function [M, inds, flag] = PartialPivoting(A) Xavier Rodriguez

%[M, inds, flag] = PartialPivoting(A)

% Uses partial pivoting to solve A

% that is if A is invertible.

%

% M is the LU factors of A

% inds is the indices that is the size of the matrix

% flag is a Boolean that will flag 1 if the matrix is or near singularity

%

% Input: square matrix A

% Output: square matrix U containing the LU factors of A.

M = A;

[r,c] = size(M); % row and column

inds = (1 : r)'; % makes a column for inds

flag = 0; % flag for singularity or near singularity

n = norm(A,1) \* c \* sqrt(eps); % grabs the smallest number near singularity

for k = 1 : (c - 1)

ind = k + 1 :r;

[~, maxind] = max(abs(M(k:r, k))); %grabs the max index

if maxind ~= 1 % partial pivoting

j = maxind + k - 1;

M = swapRow(M, k, j); % swaps row

inds = swapRow(inds,k,j);% swaps inds

end

M(ind,k) = M(ind,k)/M(k,k); % compute mutipliers

for i =ind

M(i,ind) = M(i,ind) - M(i,k) \*M(k,ind); %row reduction

end

end

if min(abs(diag(M))) < n % checks the diaginal for singularity if so

flag = 1; % flag it

end

end

function [A] = swapRow(A, row1, row2)

% [A] = swapRow(A, row1, row2)

% Swaps the row 1 for row 2 in a matrix

A([row2, row1], : ) = A ([row1, row2], :);

end

**Independent Testing for PartialPivoting(A)**

**Test 1**

A = [1 3 2; 2 2 4; 1 9 8];

[M, inds, flag] = PartialPivoting(A)

M =

2.0000 2.0000 4.0000

0.5000 8.0000 6.0000

0.5000 0.2500 -1.5000

inds =

2

3

1

flag =

0

X = lu(A)

X =

2.0000 2.0000 4.0000

0.5000 8.0000 6.0000

0.5000 0.2500 -1.5000

**Test 2**

A = [1 0; 0 0];

[M, inds, flag] = PartialPivoting(A)

M =

1 0

0 0

inds =

1

2

flag =

1

y = lu(A)

y =

1 0

0 0

**Test 3**

A = [1 2 3; 4 5 6; 7 8 9];

[M, inds, flag] = PartialPivoting(A)

M =

7.0000 8.0000 9.0000

0.1429 0.8571 1.7143

0.5714 0.5000 0.0000

inds =

3

1

2

flag =

1

y = lu(A)

y =

7.0000 8.0000 9.0000

0.1429 0.8571 1.7143

0.5714 0.5000 -0.0000

Conclusion: For this program my testing all ran correctly when the matrix is singular or near singularity it correctly throws a flag. This is proven through my tests and then I verified the LU factors, and they are correct as well.

-------------------------------------------------------------------------------------

function [x] = Solves(M, b, inds)

% [x] = Solves(M, b, inds)

% Computes Ax = b using the LU factorization matrix

%

% M is the LU factors of A assume M is invertible

% b is the column vector that we test the size if inds

% inds is the indices that is the size of the matrix

%

% Input: LU matrix M, and a column vector b

% inds is a column vector

% Output: Solution of Ax = b as a column vector x

[r,c] = size(M);

b = b(inds); % swap b row in index

x = b ;

threshold = norm(x) \* length (x) \* sqrt(eps); % Threshold that is near singularity

i = 1;

while abs(x(i)) <= threshold %compares xi to the threshold

i = i + 1;

end

for j = i : r

x(j) = x(j) - M(j, i:(j - 1))\*x(i:(j - 1));% forward-substitution

end

i = r;

while abs(x(i)) <= threshold %compares xi to the threshold

i = i - 1;

end

for j = i : -1: 1

x(j) = (x(j) - M(j, j + 1:c)\*x(j + 1:c)) / M(j,j); %backward substitution

end

end

**Independent Testing for Solves(M, b, inds)**

**Test 1**

A = [1 3 2; 2 2 4; 1 9 8];

b = [1 2 3 4]';

[M, inds]= PartialPivoting(A)

M =

2.0000 2.0000 4.0000

0.5000 8.0000 6.0000

0.5000 0.2500 -1.5000

inds =

2

3

1

[x] = Solves(M, b, inds)

x =

0.3333

0

0.3333

y = linsolve(A, b)

y =

0.3333

0.0000

0.3333

**Test 2**

A = [2 3; 4 2;];

b = [0 2]';

[M, inds] = PartialPivoting(A)

M =

4.0000 2.0000

0.5000 2.0000

inds =

2

1

[x] = Solves(M, b, inds)

x =

0.7500

-0.5000

y = linsolve(A, b)

y =

0.7500

-0.5000

**Test 3**

A = [3 5 4 8; 3 5 1 2; 2 4 5 3; 4 6 2 1];

b = [2 1 0 0]';

[M, inds] = PartialPivoting(A)

M =

4.0000 6.0000 2.0000 1.0000

0.5000 1.0000 4.0000 2.5000

0.7500 0.5000 -2.5000 0

0.7500 0.5000 -0.2000 6.0000

inds =

4

3

2

1

[x] = Solves(M, b, inds)

x =

-0.9167

0.6833

-0.4000

0.3667

y = linsolve(A, b)

y =

-0.9167

0.6833

-0.4000

0.3667

Conclusion: For this program how I tested it was I used my

[M, inds] = PartialPivoting(A) and with that I got my LU factors M and my inds. I then used M and inds in my Solves(M, b, inds), my b I would choose a vector that would cover the case of leading zeros or trailing zeros.

-------------------------------------------------------------------------------------

function [Ainv] = Inverse(A)

%[Ainv] = Inverse(A)

% Uses the matrix A and finds the inverse

% using the LU factorization of A.

% A is a n x n matix

% Input: matrix A

% Output: Inverse of LU = A

[r,c] = size(A);

Ainv = eye(r, c); % creates a blank matrix of zeros

[m, ind] = PartialPivoting(A); % calls my PartialPivoting

for k = 1: r

Ainv(:,k) = Solves(m, Ainv(:, k), ind); % computes the x

end

end

**Independent Testing for Inverse(A)**

**Test 1**

A = [1 3 2; 2 2 4; 1 9 8];

[Ainv] = Inverse(A)

Ainv =

0.8333 0.2500 -0.3333

0.5000 -0.2500 0

-0.6667 0.2500 0.1667

x = inv(A)

x =

0.8333 0.2500 -0.3333

0.5000 -0.2500 0

-0.6667 0.2500 0.1667

**Test 2**

A = [4 6; 2 1];

[Ainv] = Inverse(A)

Ainv =

-0.1250 0.7500

0.2500 -0.5000

x = inv(A)

x =

-0.1250 0.7500

0.2500 -0.5000

**Test 3**

A = [2 2 3 4; 9 4 5 1; 1 1 3 6; 8 7 1 2];

[Ainv] = Inverse(A)

Ainv =

-0.8451 0.1878 0.5117 0.0610

0.9718 -0.2160 -0.6385 0.0798

0.8310 0.0376 -0.4977 -0.1878

-0.4366 -0.0141 0.4366 0.0704

x = inv(A)

x =

-0.8451 0.1878 0.5117 0.0610

0.9718 -0.2160 -0.6385 0.0798

0.8310 0.0376 -0.4977 -0.1878

-0.4366 -0.0141 0.4366 0.0704

Conclusion: For this program I ran my inverse program that pulls from my PartialPivoting and Solves program. With that it gives me my inverted matrix A and then I compared it to the matlab inverse function of A and the results check out.

-------------------------------------------------------------------------------------

(i)

A= [1 0 0 1; -1 1 0 1; -1 -1 1 1; -1 -1 -1 1];

[M] = partialpivotLU(A)

M =

1 0 0 1

-1 1 0 2

-1 -1 1 4

-1 -1 -1 8

A= [1 0 0 0 1; -1 1 0 0 1; -1 -1 1 0 1; -1 -1 -1 1 1; -1 -1 -1 -1 1];

[M] = partialpivotLU(A)

M =

1 0 0 0 1

-1 1 0 0 2

-1 -1 1 0 4

-1 -1 -1 1 8

-1 -1 -1 -1 16

Conclusion: Looking at this problem the main pattern we see is for the largest entry. N being the size of the matrix, for example when n is 4 it is and when n is 5 it is . This checks out because that number we get is the very last entry in the matrix.

-------------------------------------------------------------------------------------

(ii)

A = [ 1 0 0 0; 1 2 0 0; 1 2 3 0; 1 2 3 4];

[M, inds] = partialpivotLU(A)

M =

1 0 0 0

1 2 0 0

1 1 3 0

1 1 1 4

inds =

1

2

3

4

x = lu(A)

x =

1 0 0 0

1 2 0 0

1 1 3 0

1 1 1 4

y = triu(A)

y =

1 0 0 0

0 2 0 0

0 0 3 0

0 0 0 4

Looking at U it will always be a diagonal matrix with diagonals 1 to n.

------------------------------------------------------------------------------------------------------------------------------------------

(iii)

**Test 1**

n = 3;

b = [1 2 3]';

x = [1 0 0]';

A =

1 1 1

2 4 8

3 9 27

y = A \* x

y =

1

2

3

**Test 2**

n = 4;

x = [ 1 0 0 0]';

b = [1 2 3 4]';

A = generateD(n)

A =

1 1 1 1

2 4 8 16

3 9 27 81

4 16 64 256

y = A \* x

y =

1

2

3

4

Conclusion: Since b(i) = I, the b column vector will always have values 1 to n. Therefore, the solve Dnx = b, x needs to be a n by 1 column vector of the first identity column. Meaning, xnX1 vector x has all zeroes with 1 as the first entry. This will zero out all xn where n = 2:n, and keep the value of x1. Therefore, the first column of D will equal b. D \* ei = b.

------------------------------------------------------------------------------------------------------------------------------------------

(iv)

**Test 1**

n = 3;

A = generateQ(n)

A =

0.8571 -0.2857 -0.4286

-0.2857 0.4286 -0.8571

-0.4286 -0.8571 -0.2857

V = [1 2 3]';

[Ainv] = Inverse(A)

Ainv =

0.8571 -0.2857 -0.4286

-0.2857 0.4286 -0.8571

-0.4286 -0.8571 -0.2857

x1 = inv(A)

x1 =

0.8571 -0.2857 -0.4286

-0.2857 0.4286 -0.8571

-0.4286 -0.8571 -0.2857

x2 = A'

x2 =

0.8571 -0.2857 -0.4286

-0.2857 0.4286 -0.8571

-0.4286 -0.8571 -0.2857

x3 = A \* Ainv

x3 =

1.0000 0 -0.0000

0 1.0000 -0.0000

0.0000 0 1.0000

[M, inds] = PartialPivoting(A)

M =

0.8571 -0.2857 -0.4286

-0.5000 -1.0000 -0.5000

-0.3333 -0.3333 -1.1667

inds =

1

3

2

Solved = Solves(M,b,inds)

Solved =

-1.0000

-2.0000

-3.0000

Verify = Ainv \* v

Verify =

-1.0000

-2.0000

-3.0000

**Test 2**

n = 4;

A = generateQ(n)

A =

0.9333 -0.1333 -0.2000 -0.2667

-0.1333 0.7333 -0.4000 -0.5333

-0.2000 -0.4000 0.4000 -0.8000

-0.2667 -0.5333 -0.8000 -0.0667

V = [1 2 3 4]';

[Ainv] = Inverse(A)

Ainv =

0.9333 -0.1333 -0.2000 -0.2667

-0.1333 0.7333 -0.4000 -0.5333

-0.2000 -0.4000 0.4000 -0.8000

-0.2667 -0.5333 -0.8000 -0.0667

x1 = inv(A)

x1 =

0.9333 -0.1333 -0.2000 -0.2667

-0.1333 0.7333 -0.4000 -0.5333

-0.2000 -0.4000 0.4000 -0.8000

-0.2667 -0.5333 -0.8000 -0.0667

x2 = A \* Ainv

x2 =

1.0000 0 0.0000 0.0000

0 1.0000 0.0000 -0.0000

0.0000 0 1.0000 -0.0000

-0.0000 -0.0000 0.0000 1.0000

x3 = A'

x3 =

1.0000 0 0.0000 0.0000

0 1.0000 0.0000 -0.0000

0.0000 0 1.0000 -0.0000

-0.0000 -0.0000 0.0000 1.0000

[M, inds, flag] = PartialPivoting(A)

M =

0.9333 -0.1333 -0.2000 -0.2667

-0.1429 0.7143 -0.4286 -0.5714

-0.2857 -0.8000 -1.2000 -0.6000

-0.2143 -0.6000 -0.0833 -1.2500

inds =

1

2

4

3

Sovled = Solves(M,b,inds)

Solved =

-1.0000

-2.0000

-3.0000

-4.0000

Verify = Ainv \* v

Verify =

-1

-2

-3

-4

Conclusion: I printed out each inverse that was equivalent and compared it to my Ainv and verified that for all my functions . Where and . Using the LU factorization, we solved Ax = v, then we verified our results for x by computing x =.