac{1}{2x_1^{(0)} - 2x_2^{(0)}} \begin{bmatrix} 1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} \tag{27}$$

Therefore, we obtain the first iteration:

$$x^{(1)} = x^{(0)} - [J_f(x^{(0)}]^{-1} f(x^{(0)}) = \tag{28}$$

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} - \frac{1}{2(x_1^{(0)} - x_2^{(0)})} \begin{bmatrix} 1 & -2x_2^{(0)} \\ -1 & 2x_1^{(0)} \end{bmatrix} \begin{bmatrix} (x_1^{(0)})^2 + (x_2^{(0)})^2 - 2 \\ x_1^{(0)} - x_2^{(0)-2} \end{bmatrix} = \tag{29}$$

$$\begin{bmatrix} \frac{(x_1^{(0)})^2 + (x_2^{(0)})^2 - 4(x_2^{(0)}) + 2}{2(x_1^{(0)} - x_2^{(0)})} \\ \frac{-(x_1^{(0)})^2 - (x_2^{(0)})^2 + 4x_1^{(0)}) - 2}{2(x_1^{(0)} - x_2^{(0)})} \end{bmatrix} \tag{30}$$

Thus, we see that $x_1^{(1)} + x_2^{(1)} = \frac{4(x_1^{(0)} - x_2^{(0)})}{2(x_1^{(0)} - x_2^{(0)})} = 2$. We see that when $x_1^{(0)} = 1 + \alpha$ and $x_2^{(0)} = 1 - \alpha$, we have $x_1^{(0)} - x_2^{(0)} = 2\alpha$. Thus, we have:

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \frac{1}{4\alpha} \begin{bmatrix} 2\alpha^2 + 4\alpha \\ 4\alpha - 2\alpha^2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \alpha + 2 \\ 2 - \alpha \end{bmatrix} = \begin{bmatrix} 1 + \frac{\alpha}{2} \\ 1 - \frac{\alpha}{2} \end{bmatrix} \tag{31}$$

We thus see that given the initialization of $x_1^{(0)}$ and $x_2^{(0)}$, we see that we have the iteration:

$$\begin{bmatrix} x_1^{(n)} \\ x_2^{(n)} \end{bmatrix} = \begin{bmatrix} 1 + \frac{\alpha}{2^n} \\ 1 - \frac{\alpha}{2^n} \end{bmatrix} \tag{32}$$

Thus, we have:

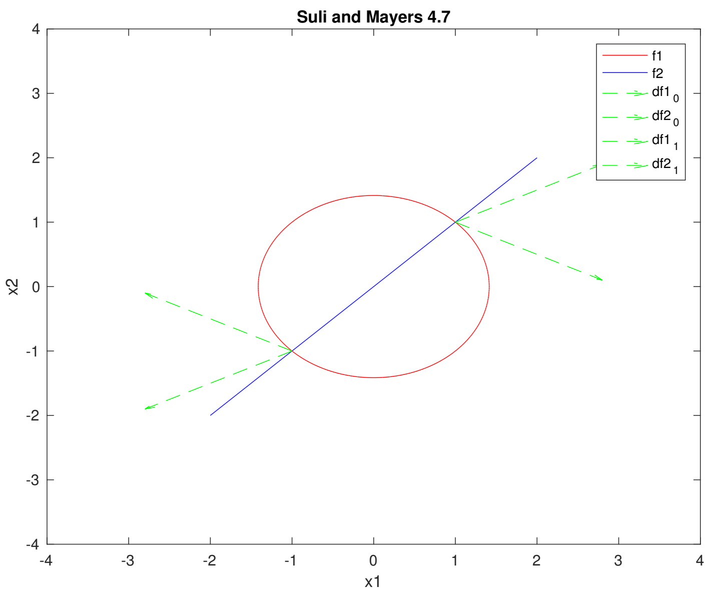
$$\begin{bmatrix} x_1^{(n)} - 1 \\ x_2^{(n)} - 1 \end{bmatrix} = \begin{bmatrix} \frac{\alpha}{2^n} \\ -\frac{\alpha}{2^n} \end{bmatrix} \tag{33}$$

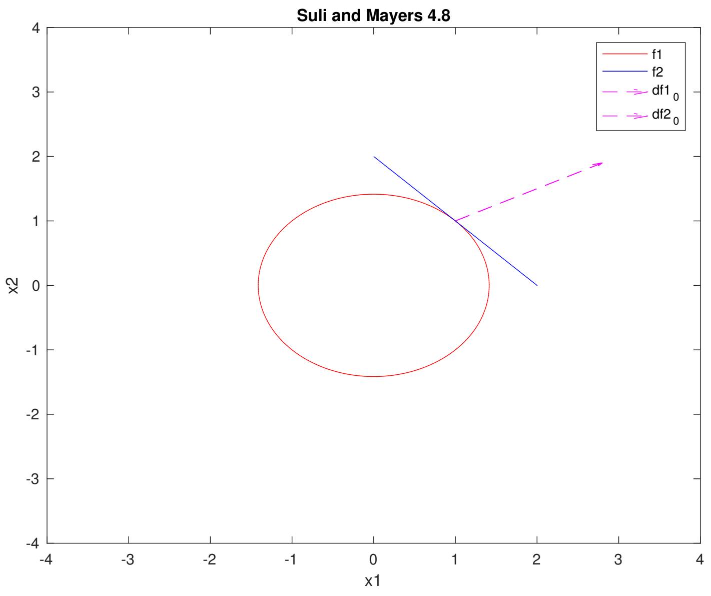
So, we have: $\|x^{(n)} - 1\| = |\frac{\alpha}{2^n}|$. Thus, we have: $\lim_{n\to\infty} \|x^{(n)} - 1\| = \lim_{n\to\infty} |\frac{\alpha}{2^n}| = 0$. Thus, we have as $n \to \infty$, $x^{(n)} \to 1$, so $x_1^{(n)} \to 1$ and $x_2^{(n)} \to 1$. Therefore, the sequence converges to $(1, 1)^T$, as desired. We see that the sequence converges linearly because we can define $\varepsilon_k = \|x^{(k)} - 1\|$ and see that:

$$\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n} = \frac{\|x^{(n+1)} - 1\|}{\|x^{(n)} - 1\|} = \frac{|\frac{\alpha}{2^{n+1}}|}{|\frac{\alpha}{2^n}|} = \frac{1}{2} = \mu > 0 \tag{34}$$

We also see that $\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^m}$ diverges for $m > 1$, so it does not converge any greater than linearly, and so for the case $m = 2$, we do not have quadratic convergence. Thus, we have that the iteration converges linearly with an asymptotic rate of $-\log_{10}\mu = -\log_{10}(1/2) = \log_{10}(2)$, as desired. We also see that the iteration does not converge linearly because the Jacobian is singular at $(1, 1)^T$

We draw the sets $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$, as shown below:





Which are obtained from the code for Problem 4.7:

```
%draw the sets of f1 and f2 from the Suli and Mayers exercises
figure
%Suli and Mayers 4.7
%define f
f = @(x) [x(1)^2 + x(2)^2 -2; x(1) - x(2)];
%define partials
df1 = @(x1) [2*x1 ; 1];
df2 = @(x2) [2*x2 ; -1];

%define solution points
x_0 = [1.0; 1.0];
x_1 = [-1.0, -1.0];

%plot f1 = 0 and f2 = 0
```

```
figure;
x1 = linspace(-2,2,401);
x2 = linspace(-2,2,401);
[X1, X2] = meshgrid(x1, x2);

%plot f1 = 0
f1 = X1.^2 + X2.^2 -2;
contour(x1,x2,f1,[0,0],'r');

%plot details
hold on;
xlim([-4 4])
ylim([-4 4])
xlabel('x1')
ylabel('x2')
title('Suli and Mayers 4.7')

%plot f2 = 0
f2 = X1 - X2;
hold on;
contour(x1, x2, f2, [0,0],'b')

%graph the gradients of f1 and f2
quiver(x_0(1),x_0(2), 2*x_0(1), 1,'g--');
quiver(x_0(1),x_0(2), 2*x_0(2), -1,'g--');
quiver(x_1(1),x_1(2), 2*x_1(1), 1,'g--');
quiver(x_1(1),x_1(2), 2*x_1(2), -1,'g--');

legend('f1','f2', 'df1_0', 'df2_0', 'df1_1', 'df2_1')
```

and for Problem 4.8

```
%draw the sets of f1 and f2 from the Suli and Mayers exercises
figure
%Suli and Mayers 4.8
%define f
f = @(x) [x(1)^2 + x(2)^2 -2; x(1) + x(2)-2];

%solution point
x_0 = [1.0; 1.0];

%partials
df1 = @(x1) [2*x1 ; 1];
df2 = @(x2) [2*x2 ; 1];

%plot f1 = 0 and f2 = 0
x1 = linspace(-2,2,401);
x2 = linspace(-2,2,401);
[X1, X2] = meshgrid(x1, x2);

%f1 =0
f1 = X1.^2 + X2.^2 -2;
contour(x1,x2,f1,[0,0],'r');
hold on;

%f2 =0
```

```
f2 = X1 + X2-2;
hold on;
contour(x1, x2, f2, [0,0],'b')

%plot gradients
quiver(x_0(1),x_0(2), 2*x_0(1), 1,'m--');
quiver(x_0(1),x_0(2), 2*x_0(2), 1,'m--');

%plot details
hold on;
xlim([-4 4])
ylim([-4 4])
xlabel('x1')
ylabel('x2')
title('Suli and Mayers 4.8')
legend('f1','f2', 'df1_0', 'df2_0')
```

We know that the first case (4.7) has quadratic convergence, while the second case (4.8) only has linear convergence. We also graph the gradients of f_1 and f_2 at the solutions for both cases, ultimately observing that in the first case, the gradients are noncolinear and linearly independent, while in the second case at the solution, the gradients are colinear and are thus linearly dependent because in the first case, the f_2 function intersects f_1 at two points, while in the second case it is tangent to f_1. We see that by Theorem 4.4 in Suli and Mayers, the sequence defined by Newton's Method converges quadratically when not only f and its second partials are defined and continuous, but it also requires that the Jacobian at the solution point ξ to be nonsingular. Thus, in Case 1, because the gradients are linearly independent, the Jacobian is nonsingular at the solution points; however in Case 2, the gradients are linearly dependent, so the Jacobian is singular at the solution point, and the iteration does not converge quadratically.

2 Finding Roots of a Polynomial via eig

We are given the polynomial $p(x) = x^n + c_1x^{n-1} + c_2x^{n-2} + ... + c_{n-1}x + c_n$.
We have the matrix:

$$A = \begin{pmatrix} -c_1 & -c_2 & -c_3 & ... & -c_n \\ 1 & 0 & 0 & & \\ & 1 & 0 & 0 & \\ & & \ddots & 0 & 0 \\ & & & 1 & 0 \end{pmatrix}$$

We wish to find the eigenvalues of A, which we can obtain by acquiring the matrix A-λI and taking its determinant. Thus, we have:

$$A - \lambda I = \begin{pmatrix} -c_1 - \lambda & -c_2 & -c_3 & ... & -c_n \\ 1 & -\lambda & 0 & & \\ & 1 & -\lambda & 0 & \\ & & \ddots & 0 - \lambda & 0 \\ & & & 1 & -\lambda \end{pmatrix}$$

So, we have det(A-λI) =

$$p_A(\lambda) = \det(A - \lambda I) = (-c_1 - \lambda)\begin{vmatrix} -\lambda & 0 & ... & ... & 0 \\ 1 & -\lambda & & & \\ 0 & 1 & -\lambda & & \\ & & & & \\ & & & & -\lambda \end{vmatrix} - (\begin{vmatrix} -c_2 & -c_3 & ... & ... & -c_n \\ 1 & -\lambda & & & \\ 0 & 1 & -\lambda & & \\ & & & & \\ & & & & -\lambda \end{vmatrix}) \tag{35}$$

$$= (-c_1 - \lambda)(-\lambda)^{n-1} - ((-c_2)(-\lambda)^{n-2} - ((-c_3)(-\lambda)^{n-3} ... - (-c_n)(-\lambda)^{n-n}))) = \tag{36}$$

$$(-c_1 - \lambda)(-\lambda)^{n-1} + c_2(-\lambda)^{n-2} - c_3(-\lambda)^{n-3} ... = \tag{37}$$

$$(-\lambda)^n - c_1(-\lambda)^{n-1} + c_2(-\lambda)^{n-2} + ... = \tag{38}$$

$$(-1)^n(\lambda)^n - c_1(-1)^{n-1}(\lambda)^{n-1} + c_2(-1)^{n-2}(\lambda)^{n-2} + ... = \tag{39}$$

$$(-1)^n(\lambda)^n + c_1(-1)^n(\lambda)^{n-1} + c_2(-1)^n(\lambda)^{n-2} + ... = \tag{40}$$

$$(-1)^n((\lambda)^n + c_1(\lambda)^{n-1} + c_2(\lambda)^{n-2} + ...) = \tag{41}$$

$$(-1)^n p(\lambda) \tag{42}$$

Thus, we can conclude that $p_A(x) = (-1)^n p(x)$, as desired to demonstrate that the eigenvalues of A are the solutions to the characteristic polynomial, which is the same as the function $p(x)$, so the eigenvalues of A are solutions of the characteristic polynomial, which are the roots of p.

3 Power and Inverse Iteration

3.1 Task 1

We implement a function that implements the Rayleigh Taylor Iteration to obtain an eigenpair (x, μ) of A, also returning the number of iterations run. Our algorithm can be described as follows:

Algorithm 1 Rayleigh Quotient Iteration Algorithm

```
 1: Assign a random vector to $x$
 2: Solve for $v$: $(A - \mu_0 I)v = x$                ▷ Implement RQI, explained below
 3: $\mu_1 \leftarrow \mu_0 + \frac{v^T x}{v^T v}$
 4: $x \leftarrow v/\|v\|$
 5: k ← 1                                              ▷ Initialize the count
 6: err ← $\mu_1 - \mu_0$        ▷ Establish error variable between eigenvalues obtained
 7: while err > tolerance do
 8:     $\mu_0 \leftarrow \mu_1$
 9:     Solve for $v$: $(A - \mu_0 I)v = x$
10:     $\mu_1 \leftarrow \mu_0 + \frac{v^T x}{v^T v}$
11:     $x \leftarrow v/\|v\|$                         ▷ update $x$
12:     err ← $\mu_1 - \mu_0$
13:     $k \leftarrow k + 1$                           ▷ update counnt
14: end while
15: $\mu \leftarrow \mu_1$                             ▷ final eigenvalue
```

We avoid matrix-vector multiplication by observing that using the Rayleigh Taylor Iteration formula for the vectors $x^{(k)}$:

$$x^{(k+1)} = \frac{(A - \mu_k I)^{-1} x^{(k)}}{\|(A - \mu_k I)^{-1} x^{(k)}\|_2} \tag{43}$$

Let $(A - \mu_k I)^{-1} x^{(k)} = v$ so $x^{(k+1)} = \frac{v}{\|v\|_2}$. We know that the formula for the eigenvalue iteration is:

$$\mu_{k+1} = \frac{x^{(k+1)T} A x^{(k+1)}}{x^{(k+1)T} x^{(k+1)}} = \tag{44}$$

$$\frac{\frac{v}{\|v\|_2})^T A \frac{v}{\|v\|_2}}{(\frac{v}{\|v\|_2})^T \frac{v}{\|v\|_2}} = \frac{(v^T A v}{v^T v} = \frac{(v^T(\mu_k v + x^{(k)}))}{v^T v} = \tag{45}$$

$$\mu_{k+1} = \mu_k + \frac{v^T x^{(0)}}{v^T v} \tag{46}$$

$$x^{(k+1)} = \frac{v}{\|v\|_2} \tag{47}$$

We implement our algorithm in MATLAB, as shown below, using a stopping criterion that looks at the error between the eigenvalue obtained from the last two iterates to observe convergence:

```
%Problem 3 Task 1
function[x, mu, k] = rqi(A, mu)

%define a tolerance for stopping criterion
tolerance = 10^(-16);

%determine the size of A, A is symmetric
[m,n] = size(A);

%find the vector x_0
%initialize starting vector
x = randn(m,1);
mu_0 = mu;

%solve for the next x_1
M2 = A - mu_0*eye(m);
v = M2 \ x;

%update eigenvalue
mu_1 = mu_0 + v'*x/(v'*v);
%update eigenvector
x = v/norm(v);

%initiate error
err = abs(mu_1 - mu_0);

%initialize the count
k = 1;

%start loop
while((err > tolerance))

    %initialize loop
    mu_0 = mu_1;

    % RQI iteration
    B = A - mu_0*eye(m);
    y = B \ x;

    %update eigenvalue
    mu_1 = mu_0 + (y'*x)/(y'*y);

    %update eigenvector
    x = y/norm(y);

    %update count
    k = k+1;

    %update error
    err = abs(mu_1 - mu_0);
end

%final values
mu = mu_1;
x = x;
k = k;

end
```

3.2 Task 2

1. We are given that A is symmetric and $A = PHP^T$, where P is orthogonal and H is tridiagonal. Because A is symmetric, we can use the Spectral Theorem to obtain a factorization of A such that $A = Q^T DQ$, where Q is an orthonormal matrix with the eigenvectors of A and D is a diagonal matrix with the eigenvalues of A. Since H is tridiagonal, H must also be symmetric. By the Spectral Theorem, we can factor $H = Q_1^T D_1 Q_1$, with D_1 giving the eigenvalues of H. Thus, we see that:

$$A = PHP^T \to QDQ^T = PQ_1D_1Q_1^TP^T \to D = (Q^TPQ_1)D_1Q_1^TP^TQ = (Q^TPQ_1)D_1(Q^TPQ_1)^T \tag{48}$$

Thus, we see that (Q^TPQ_1) is an orthogonal matrix because the product of orthogonal matrices is orthogonal and Q^T, Q_1, and P are all orthogonal because Q is orthonormal, so Q^T is also orthogonal. Thus, we can represent $D = Q'D_1(Q')^T$ for an orthogonal matrix Q', so $D_1 = (Q')^TD(Q')$. Thus, we see that $D = D_1$ because the two matrices are similar because $(Q')^T = (Q')^{-1}$. Therefore, because D contains the eigenvalues of A, and D_1 contains the eigenvalues of H, A and H must have the same eigenvalues.

We let x be an eigenvector of H and λ be its corresponding eigenvalue. Then, we have: $Hx = \lambda x$. So, $A(Px) = (PHP^T)(Px) = PHx = P(\lambda x) = \lambda(Px)$. Thus, Px is an eigenvector of A, as desired.

2. Looking at our RQI algorithm, we see that the most computational steps are the solving of linear systems and the transformation of A into a tridiagonal matrix. Let's consider the process of trying to find m eigenvalues, so we must apply the iteration for at least (but actually a lot more) m times. We first consider the case of calling rqi on A t times, with $t > m$. We see that we need to solve the linear system $(A - \mu_0 I)v = x$, for v, which requires n^3 flops.

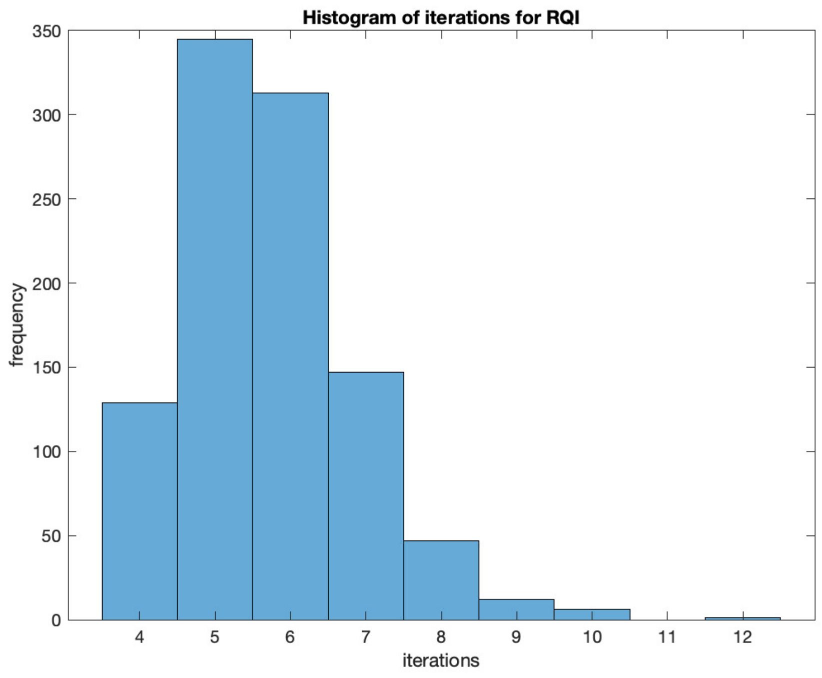
Then, to update μ, we require two inner products, which require $2n + 2n = 4n$ flops, thus giving us a total of $4n + 1$, including the division. Then, to normalize x, we require another inner product (and a square root and division), thus giving us another $2n + 2$ flops. Finding the error requires subtraction, giving us another flop. Thus, we see that within the RQI algorithm, after running k iterations of Rayleigh Quotient iteration, we see that rqi(A) costs us $k(n^3 + 4n + 1 + 2n + 2) = k(n^3 + 6n + 3) \approx k(n^3 + 6n)$ for finding a single eigenpair, and $tk(n^3 + 6n)$ for finding m eigenvalues. Meanwhile, let's consider implementing rqi(H) to find the eigenvalues. We first have to tridiagonalize A, which requires $\frac{4}{3}n^3$ flops. Then, the complexity of the algorithm is similar to that of finding rqi(A), as described above, but in the solving of the linear system, in lieu of requiring n^3 flops, solving the linear system for H only reuqires $8n$ flops, as proven earlier. Thus, to find one eigenvalue, implementing RQI on H, we see that the total complexity is: $\frac{4}{3}n^3 + k(8n + 6n + 3) = \frac{4}{3}n^3 + k(14n + 3)$. Thus, to find m eigenvalues, we see that it requires: $\frac{4}{3}n^3 + tk(14n + 3)$ total flops because we only have to tridiagonalize A once. Therefore, it is better to implement rqi on A only when $tk(n^3 + 6n) < \frac{4}{3}n^3 + tk(14n + 3)$, which would only occur when $n = 1$. Thus, it is more reasonable to apply the RQI algorithm on the tridiagonal matrix H to find the eigenvalues of A.

3. We wish to calculate all the eigenvalues of A. Nevertheless, although the RQI algorithm allows crazy fast convergence, the eigenpair obtained from rqi is unpredictable, since we know that the eigenvalues lie within the range $[-b, b]$, we sample the initialization of μ uniformly distributed along the range given, observing all the different eigenvalues, and obtaining distinct eigenvalues only using the "uniquetol" MATLAB function. This the most pragmatic method at the moment, but we can also apply a QR algorithm or use deflation for greater efficiency with less initialization. We sample 1000 initializations and set the tolerance for similarity to be 10^{-8} to ensure distinction. Furthermore, for rqi, we wish to obtain accurate values for the eigenvalue, thus using a stopping criteria with tolerance $= 10^{-15}$. We implement the algorithm below:

```
%find the eigenvalues of A
n = 10;
A = randn(n);
A = A + A';
b = norm(A,1); %eigenvalues lie between [-b,b]
eig_min = -b;
eig_max = b;

%eigenpair unpredictable
%Initialize based on uniform distribution

%transform A into tridiagonal matrix
[P,H] = hess(A);

%count the total number of eigenvalues obtained, should get 10

%sample the eigenvalues
m = 1000;
sample = eig_min + rand(1,m)*(eig_max - eig_min);

%record eigenvalues and iterations
eigen_values = zeros(1,m);
iterations = zeros(1,m);

%loop for each initialization
```

```
for i = 1:m

    %apply rqi
    [x, mu, k] = rqi(H,sample(i));

    %record iterations and eigenvalues obtained
    iterations(i) = k;
    eigen_values(i) = mu;
end

%establish tolerance to find distinct eigenvalues
tolerance = 10^(-8);

%obtain distinct eigenvalues
eigenvalues = uniquetol(eigen_values, tolerance)

%plot histogram of iterations
histogram(iterations)
%plot details
title('Histogram of iterations for RQI')
xlabel('iterations')
ylabel('frequency')

%theoretical eigenvalues of A using MATLAB
e = eig(A);

%difference between values
diff = eigenvalues - e'
```

The histogram obtained is shown, and we observe that the number of iterations is very small, so we can do many initiations to find all the eigenvalues:



4. We compare our obtained eigenvalues with the ones obtained by the MATLAB function.

RQI Alg.	MATLAB	Diff.
-8.09884206769612	-8.09884206769612	5.32907051820075e-15
-5.520252426233283	-5.520252426233279	-3.55271367880050e-15
-1.65579887445641	-1.65579887445641	2.22044604925031e-16
-0.847999781317824	-0.847999781317825	2.22044604925031e-16
0.592131635591141	0.592131635591141	-1.11022302462516e-16
1.44001403602594	1.44001403602594	0.00
2.71162398006582	2.71162398006582	4.44089209850063e-16
3.85894293718234	3.85894293718234	-8.88178419700125e-16
6.77311706549425	6.77311706549424	1.77635683940025e-15
9.30160440703088	9.30160440703089	-5.32907051820075e-15

4 Singular Value Decomopositions

4.1 M^TM

We are given that the singular values of M are $\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n$. We know that for the matrix M, we can apply SVD decomposition to express $M = U\Sigma V^T$, where U is an $n \times n$ orthogonal matrix, Σ is a diagonal matrix with the singular values of M, and V^T is also an orthogonal matrix. We observe that $M^TM = (V\sigma U^T)(U\Sigma V^T) = V\Sigma^2V^T$. Thus, because M^TM is a symmetric matrix, we know we can factorize it into QDQ^T such that Q is an orthogonal matrix and D is the diagonal matrix with the eigenvalues of M^TM. Thus, we see that $D = \Sigma^2$, so we see that the eigenvalues of M^TM are the elements of the diagonal matrix Σ^2, which gives us the n eigenvalues $\sigma_1^2, \sigma_2^2, ...\sigma_n^2$, as desired.

4.2 H

We define:

$$H = \begin{bmatrix} 0 & M^T \\ M & 0 \end{bmatrix} \to H^T = \begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix} \to H^TH = \begin{bmatrix} M^2 & 0 \\ 0 & (M^T)^2 \end{bmatrix} \tag{49}$$

We know that because M is a square matrix, M and M^T have the same eigenvalues. So, the eigenvalues of M^2 are the same as the eigenvalues of $(M^T)^2$ and the same as the eigenvalues of M^TM, which are $\sigma_1^2, \sigma_2^2, ...\sigma_n^2$. We see that the eigenvalues of H^TH are the solutions to the characteristic polynomial $\det(H^TH - \lambda I) = \det((M^2 - \lambda I)((M^T)^2 - \lambda I)) = \det((M^2 - \lambda I)\det((M^T)^2 - \lambda I) = 0$. So, the eigenvalues of H^TH are the eigenvalues of M^2 with algebraic multiplicity 2, which are: $\sigma_1^2, \sigma_2^2, ...\sigma_n^2$. We know that H is a $(2n) \times (2n)$ matrix, so we know that if the eigenvalues of H are $\lambda_1, \lambda_2, ...\lambda_2$, the the eigenvalues of H^TH are $\lambda_1^2, \lambda_2^2, ...\lambda_{2n}^2$. Furthermore, we know that $Tr(H) = 0$ because the diagonals of H are 0. Thus, the sum of all the eigenvalues of H is 0. Thus, because the eigenvalues of H^TH are $\sigma_1^2, \sigma_2^2, ...\sigma_n^2$, in order for the eigenvalues of H to have a sum of 0, the eigenvalues of H must be: $\pm\sigma_1, \pm\sigma_2, ... \pm \sigma_n$, giving us a total of $2n$ eigenvalues total, as desired.

We implement the two methods to compute the singular values of M:

```
%Problem Set 3 Problem 4
%Implement two methods for computing singular values of M

```

```
n1 = 4;
n2 = 8;
n3 = 16;

M1 = hilb(n1);
M2 = hilb(n2);
M3 = hilb(n3);

%Method 1
eig1 = eig(M1'*M1);
sing1_M = sort(sqrt(eig1));

eig2 = eig(M2'*M2);
sing2_M = sort(sqrt(eig2));

eig3 = eig(M3'*M3);
sing3_M = sort(sqrt(eig3));

%Method2
H1 = zeros(2*n1,2*n1);
H1(1:n1,n1+1:2*n1) = M1';
H1(n1+1:2*n1,1:n1) = M1;
new_eig1 = sort(eig(H1));
new_sing1_M = sort((new_eig1));
new_sing1_M = new_sing1_M(n1+1:2*n1);

H2 = zeros(2*n2,2*n2);
H2(1:n2,n2+1:2*n2) = M2';
H2(n2+1:2*n2,1:n2) = M2;
new_eig2 = sort(eig(H2));
new_sing2_M = sort((new_eig2));
new_sing2_M = new_sing2_M(n2+1:2*n2);

H3 = zeros(2*n3,2*n3);
H3(1:n3,n3+1:2*n3) = M3';
H3(n3+1:2*n3,1:n3) = M3;
new_eig3 = sort(eig(H3));
new_sing3_M = sort((new_eig3));
new_sing3_M = new_sing3_M(n3+1:2*n3);

%Theoretical computation
s1 = sort(svd(M1));
s2 = sort(svd(M2));
s3 = sort(svd(M3));

%absolute errors
method1_abs1 = abs(sing1_M - s1);
fprintf('%.16e\n',max(method1_abs1));

method1_abs2 = abs(sing2_M - s2);
fprintf('%.16e\n',max(method1_abs2));

method1_abs3 = abs(sing3_M - s3);
fprintf('%.16e\n',max(method1_abs3));

```

```
method2_abs1 = abs(new_sing1_M - s1);
fprintf('%.16e\n',max(method2_abs1));

method2_abs2 = abs(new_sing2_M - s2);
fprintf('%.16e\n',max(method2_abs2));

method2_abs3 = abs(new_sing3_M - s3);
fprintf('%.16e\n',max(method2_abs3));

%relative errors
method1_rel1 = method1_abs1./s1;
fprintf('%.16e\n',max(method1_rel1));

method1_rel2 = abs(sing2_M - s2)./s2;
fprintf('%.16e\n',max(method1_rel2));

method1_rel3 = abs(sing3_M - s3)./s3;
fprintf('%.16e\n',max(method1_rel3));

method2_rel1 = abs(new_sing1_M - s1)./s1;
fprintf('%.16e\n',max(method2_rel1));

method2_rel2 = abs(new_sing2_M - s2)./s2;
fprintf('%.16e\n',max(method2_rel2));

method2_rel3 = abs(new_sing3_M - s3)./s3;
fprintf('%.16e\n',max(method2_rel3));
```

Which outputs:

```
Method 1:

abs. error for Method 1 for n = 4:
1.0422089894660674e-13

abs. error for Method 1 for n = 8:
9.1796267999915002e-09

abs. error for Method 1 for n = 16:
9.1413932053414628e-09
------------------------------------
Method 2:

abs. error for Method 2 for n = 4:
8.8817841970012523e-16

abs. error for Method 2 for n = 8:
2.2204460492503131e-16

abs. error for Method 2 for n = 16:
8.8817841970012523e-16
------------------------------------
Method 1:
```

```
rel. error for Method 1 for n = 4:
1.0777499047203662e-09

rel. error for Method 1 for n = 8:
8.2584839330019420e+01

rel. error for Method 1 for n = 16:
4.4945132089125127e+08
------------------------------------
Method 2:

rel. error for Method 2 for n = 4:
2.5366587074026078e-13

rel. error for Method 2 for n = 8:
1.6220178702984963e-07

rel. error for Method 2 for n = 16:
1.6778415245530567e+00
```

Hence, we observe that the relative error for the singular values of M when the matrix is large becomes smaller with the second method because for larger matrices, M^TM requires multiplication of two matrices and then Method 1 requires taking the square root of eigenvalues then obtained, thus incurring larger error for the singular values. Meanwhile, in the second method, we do not have to manipulate any floating numbers, just take the largest n values (which will give us positive), so the error is smaller.