1. (a) Find the absolute and relative condition numbers of $f(x) = e^{-2x}$ and $f(x) = \ln^3(x)$. For what values of x are these functions sensitive to perturbations?

Solution. For $f(x) = e^{-2x}$,

$$\kappa_A = \|J_f(x)\| = |2e^{-2x}|$$
$$\kappa_r = \frac{\|J_f(x)\||x|}{|e^{-2x}|} = 2|x|$$

That is, the absolute condition number is never sensitive since e^{-2x} is bounded above by 1. The relative condition number is only sensitive when x gets really large. When $f(x) = \ln^3(x)$, we have

$$\kappa_A = \frac{3\ln^2(x)}{|x|}$$
$$\kappa_r = \frac{\kappa_A|x|}{|\ln(x)|\ln^2(x)} = \frac{3}{\ln(x)}$$

The absolute condition number becomes large as $x \to 0^+$. The relative conditioning behaves in a similar manner when $x \to 1$. Both are incredibly sensitive to perturbations around their respective critical values.

(b) Let $x_1, x_2 \in \mathbb{R}^+$ and $f(x_1, x_2) = x_1^{x_2}$. Find the relative condition number of $f(x)$, and for what range of values of x_1 and x_2 is this problem ill-conditioned?

Solution. Let $f(x_1, x_2) = x_1^{x_2}$. Then

$$\kappa_r = \frac{\|J_f(x)\|_\infty \|x\|_\infty}{\|f(x)\|} = \frac{\max\{x_1^{x_2-1}x_2, x_1^{x_2-1}x_1\log x_2\}\max\{x_1, x_2\}}{|x_1^{x_2-1}||x_1|}$$
$$= \frac{\max\{x_2, x_1\log x_2\}\max\{x_1, x_2\}}{|x_1|}$$

From this construction we see that if $x_1 \log x_2 > x_2$, then our relative condition number is simply $\log x_2 \max\{x_1, x_2\}$. This is only ill-conditioned when both x_1 and x_2 are very (very) large. However, if $x_1 \log x_2 < x_2$, then our relative condition number is $\frac{x_2^2}{x_1}$ which can clearly grow very fast when $x_1 \to 0$ or $x_2 \to \infty$.

2. Consider the recurrence $x_k = 111 - \frac{1130 - \frac{3000}{x_{k-1}}}{x_k}$ who general solution is $x_k = \frac{100^{k+1}a + 6^{k+1}b + 5^{k+1}c}{100^k a + 6^k b + 5^k c}$, where a, b and c depend on the initial values. Given $x_0 = \frac{11}{2}$ and $x_1 = \frac{61}{11}$, we have $a = 0, b = c = 1$.

(a) Show that this gives a monotonically increasing sequence to 6.

Solution. Consider a rescaling by a factor of 6^k for a fixed k. Then

$$\frac{6^{k+1} + 5^{k+1}}{6^k + 5^k} = \frac{6 + 5(\frac{5}{6})^k}{1 + (\frac{5}{6})^k}$$

It follows that

$$\lim_{k\to\infty} \frac{6^{k+1} + 5^{k+1}}{6^k + 5^k} = \lim_{k\to\infty} \frac{6 + 5(\frac{5}{6})^k}{1 + (\frac{5}{6})^k} = 6$$

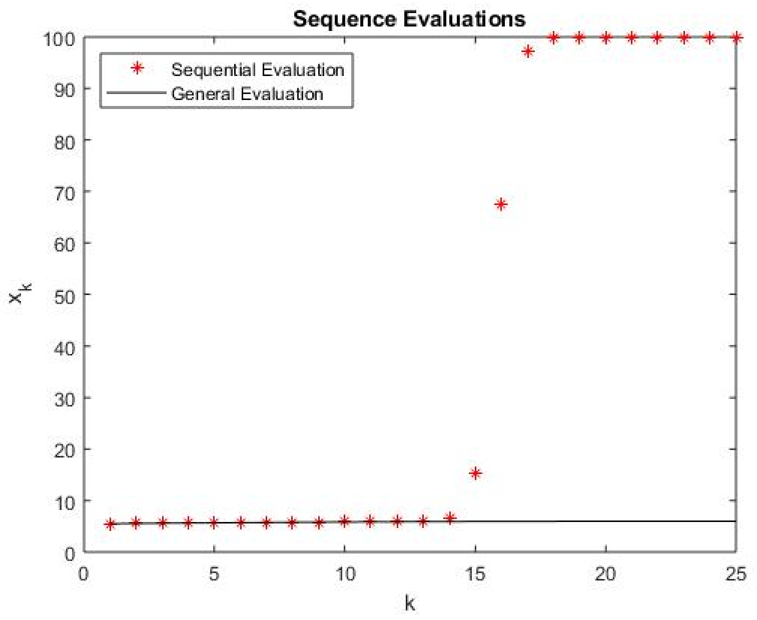

Figure 1: Sequence x_k

For monotonicity, note that as a function of k, $f(k) = \frac{6^{k+1}+5^{k+1}}{6^k+5^k}$ has a derivative of $f'(k) = \frac{30^k \log(\frac{6}{5})}{(6^k+5^k)^2}$ which is positive for all nonnegative values of k.

(b) Implement this recurrence on MATLAB, plot $\{x_k\}$, compare with the exact solution. What is the condition number of the limit of this particular sequence as a function of x_0 and x_1?

Solution. The conditioning of the sequence as a function of x_0 and x_1 is infinity. For any perturbation of size $\varepsilon > 0$ on the inputs x_0, x_1, we see that the limit jumps from 6 to 100 (since a becomes nonzero). That is, $\frac{100-6}{\varepsilon} \to \infty$ as $\varepsilon \to 0$. We see this behavior in Figure 1. After only a few iterations, the sequence shoots to 100. The general solution avoids this problem by setting $a = 0$ in exact arithmetic.

3. Let $p_2 4(x) = (x-1)(x-2)\cdots(x-24) = a_0 + a_1 x + \cdots + a_{24}x^{24}$. Evaluate the relative condition number of the k-th root $x_k = k$ subject to the perturbation of a_k for $k = 16, 17, 18, 19, 20$ and find the root that is most sensitive to the perturbation of the corresponding coefficient. Use MATLAB to compute the roots and compare them to the true roots.

Solution. Recall from class that for $p_2 4$, we have that

$$\kappa_r = \frac{x_j^{i-1} a_i}{p'(x_j)}$$

which simplifies to $\frac{x_k^{k-1} a_k}{p'(x_k)}$ for this problem since $i = j = k$. The computed results are summarized in Table 1. We can clearly see that $k = 16$ is the most sensitive root to

Root	**Relative Condition Number**
$k = 16$	$-2.12e16$
$k = 17$	$7.88e14$
$k = 18$	$-1.77e15$
$k = 19$	$-1.59e15$
$k = 20$	$-2.31e15$

Table 1: Relative Conditioning of Roots

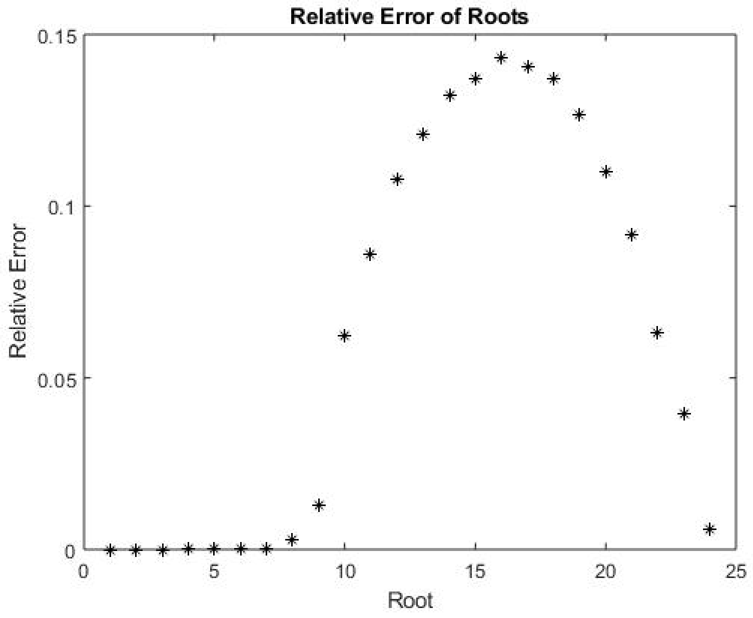

Figure 2: Root Relative Errors

perturbations of the corresponding coefficient. Using MATLAB to compute the roots, we see that numerous roots had a relative error of over 0.1. Because of the large relative condition number at these roots, this is already terrible. Figure 2 shows the relative error of the computed roots.

4. Let $x_0, \dots, x_n$ be $n+1$ equidistant points on $[-1, 1]$ where $x_0 = -1, x_n = 1$. Use MATLAB's vander to generate Vandermonde matrices for $n = 9, 19, 29, 39$. Let $x = [1, \dots, 1]^T$ and $b = Ax$. Pretend that we do not know x and use numerical algorithms to solve for x. Let $\hat{x}$ be the computed solution. Compute the relative forward errors and the smallest relative backward errors for GEPP, QR factorization, Cramer's Rule, $A^{-1}b$, and GE without pivoting. Comment on the forward/backward stability of these methods.

Solution. Table 2 presents the numerical results for $n = 39$. We immediately see that both GEPP and QR factorization appear to be numerically backward stable. The final three algorithms cannot make such claim, but two of them, $A^{-1}b$ and Cramer's Rule appear to at least be forward stable, i.e. they produce forward errors similar to the

Algorithm	**Forward Error**	**Backward Error**
GEPP	4.1723	$0.2113e-16$
QR Factorization	4.5289	$0.0408e-15$
$A^{-1}b$	4.7941	0.0065
Cramer's Rule	2.3003	0.0784
GE without pivot	$1.828e28$	0.0134

Table 2: Algorithm Stability

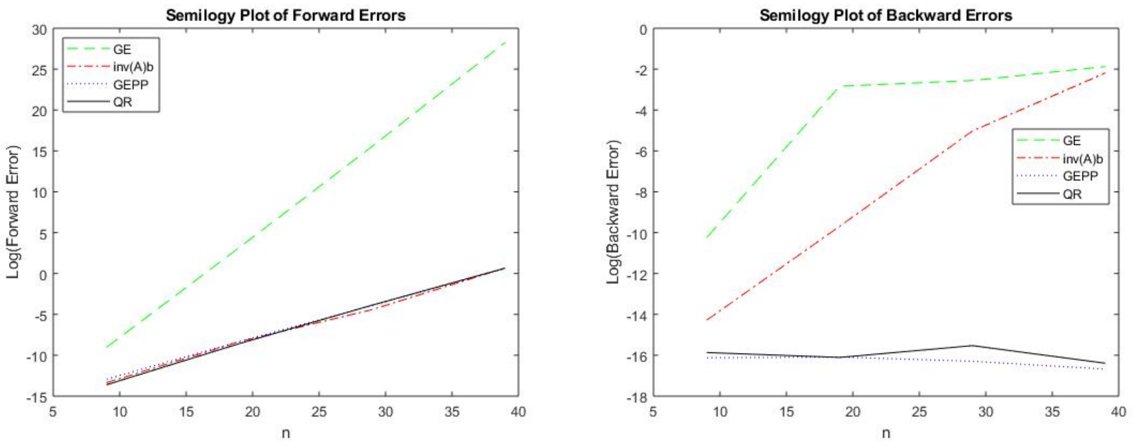

Figure 3: Stability Graphs

forward errors of a backwards stable algorithm. However, for GE without pivoting, it is neither forward nor backward stable, for such a large n. Admittedly, it is likely the large n that creates the most problems. The forward error for $n = 9$ of GE without pivoting is on the order of 10^{-13}. Full evaluations can be found in Figure 3.

5. Though pivoting is needed for factorizing general matrices, it is not needed for symmetric positive definite and diagonally dominant matrices.

   (a) For a symmetric positive definite A, with the one-step Cholesky factorization

$$A = \begin{bmatrix} a_{11} & w^T \\ w & K \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{w}{\sqrt{a_{11}}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - \frac{ww^T}{a_{11}} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{w^T}{\sqrt{a_{11}}} \\ 0 & I \end{bmatrix} = R_1^T A_1 R_1$$

   show that the submatrix $K - \frac{ww^T}{a_{11}}$ is symmetric positive definite. Consequently the factorization can be completed without break-down. Then, show that $\|R\|_2 = \|A\|_2^{\frac{1}{2}}$, which means the elements in R are uniformly bounded by that of $\|A\|$. Explain why this observation leads to the backward stability of Cholesky factorization.

   Solution. Since $\det(R_1^T) = \det(R_1) = \sqrt{a_{11}} > 0$, and all other leading principal minors of R_1, R_1^T are positive (the submatrix is the identity in both cases), these matrices must be invertible and symmetric positive definite. Thus, their inverses must also be SPD. So $A_1 = R_1^{-T} A R_1^{-1}$ is also positive definite since it is the product of 3 SPD matrices. Finally, because $K - \frac{ww^T}{a_{11}}$ is a principal minor of A_1 which itself is SPD, it must also be SPD. Recall from class that the magnitude of ρ_n controls the backward stability of an LU factorization algorithm.

To see the equivalence of norms, let $R = U\Sigma V^T$ be a singular value decomposition of R. We can then compute

$$A = R^T R = V\Sigma^2 V^T$$

which is a singular value decomposition of A. From here we see that the singular values of R are the square root of the corresponding singular values in A. Because both R and A are symmetric, this is also true for the eigenvalues. Thus,

$$\|R\|_2 = \rho(R) = \rho(A)^{\frac{1}{2}} = \|A\|_2^{\frac{1}{2}}$$

Because the elements of R are uniformly bounded by $\|A\|$, the growth factor is incredibly well behaved for Cholesky factorization and it is consequently backwards stable.

(b) Suppose that $A = \begin{bmatrix} \alpha & w^T \\ v & C \end{bmatrix}$ is column diagonally dominant, with one step LU factorization $A = \begin{bmatrix} 1 & 0 \\ \frac{v}{\alpha} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C - \frac{1}{\alpha}vw^T \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & I \end{bmatrix}$. Show that the submatrix $C - \frac{1}{\alpha}vw^T$ is also column diagonally dominant, and no pivoting is needed.

Solution. For brevity, let $D = C - \frac{1}{\alpha}vw^T$ and consequently, $A = \begin{bmatrix} \alpha & w^T \\ v & D + \frac{vw^T}{\alpha} \end{bmatrix}$. Suppose for the sake of contradiction that there exists some column j such that $D_{jj} < \sum_{i=1, i\neq j}^{n} D_{ij}$, i.e. D is not strictly diagonally dominant. Note that since A is column diagonally dominant, we have $\sum_{i=1}^{n} v_i < \alpha$. We may rearrange this to obtain $\frac{1}{\alpha}\left[\sum_{i=1, i\neq j} v_i w_j + v_j w_j\right] < w_j$. Combining the previous two inequalities gives us

$$D_{jj} + \frac{v_j w_j}{\alpha} < \sum_{i=1, i\neq j}^{n} \left(D_{ij} + \frac{v_i w_j}{\alpha}\right) + w_j$$

which is precisely the condition that column j of A is not strictly diagonally dominant, a contradiction.

(c) Show that worst-case growth factor $\rho_n = 2^{n-1}$ for GEPP. However, we construct matrices with random elements, each are independent samples from the normal distribution of means 0 and standard deviation $\frac{1}{\sqrt{n}}$. Let $n = 32, 64, \ldots, 2048$, and for each n, repeat the experiment 5000 times. Find the percent of experiments when $\rho_n > \sqrt{n}$. Comment on the chance of having a large ρ_n.

Solution. After quite some time in the cluster[1], I've actually fully completed this problem. For full details, see errors.output in the appendix. It is clear from the results in Table 3 that the likelihood of having a troublesome growth factor is very small. Of the results tallied, they only indicate an instance of $\rho_n > \sqrt{n}$, which itself is not incredibly intimidating. However, it is important to note that many applications in practice are not on matrices sampled from a normal distribution. Often times, systems are banded or at least sparse. If I had written an LU factorization algorithm that took advantage of these special structures, I would test this as well, but I have not. Nonetheless, GEPP is most likely a very backward stable algorithm in practice.

n	**Bad Growth Factor Count**
32	15
64	13
128	19
256	16
512	9
1024	8
2048	5

Table 3: Large Growth Factor Frequencies

6. Consider the eigenvalue problem $Av = \lambda v$. Let $(\hat{\lambda}, \hat{v})$ be a computed eigenpair, which is assumed to be the exact eigenpair of a perturbed matrix $A + \Delta A$. Show that the minimum 2-norm of all ΔA is $\frac{\|A\hat{v} - \hat{\lambda}\hat{v}\|_2}{\|\hat{v}\|_2}$ and find a particular ΔA whose 2-norm is the minimum.

   Solution. Consider the perturbed equation $(A + \Delta A)\hat{v} = \hat{\lambda}\hat{v}$. Rearranging gives $\Delta A\hat{v} = \hat{\lambda}\hat{v} - A\hat{v}$. It follows that $\|A\hat{v} - \hat{\lambda}\hat{v}\|_2 = \|\Delta A\hat{v}\|_2 \leq \|\Delta A\|_2\|\hat{v}\|_2$. Thus,

$$\frac{\|A\hat{v} - \hat{\lambda}\hat{v}\|_2}{\|\hat{v}\|_2} \leq \|\Delta A\|_2$$

   To find a matrix satisfying this inequality, we will need a lemma. Let $u, v \in \mathbb{R}^n$. Then $\|uv^T\|_2 = \|u\|_2\|v\|_2$.

   Proof. Let $u, v \in \mathbb{R}^n$. To prove the lemma, first set $\tilde{u}, \tilde{v}$ to be unit vectors in the directions of u, v respectively, i.e. $\tilde{u}\|u\|_2 = u, \tilde{v}\|v\|_2 = v$. Let U, V to be orthonormal basis extensions of u, v and $A = uv^T$. Denote E to be the zero $n \times n$ matrix with a 1 in the top left entry. Then $\tilde{u}\tilde{v}^T = \tilde{A} = UEV^T$ is a singular value decomposition of $\hat{A}$. Scaling up to A, we see that $\|u\|_2\|v\|_2$ is the only nontrivial singular value of A. Thus, it must be the 2-norm. $\square$

   Now set $r = \hat{\lambda}\hat{v} - A\hat{v}$. Then consider $\Delta A = \frac{r\hat{v}^T}{\hat{v}^T\hat{v}}$. From the lemma we have $\|r\hat{v}^T\|_2 = \|r\|_2\|\hat{v}\|_2$. Consequently, it follows that

$$\|\Delta A\|_2 = \frac{\|r\|_2\|\hat{v}\|_2}{\|\hat{v}\|_2^2} = \frac{\|A\hat{v} - \hat{\lambda}\hat{v}\|_2\|\hat{v}\|_2}{\|\hat{v}\|_2^2} = \frac{\|A\hat{v} - \hat{\lambda}\hat{v}\|_2}{\|\hat{v}\|_2}$$

[1]Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster

Thus we have found a matrix satisfying the minimum 2-norm.

1 Appendix

1.1 Script files

1.1.1 Question 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 1
%
% Problem
%   2
%
% Function Dependencies
%   None
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

length = 25;
f = @(k) (6.^(k+1) + 5.^(k+1))./(6.^k + 5.^k);
xk = zeros(1,length);
fk = f(0:length-1);
xk(1) = 11/2;
xk(2) = 61/11;

for i = 2:length-1
    xk(i+1) = 111 - (1130-3000/xk(i-1))/xk(i);
end

plot(1:length, xk,'r*', 1:length, fk, '-k')
title('Sequence Evaluations')
ylabel('x_k')
xlabel('k')
legend('Sequential Evaluation','General Evaluation','location','northwest')
```

1.1.2 Question 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 1
%
% Problem
%   3
%
% Function Dependencies
%   None
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

load 'wilk24mc.mat'

%Part a

int = 16:20;
xk = int;
ak = wCoeff(25-int)';
wPrime = polyder(wCoeff)';

dpxk = polyval(wPrime,int);


conditioning = xk.^(int-1).*ak./dpxk;

[val,loc] = max(abs(conditioning));

fprintf('The %d-th root is most sensitive to perturbations in the %d-th coefficient with a rela

%Part b
compRoots = roots(wCoeff);
compRoots = sort(compRoots);
err = (1:24)'-compRoots;
relErr = err./(1:24)';
```

```
for i = 1:length(relErr)
    relErr(i) = norm(relErr(i));
end

plot(1:24,relErr,'*k')
title('Relative Error of Roots')
ylabel('Relative Error')
xlabel('Root')
```

1.1.3 Question 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 1
%
% Problem
%   4
%
% Function Dependencies
%   cramersRule.m
%   matSolve.m
%   pluFact.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

n = [9,19,29,39];
forErrMat = zeros(1,length(n));
backErrMat = zeros(1,length(n));

forErrQR = zeros(1,length(n));
backErrQR = zeros(1,length(n));

forErrCramer = zeros(1,length(n));
backErrCramer = zeros(1,length(n));

forErrAinv = zeros(1,length(n));
backErrAinv = zeros(1,length(n));

forErrGE = zeros(1,length(n));
backErrGE = zeros(1,length(n));

for j = 1:length(n)
x = linspace(-1,1,n(j)+1)';
xtrue = ones(n(j)+1,1);
A = vander(x);
normA = norm(A);
```

```
b = A*xtrue;


%Matlab's backslash
xhat = A\b;
forErrMat(j) = norm(xhat-xtrue)/norm(xtrue);
backErrMat(j) = norm(b-A*xhat)/normA/norm(xhat);

%QR Factorization
[Q,R] = qr(A);
newb = Q'*b;
xhat = R\newb;
forErrQR(j) = norm(xhat-xtrue)/norm(xtrue);
backErrQR(j) = norm(b-A*xhat)/normA/norm(xhat);

%Cramer's Rule
xhat = cramersRule(A,b);
forErrCramer(j) = norm(xhat-xtrue)/norm(xtrue);
backErrCramer(j) = norm(b-A*xhat)/normA/norm(xhat);

%A inverse b
xhat = inv(A)*b;
forErrAinv(j) = norm(xhat-xtrue)/norm(xtrue);
backErrAinv(j) = norm(b-A*xhat)/normA/norm(xhat);

%Gaussian Elimination without Pivoting
xhat = matSolve(A,b);
forErrGE(j) = norm(xhat-xtrue)/norm(xtrue);
backErrGE(j) = norm(b-A*xhat)/normA/norm(xhat);

end

plot(n,log10(forErrGE),'g--')
hold on
plot(n,log10(forErrAinv),'r-.')
hold on
plot(n,log10(forErrMat),'b:')
hold on
plot(n,log10(forErrQR),'k-')

title('Semilogy Plot of Forward Errors')
xlabel('n')
ylabel('Log(Forward Error)')
legend('GE', 'inv(A)b','GEPP','QR','location','best')
```

```
figure()
plot(n,log10(backErrGE),'g--')
hold on
plot(n,log10(backErrAinv),'r-.')
hold on
plot(n,log10(backErrMat),'b:')
hold on
plot(n,log10(backErrQR),'k-')

title('Semilogy Plot of Backward Errors')
xlabel('n')
ylabel('Log(Backward Error)')
legend('GE', 'inv(A)b','GEPP','QR','location','best')
```

1.1.4 Question 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 1
%
% Problem
%   5
%
% Function Dependencies
%   None
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;
tic

exp = 5:11;
n = 2.^exp;
k = length(n);
trials = 5000;
counts = zeros(1,k);
saveFreq = 100;

for i = 1:trials
    for j = 1:k
        A = randn(n(j),n(j))/sqrt(n(j));
        [~,~,growth] = pluFact(A,'pivot');
        if growth > sqrt(n(j))
            counts(j) = counts(j) + 1;
        end
    end
    if mod(i,saveFreq) == 0
save('results')
    end
end
time = toc;

save('results')
```

1.2 Accompanying Functions

1.2.1 Batch Script for Palmetto

```
#!/bin/bash
#
#PBS -N xueHW1Q5
#PBS -l select=1:ncpus=8:mem=125gb
#PBS -l walltime=72:00:00
#PBS -m abe
#PBS -M tsquire@g.clemson.edu
#PBS -j oe
#PBS -o errors

module add matlab/2017a

cd /home/tsquire/CompMath/8610\ Homeworks

taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nodesktop -nosplash -r xueHW1Q5
```

1.2.2 Output information for Batch Script

```
+-------------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+-------------------------------------------+

mem=125gb,ncpus=8,walltime=72:00:00


+--------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+--------------------------------------+

cpupercent=381,cput=124:14:14,mem=738772kb,ncpus=8,vmem=5675028kb,walltime=36:44:58
```

1.2.3 Cramer's Rule

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cramersRule.m
%
% DESCRIPTION
%   Solves a system of equations Ax = b using Cramer's Rule
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - n x n  matrix
%   b - n x 1 vector
%
% OUTPUT
%   x - solution to Ax = b
%
% NOTES
%   Issues a warning if matrix is rank deficient
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = cramersRule(A,b)

%Check sizes of inputs
[m,n] = size(A);
assert(m == n);
assert(isvector(b));
assert(n == length(b));
[m,n] = size(b);

if m<n
    b = b';
end

n = length(b);
x = zeros(n,1);
detA = det(A);

for i = 1:n
    tmpA = A;
    tmpA(:,i) = b;
    x(i) = det(tmpA)/detA;
end
```

1.2.4 Linear System Solver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BACKSUB.m
%
% DESCRIPTION
%   Solves Ax = b using LU factorization with optional partial pivoting
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%   b - Nx1 vector
%
% OUTPUT
%   x - solution to Ax = b
%
% NOTES
%   Asserts the matrix is square and the vector is of appropriate length
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = matSolve(A,b,~)
%% Assertions
n = size(A);
assert(n(1) == n(2))

assert(isvector(b));
assert(length(b) == n(1));

n = n(1);

%% Solving Ax = b
if nargin > 2
    [A,p] = pluFact(A,'pivot'); %factorize with pivoting
else
    [A,p] = pluFact(A); %factorize without pivoting
end
b = b(p); %repermute

diagonal = diag(A);
A(1:n+1:n^2) = ones(1,n); %change the diagonal in the combined LU matrix
y = forwardSub(A,b); %solve Ly = b
A(1:n+1:n^2) = diagonal;
```

```
x = backSub(A,y); %solve Ux = y


end
```

1.2.5 GEPP Factorization Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pluFact.m
%
% DESCRIPTION
%   Decomposes a matrix A using LU factorization with optional partial pivoting.
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%
% OUTPUT
%   A - Decomposed L/U matrix for efficient space storage
%   P - pivoting vector
%
% NOTES
%   Asserts the size of the input matrix
%   Updating the submatrix was taken directly from the textbook
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [A,p,growth] = pluFact(A,~)

[m,n] = size(A);
assert(m == n,'Please insert square matrix')
p = 1:n;
iniGrowth = max(max(A));
growth = iniGrowth;

for i = 1:n-1
    J = i+1:n;
    if nargin > 1
        %First, find the maximum row to swap with
        [~,loc] = max(abs(A(i:n,i)));
        loc = loc + i-1;

        %Swap row i with row loc and also in p
        A([i,loc],:) = A([loc,i],:);
        p([i,loc]) = p([loc,i]);
    end

    %Now replace the "eliminated" elements with their elimination constant
    A(J,i) = A(J,i) / A(i,i);
    %And then do the elimination on the rest of the matrix
    A(J,J) = A(J,J) - A(J,i) * A(i,J);

    %Identify growth factor
    if max(max(A(J,J))) > growth
        growth = max(max(A));
    end
end
growth = growth/iniGrowth;
end
```