Assignment 4 Solutions

Aaron Cahn
University of Wisconsin-Madison
cahn@cs.wisc.edu

March 26, 2015

1 Solutions

1.1 Question 1

1.1.1 Part A

Let A be a symmetric matrix of dimension m. The entries of A are denoted by a_{ij}.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1j} \\ \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} \end{bmatrix} \tag{1}$$

After one iteration the first row of A will be unchanged and the first column will contain zeros below the first element. The $(m-1)x(m-1)$ sub matrix of A is computed by the equation $a_{ij} = a_{ij} - \frac{a_{i1}}{a_{11}} * a_{1j}$. Since, A is a symmetric matrix $a_{ij} = a_{ji}$. Therefore this submatrix will remain symmetric because the equation to compute a_{ji} is the same as the equation to compute a_{ij} listed above. The next iteration of LU factorization is now operating on this symmetric matrix so we can simply relable the elements of the submatrix is we did in 1 because we can ignore the first row and first column of A.

Therefore, the same argument applies for the second iteration. By induction it can then be shown that after iteration k the $(m-k)x(m-k)$ sub matrix of the $(m-k+1)x(m-k+1$ sub matrix of A is symmetric. Explicitly the computation of the sub matrix is $a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} * a_{kj}$. From this it is clear the $(m-k)x(m-k)$ sub matrix is still symmetric because we have proven above the $(m-k+1)x(m-k+1)$ sub matrix of A after iteration $k-1$ is symmetric (*e.g.,* if we unroll to where $k = 1$). This concludes the proof.

1.1.2 Part B

Listing 1: Matlab Commands

```
function [L,U,t] = lu_sym(U)
    tic
    % get the dimension of the input matrix
    m = size(U);
    % initialize L
    L = eye(m);

    % each row of U
    for k=1:m-1
        % each row below the current row (rows to modify)
        for i=k+1:m
            % create the elimination factor
            % grab the elimination factor from above the diagonal
            % U(k,i) instead of U(i,k)
            L(i,k)=U(k,i)/U(k,k);
            % update the upper triangular part of row i of U
            % start j at i (diagonal element)
            % process each j (from diagonal to end of row)
            for j=i:m
                U(i,j)=U(i,j)-L(i,k)*U(k,j);
            end
        end
    end
    % grab the upper triangular portion of U
    U = triu(U);
    t = toc;
end
```

1.1.3 Part C

The complexity of Gauss elimination is is $O(\frac{m^3}{3})$. Regular gauss elimination works on an entire matrix (*i.e.,* touching all elements of the matrix) during *LU* factorization. However, for a symmetric matrix we have proven in part A that we only need to touch the upper triangular elements of a matrix. This constitutes half of the matrix. Therefore, the complexity of the symmetric *LU* factorization algorithm will be $\frac{1}{2}\frac{m^3}{3}$ or simply $O(\frac{m^3}{6})$.

1.1.4 Part D

Listing 2: Matlab Commands

```
% run on matrices of different sizes
for j=3:6
    % create random symmetric matrices
    B=rand(10*j);
    A=B*B';

    % arrays for time storage
    sym_t=0; reg_t=0;

    % run the experiment 100 times
    for i=1:100
        % symmetric LU factorization
        [L,U,t] = lu_sym(A);
        sym_t(i) = t;

        % Gauss elimination LU factorization
        [L,U,t] = lu_basic(A);
        reg_t(i) = t;
    end
    sym_over_gauss = mean(sym_t)/mean(reg_t)
end
```

Listing 3: Matlab Commands

```
q1_partD

sym_over_gauss = 0.6011

sym_over_gauss = 0.5770

sym_over_gauss = 0.5647

sym_over_gauss = 0.5568
```

It is clear from the results that the ratio between the run time of the symmetric *LU* factorization script over the Gauss elimination *LU* factorization script is converging to $\frac{1}{2}$. This confirms the complexity predicted for symmetric *LU* factorization in part C of this question. Note, experiments were run on symmetric matrices with $m \in \{30, 40, 50, 60\}$.

1.2 Question 2

1.2.1 Part A

This code does the basic Gauss elimination *LU* factorization. It is the same code used in question 1 during the comparison to the symmetric *LU* factorization.

Listing 4: Matlab Commands

```
function [L,U,t] = lu_basic(U)
    tic
    % grab the dimension of U (A)
    m = size(U);
    % initialize L
    L = eye(m);

    % for each row of U
    for k=1:m-1
        % for each row below current (rows to modify)
        for i=k+1:m
            % create the elemination value
            L(i,k)=U(i,k)/U(k,k);
            % update of all elements of row i
            for j=k:m
                U(i,j)=U(i,j)-L(i,k)*U(k,j);
            end
        end
    end
    t = toc;
end
```

1.2.2 Part B

As stated in the question, we simply use Matlab's lu routine to accomplish this algorithm. Note, Matlab's lu routine does partial pivoting.

1.2.3 Part C

This code does complete pivoting before Gauss elimination. The code returns [L,U,P,Q] where P and Q are permutation matrices. The algorithm produces matrices for the equation $PAQ = LU$.

Listing 5: Matlab Commands

```
function [L,U,P,Q] = lu_pivot(U)
    % grab the dimension of U (A)
    m = size(U);
    % pivot matrices
    P=eye(m);Q=eye(m);

    % for each row of U
    for k=1:m-1
        % find the max element in the sub-matrix
        pivot = max(max(abs(U(k:m,k:m))));
        [xinds,yinds] = find(pivot == abs(U(k:m,k:m)));

        % project back to the full matrix U
        x=xinds(1)+(k-1); y=yinds(1)+(k-1);

        % Pivot the rows and columns of U
        U([k,x],:) = U([x,k],:);
        U(:,[k,y]) = U(:,[y,k]);

        % Store the permutations
        P([k,x],:) = P([x,k],:);
        Q(:,[k,y]) = Q(:,[y,k]);

        % for each row below current (rows to modify)
        for i=k+1:m
            % create the elemination value
            U(i,k)=U(i,k)/U(k,k);
            % update of upper diagonal elements of row i
            for j=k+1:m
                U(i,j)=U(i,j)-U(i,k)*U(k,j);
            end
        end
    end
    L=tril(U,-1)+eye(m);
    U=triu(U);
end
```

1.2.4 Part D

The code below is the experimental setup that was run to obtain results for this question.

Listing 6: Matlab Commands

```
A = {};
% Question matrices
A{1}=hilb(25);
A{2}=q4_partA(linspace(0,10,25),24);
A{3}=magic(25);
A{4}=magic(25)*rand(25);
% Random matrices
A{5}=rand(25); A{6}=rand(25);
A{7}=rand(25); A{8}=rand(25);
% Random symmetric matrices
B=rand(25);A{9}=B*B';
B=rand(25);A{10}=B*B';
% x vector
x=(1:25)';
% Run analysis on each matrix in A{}
for i=1:10
    % compute b from our known x
    b=A{i}*x;
    if issymmetric(A{i})
        % Gauss Elimination on symmetric A
        [L,U] = lu_sym(A{i});
    else
        % Gauss Elimination LU factorization
        [L,U] = lu_basic(A{i});
    end
    x1 = inv(U)*(inv(L)*b);
    e1 = norm(x1-x);
    % Matlab's LU factorization
    [L,U,P] = lu(A{i});
    x2 = inv(U)*(inv(L)*(P*b));
    e2 = norm(x2-x);
    % Full pivoting LU factorization
    [L,U,P,Q] = lu_pivot(A{i});
    x3 = Q*(inv(U)*(inv(L)*(P*b)));
    e3 = norm(x3-x);
    results = sprintf('cond: %e\ne1: %e e2: %e e3: %e',
                      cond(A{i}),e1,e2,e3)
end
```

The code below is the output from the experimental setup above and represents the results from the experiment.

Listing 7: Matlab Commands

```
results =
cond: 1.779069e+18
e1: 1.018384e+05 e2: 2.019009e+03 e3: 3.009446e+03
results =
cond: 3.433552e+28
e1: 1.313703e+17 e2: 6.682693e+16 e3: 3.720847e+13
results =
cond: 2.502868e+01
e1: 4.714509e+16 e2: 8.089862e-14 e3: 9.853003e-14
results =
cond: 5.525403e+05
e1: 3.733332e-09 e2: 5.109505e-10 e3: 1.815058e-09
results =
cond: 5.189852e+02
e1: 1.059539e-11 e2: 1.121544e-12 e3: 1.566507e-12
results =
cond: 3.294859e+02
e1: 3.181888e-11 e2: 1.436185e-12 e3: 6.052195e-13
results =
cond: 4.414682e+03
e1: 9.162326e-11 e2: 5.464907e-12 e3: 5.559763e-12
results =
cond: 2.677285e+02
e1: 8.270899e-12 e2: 9.295491e-13 e3: 6.933287e-13
results =
cond: 1.815954e+05
e1: 9.091273e-10 e2: 1.042460e-09 e3: 1.479082e-10
results =
cond: 2.636233e+06
e1: 5.075205e-09 e2: 5.252364e-09 e3: 2.431109e-09
```

From the experimental analysis we can see that the first two matrices are very poorly conditioned. Therefore, there is no hope of obtaining a correct answer even before LU factorization is performed in any fasion. This is to say the conditioning of the original matrix A is so bad there the stability of the algorithms does not matter.

The next issue is that of stability. It is clear from the experiments that the stability of the algorithms improves in the order Gauss Elimination, partial pivoting, and then full pivoting being the most stable. Note, the complexity increases in the same manner (*i.e.,* full pivoting is the most costly algorithm). This is evident from the errors dropping from one algorithm to the next. The

stability does not improve drastically between partial and full pivoting. Note, this is only really relevant for matrices that are not poorly conditioned.

The final issue is that of conditioning and stability on random matrices. It is almost always the case that the conditioning of a random matrix A will be good. This is because the probility of randomly creating to vectors that are linearly dependent is so low, and this increases as the size of the matrix increases. Therefore, the penalty incurred by using a more stable algorithm is generally not worth it for a random matrix.

1.3 Question 3

1.3.1 Part A

Listing 8: Matlab Commands

```
% The matrix to deflate
A = [309,228,-240;60,-117,510;12,6,298]/49;

% The eigen vector (normalized)
v = [6 3 2]';
v1 = v/norm(v);

% The eigen value associated with v1
l1=(A*v)./v;
l1=l1(1);

% Create the Housholder matrix
e1=[1 0 0]';
w = e1-v1;
w1 = w/norm(w);
H = eye(3)-2*w1*w1'

H =

    0.8571    0.4286    0.2857
    0.4286   -0.2857   -0.8571
    0.2857   -0.8571    0.4286

% Deflate A to A1
A1 = H*A*H

A1 =

    7.0000   -0.0000    0.0000
   -0.0000    9.0000   -6.0000
   -0.0000    6.0000   -6.0000
```

As we see above, after we deflate v from A we obtain,

$$A_1 = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 9 & -6 \\ 0 & 6 & -6 \end{bmatrix} \tag{2}$$

1.3.2 Part B

Listing 9: Matlab Commands

```
% The 2x2 deflated matrix
tilA1 = A1(2:end,2:end)

tilA1 =

    9.0000   -6.0000
    6.0000   -6.0000

% The eigen values and eigen vectors
[V,E] = eig(tilA1)

V =

    0.8944    0.4472
    0.4472    0.8944


E =

    6.0000         0
         0   -3.0000
```

As we see above the eigen pairs of $\tilde{A}_1$ are,

$$e_1 = \left(6, \begin{bmatrix} 0.8944 \\ 0.4472 \end{bmatrix}\right), e_2 = \left(-3, \begin{bmatrix} 0.4472 \\ 0.8944 \end{bmatrix}\right) \tag{3}$$

1.3.3 Part C

Listing 10: Matlab Commands

```
% Eigenpair e1
v2 = V(:,1);
l2 = E(1,1);
% Eigenpair e2
v3 = V(:,2);
l3 = E(2,2);

% Reinflate v2 to eigenvector of A
x = A1(1,2:end);
a = dot(x,v2)/(l2-l1);
v2 = [a; v2];
v2 = H*v2

v2 =

    0.5111
   -0.6389
   -0.5750

% Reinflate v3 to eigenvector of A
a = dot(x,v3)/(l3-l1);
v3 = [a; v3];
v3 = H*v3

v3 =

    0.4472
   -0.8944
   -0.0000
```

Therefore, v_1, v_2, v_3 are the eigenvectors of A. They are all normalized and therefore form an eigenbasis of A to $\mathbb{R}^3$. The eigenbasis is thus,

$$\left(\begin{bmatrix} 0.8571 \\ 0.4286 \\ 0.2857 \end{bmatrix}, \begin{bmatrix} 0.5111 \\ -0.6389 \\ -0.5750 \end{bmatrix}, \begin{bmatrix} 0.4472 \\ -0.8944 \\ 0 \end{bmatrix} \right) \tag{4}$$

1.3.4 Part D

We can simply use the eigenbasis from above to form the matrix P. The matrix D is simply the eigenvalues l_1, l_2, l_3 from above. Therefore we have diagonalized A. Note, we use Matlab to compute P^{-1}.

$$P = \begin{bmatrix} 0.8571 & 0.5111 & 0.4472 \\ 0.4286 & -0.6389 & -0.8944 \\ 0.2857 & -0.5750 & 0 \end{bmatrix}, D = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & -3 \end{bmatrix}, P^{-1} = \begin{bmatrix} 0.8571 & 0.4286 & 0.2857 \\ 0.4259 & 0.2130 & -1.5972 \\ 0.1065 & -1.0648 & 1.2778 \end{bmatrix} \quad (5)$$

Listing 11: Matlab Commands

```
P = [v1 v2 v3]
P =
    0.8571    0.5111    0.4472
    0.4286   -0.6389   -0.8944
    0.2857   -0.5750   -0.0000

D = diag([11 12 13])
D =
    7.0000         0         0
         0    6.0000         0
         0         0   -3.0000

inv(P)
ans =
    0.8571    0.4286    0.2857
    0.4259    0.2130   -1.5972
    0.1065   -1.0648    1.2778

% Check P*D*inv(P) == A
P*D*inv(P)
ans =
    6.3061    4.6531   -4.8980
    1.2245   -2.3878   10.4082
    0.2449    0.1224    6.0816

A
A =
    6.3061    4.6531   -4.8980
    1.2245   -2.3878   10.4082
    0.2449    0.1224    6.0816
```