Assignment 4 Solutions

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

March 25, 2015

1 Solutions

1.1 Question 1

1.1.1 Part A

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

1.3 Question 3

1.3.1 Part A

1.3.2 Part B

1.3.3 Part C

1.3.4 Part D