Programming Assignment 05

Two Methods of Gaussian Elimination

Nick Handelman
3/21/2018

1 Project Description

In this project, I implemented Gaussian elimination with backwards substitution in gaussel.m (Appendix B) and Gaussian elimination with scaled partial pivoting in gausspp.m (Appendix C). Both functions are called from proj5.m (Appendix A), and the results returned from those functions are output in proj5.m.

2 Analysis

2.1 Gaussian Elimination with Backward Substitution

The steps for this method are detailed in section 6.2 of the textbook. My implementation is in gaussel.m (Appendix B) and has the following inputs and outputs:

Inputs:

AA the system of linear equations in augmented matrix form

n the number of equations in the system

Outputs:

AB input matrix AA in triangular form or a message if no unique solution exists

X unique solution to the system of linear equations

add_count the number of additions and subtractions used (not including loop increments)

mult_count the number of multiplications and divisions used

2.1.1 First System of Linear Equations

inputs	outputs
$AA = \begin{bmatrix} 1 & -1 & 2 & -1 & -8 \\ 2 & -2 & 3 & -3 & -20 \\ 1 & 1 & 1 & 0 & -2 \\ 1 & -1 & 4 & 3 & 4 \end{bmatrix}$	$AB = \begin{bmatrix} 1 & -1 & 2 & -1 & -8 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 0 & 2 & 4 \end{bmatrix}$ $X = \begin{bmatrix} -7 & 3 & 2 & 2 \end{bmatrix}$
n = 4	add_count = 32 | mult_count = 42

2.1.2 Second System of Linear Equations

inputs	outputs
$AA = \begin{bmatrix} 30.00 & 591400 & 591700 \\ 5.291 & -6.130 & 46.78 \end{bmatrix}$	$AB = \begin{bmatrix} 3.0000e+01 & 5.9140e+05 & 5.9170e+05 \\ 0.0000e+00 & -1.0431e+05 & -1.0431e+05 \end{bmatrix}$ $X = \begin{bmatrix} 10 & 1 \end{bmatrix}$
n = 2	add_count = 4 | mult_count = 7

2.1.3 Operation Counts

The textbook gives equations (1) and (2) as the number of additions and subtractions (1) and multiplications and divisions (2) in the algorithm.

$$\frac{n^3}{3} + n^2 - \frac{n}{3} \tag{1}$$

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} \tag{2}$$

For n = 4, the number of additions and multiplications is 26 and 36, respectively. For n = 2, the number of additions and multiplications is 3 and 6. However, my implementation used 32 additions and 42 multiplications for n = 4 and 4 additions and 7 multiplications for n = 2. In the addition count, the textbook is not including the subtractions that are used during backwards substitution (see line 72 of gaussel.m). In the multiplication count, the textbook is not including the divisions AA(j, i)/AA(i, i) on line 59 of gaussel.m.

2.2 Gaussian Elimination with Scaled Partial Pivoting

The steps for this method are detailed in section 6.3 of the textbook. My implementation is in gausspp.m (Appendix C) and has the following inputs and outputs:

Inputs:

AA the system of linear equations in augmented matrix form

n the number of equations in the system

Outputs:

AB input matrix AA in triangular form or a message if no unique solution exists

X unique solution to the system of linear equations

add_count the number of additions and subtractions used (not including loop increments)

mult_count the number of multiplications and divisions used

comp_count the number of comparisons used

2.2.1 First System of Linear Equations

inputs	outputs
$AA = \begin{bmatrix} 30.00 & 591400 & 591700 \\ 5.291 & -6.130 & 46.78 \end{bmatrix}$	$AB = \begin{bmatrix} 5.2910e+00 & -6.1300e+00 & 4.6780e+01 \\ 0.0000e+00 & 5.9143e+05 & 5.9143e+05 \end{bmatrix}$ $X = \begin{bmatrix} 10 & 1 \end{bmatrix}$
n = 2	add_count = 4 | mult_count = 10 | comp_count = 3

2.2.2 Second System of Linear Equations

inputs	outputs
$AA = \begin{bmatrix} 2.11000 & -4.21000 & 0.92100 & 2.01000 \\ 4.01000 & 10.20000 & -1.12000 & -3.09000 \\ 1.09000 & 0.98700 & 0.83200 & 4.21000 \end{bmatrix}$	$AB = \begin{bmatrix} 1.09000 & 0.98700 & 0.83200 & 4.21000 \\ 0.00000 & -6.12061 & -0.68957 & -6.13963 \\ 0.00000 & 0.00000 & -4.92092 & -25.16750 \end{bmatrix}$
	$X = \begin{bmatrix} -0.42800 & 0.42690 & 5.11439 \end{bmatrix}$
n = 4	add_count = 14 | mult_count = 26 | comp_count = 9

2.2.3 Operation Counts

The textbook gives equations (1), (3) and (4) as the number of additions and subtractions (1), multiplications and divisions (3) and comparisons (4) in the algorithm.

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} + \frac{n(n+1)}{2} - 1 \tag{3}$$

$$\frac{3}{2}n(n-1) \tag{4}$$

For n = 2, the number of additions, multiplications and comparisons is 3, 8 and 3, respectively. For n = 3, the number of additions, multiplications and comparisons is 11, 22 and 9. However, my implementation used 4 additions and 10 multiplications for n = 2 and 14 additions and 26 multiplications for n = 3. The discrepancy in addition counts is explained in section 2.1.3. In the multiplication count, all of the discrepancy, except one, is explained in section 2.1.3. The for loop (lines 42-69 in gausspp.m) runs one extra iteration to perform the check in lines 48-53. This check ensures that the last row isn't all zero so the backwards substitution won't run into a divide-by-zero issue. There is no discrepancy between the number of comparisons expected by equation (4) and the number of comparisons used in my implementation.

Appendix A proj05.m

```
1;

function gaussel_arithmetic_counts(n, actual_add_count, actual_mult_count)
    book_add_count = (2*n^3 + 3*n^2 - 5*n)/6
    actual_add_count
    printf("\n")
    book_mult_count = (n^3 + 3*n^2 - n)/3
    actual_mult_count
    printf("\n")
endfunction

function gausspp_arithmetic_counts(n, actual_add_count, actual_mult_count, actual_comp_cou
    expected_add_count = (2*n^3 + 3*n^2 - 5*n)/6
    actual_add_count
    printf("\n")
    expected_mult_count = (n^3 + 3*n^2 - n)/3 + (n^2+n)/2 - 1
    actual_mult_count
    printf("\n")
    book_comp_count = 3/2*n*(n-1)
    actual_comp_count
    printf("\n")
endfunction

% Input #1 for Gaussian Elimination with Backward Substitution
printf("Input #1 for Gaussian Elimination with Backward Substitution\n");
n = 4
AA = [1 -1 2 -1 -8; 2 -2 3 -3 -20; 1 1 1 0 -2; 1 -1 4 3 4]

[BB,X, add_count, mult_count] = gaussel(AA, n);
BB
X
gaussel_arithmetic_counts(n, add_count, mult_count);
pause;

% Input #2 for Gaussian Elimination with Backward Substitution
printf("Input #2 for Gaussian Elimination with Backward Substitution\n");
n = 2
AA = [30.00 591400 591700; 5.291 -6.130 46.78]

[BB,X, add_count, mult_count] = gaussel(AA, n);
BB
X
gaussel_arithmetic_counts(n, add_count, mult_count);

pause;

% Input #1 for Gaussian Elimination with Scaled Partial Pivoting
printf("Input #1 for Gaussian Elimination with Scaled Partial Pivoting\n");
n = 2
AA = [30.00 591400 591700; 5.291 -6.130 46.78]

[BB,X, add_count, mult_count, comp_count] = gausspp(AA, n);
```

```
BB
X
gausspp_arithmetic_counts(n, add_count, mult_count, comp_count);

pause;

% Input #2 for Gaussian Elimination with Scaled Partial Pivoting
printf("Input #2 for Gaussian Elimination with Scaled Partial Pivoting\n");
n = 3
AA = [2.11 -4.21 0.921 2.01; 4.01 10.2 -1.12 -3.09; 1.09 0.987 0.832 4.21]

[BB,X, add_count, mult_count, comp_count] = gausspp(AA, n);
BB
X
gausspp_arithmetic_counts(n, add_count, mult_count, comp_count);
```

Appendix B gaussel.m

```
%
% Gaussian Elimination with Backward Substition
%
% INPUT
%  n = number of equations
% AA = augumented matrix (n,n+1)
%       1 <= i <= n
%       1 <= j <= n+1
%
% OUTPUT
% X =  n vector of solutions
% AB = solution or message that no unique solution exists
% add_count = number of additions and subtractions used, not including loop increments
% mult_count = number of multiplications and divisions used
%
function [ AB, X, add_count, mult_count ] = gaussel ( AA, n )

    % initialize return values to zero
    X = zeros(1,n);
    add_count = 0;
    mult_count = 0;

    %  your code goes here
    if(rows(AA) != n || columns(AA) != n+1)
        AB = "Matrix dimensions must be n x n+1";
        return;
    endif

    % perform Gaussian Elimination
    % provided aii != 0, Ej = Ej - (aji/aii)Ei, j = i+1, ... n
    for i=1 : n
        % check for aii equal 0
        if(AA(i,i) == 0)
            % swap row i with a subsequent row that has a nonzero value in column i
            swapped = false;
            j = i + 1;

            % search subsequent rows until nonzero value found in column i
            while j <= n && !swapped
                % row j has a nonzero value in column i so swap rows i and j
                if(AA(j,i) != 0)
                    AA([j i],:) = AA([i j],:);
                    swapped = true;
                endif

                ++j;
            endwhile

            if(!swapped) % all subsequent rows have zero in column i
                AB = strcat(mat2str(AA), "\nAll rows after row: ", num2str(i),
                    ' have zeros in column: ', num2str(i),
                    '. No solution or infinite solutions.');
```

```
                return;
            endif
        endif

        % perform gaussian elimination
        for j=i+1 : n
            AA(j, i:n+1) = AA(j, i:n+1) - AA(j, i)/AA(i, i)*AA(i, i:n+1);
            add_count = add_count + n + 2 - i; % n + 2 - i additions (vector addition)
            % 1 division, n + 2 - i multiplications (vector scalar multiplication)
            mult_count = mult_count + n + 3 - i;
        endfor
    endfor

    % perform backwards subsitution - book algorithm
    % xn = an,n+1/an,n
    % xn-1 = (an-1,n+1 - an-1,n) / an-1,n-1
    for i = n : -1 : 1
        X(i) = AA(i, n+1);
        for j = i+1 : n
            X(i) = X(i) - AA(i, j)*X(j);
            ++add_count;
            ++mult_count;
        endfor
        X(i) = X(i)/AA(i,i);
        ++mult_count;
    endfor

    %  solve complete, copy results into augumented matrix AB
    AB = AA;
endfunction
```

Appendix C gausspp.m

```
%
%  Gaussian Elimination with Scaled Partial Pivoting
%
%  INPUT
%  n = number of equations
%  AA = augumented matrix (n,n+1)
%        1 <= i <= n
%        1 <= j <= n+1
%
%  OUTPUT
%  X =  n vector of solutions
%  AB = solution or message that no unique solution exists
%  add_count = number of additions and subtractions used, not including loop increments
%  mult_count = number of multiplications and divisions used
%  comp_count = number of comparisons used
%
function [ AB, X, add_count, mult_count, comp_count ] = gausspp ( AA, n )

    % initialize return values to zero
    X = zeros(1,n);
    add_count = 0;
    mult_count = 0;
    comp_count = 0;

    %  your code goes here
    if(rows(AA) != n || columns(AA) != n+1)
        AB = "Matrix dimensions must be n x n+1";
        return;
    endif

    % calculate scale factor for each row
    S = max(abs(AA(:, 1:n).'))';
    comp_count = comp_count + n*(n-1);

    % return error message if a scale factor is 0
    if(ismember(0,S))
        AB = 'A row consists of zero coefficients. No solution or infinite solutions.';
        return;
    endif

    % perform Gaussian Elimination with scaled partial pivoting
    for i=1 : n
        %perform scaled partial pivot
        [max_ max_index] = max(abs(AA(i:n,i))./S(i:n));
        mult_count = mult_count + n - i + 1;
        comp_count = comp_count + n - i;

        if(max_ == 0) % rows not linearly independent
            AB = strcat(mat2str(AA), "\nCoefficients in Column: ", num2str(i),
                 ' starting at row: ', num2str(i),
                 ' are all zero. No solution or infinite solutions.');
            return;
```

```
        endif

        % adjust max_index to coincide with the correct row in AA
        max_index = max_index + i - 1;

        if(i != max_index)
            AA([i max_index],:) = AA([max_index i],:);
        endif

        % perform gaussian elimination
        for j=i+1 : n
            AA(j, i:n+1) = AA(j, i:n+1) - AA(j, i)/AA(i, i)*AA(i, i:n+1);
            add_count = add_count + n + 2 - i; % n + 2 - i additions (vector addition)
            % 1 division, n + 2 - i multiplications (vector scalar multiplication)
            mult_count = mult_count + n + 3 - i;
        endfor
    endfor

    % perform backwards subsitution - book algorithm
    % xn = an,n+1/an,n
    % xn-1 = (an-1,n+1 - an-1,n) / an-1,n-1
    for i = n : -1 : 1
        X(i) = AA(i, n+1);
        for j = i+1 : n
            X(i) = X(i) - AA(i, j)*X(j);
            ++add_count;
            ++mult_count;
        endfor
        X(i) = X(i)/AA(i,i);
        ++mult_count;
    endfor

    % solve complete, copy results into augumented matrix AB
    AB = AA;
endfunction
```