14 April 2019

Jakub Tłuczek
FA2

Solving a system of equations $AX = B$, where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, by the Gaussian elimination with complete pivoting

Project №1

Topic №1

1 GECP description

Gaussian elimination with complete pivoting (referred to as GECP) comes in handy, when we want to calculate the X in the $AX = B$ equation and as stated in the title of the project, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$ and $m < n$. GECP, though more complicated, provides a greater stability of an algorithm, as opposed to Gaussian elimination with partial pivoting, or regular Gaussian elimination.

In the GECP, we select the pivot from the whole unsolved part of the matrix. Pivot is an entry in the matrix, which absolute value is the greatest one. Then, we shift columns and rows in such a way, that the pivot is in the upper left corner of unsolved submatrix. Next, we perform row eliminations to zero entries in the column below pivot and move to the next submatrix, minus row and column containing pivot.

Provided we keep track of the columns shifts, we can perform GECP on the left part of augmented matrix $A|B$, and then solve for the elements of our desired X matrix. An example (pivot marked in red):

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -2 & -4 & 3 \\ 0 & 4 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix}$$

Let's create augmented matrix:

$$\left[\begin{array}{ccc|c} 2 & 0 & 1 & 1 \\ -2 & {\color{red}-4} & 3 & 7 \\ 0 & 4 & 1 & 3 \end{array}\right] \sim \left[\begin{array}{ccc|c} 0 & 2 & 1 & 1 \\ {\color{red}-4} & -2 & 3 & 7 \\ 4 & 0 & 1 & 3 \end{array}\right] \sim \left[\begin{array}{ccc|c} {\color{red}-4} & -2 & 3 & 7 \\ 0 & 2 & 1 & 1 \\ 4 & 0 & 1 & 3 \end{array}\right] \sim \left[\begin{array}{ccc|c} -4 & -2 & 3 & 7 \\ 0 & 2 & 1 & 1 \\ 0 & -2 & {\color{red}4} & 10 \end{array}\right] \sim$$

$$\left[\begin{array}{ccc|c} -4 & 3 & -2 & 7 \\ 0 & 1 & 2 & 1 \\ 0 & {\color{red}4} & -2 & 10 \end{array}\right] \sim \left[\begin{array}{ccc|c} -4 & 3 & -2 & 7 \\ 0 & {\color{red}4} & -2 & 10 \\ 0 & 1 & 2 & 1 \end{array}\right] \sim \left[\begin{array}{ccc|c} -4 & 3 & -2 & 7 \\ 0 & 4 & -2 & 10 \\ 0 & 0 & 2.5 & -1.5 \end{array}\right]$$

Then, we solve system of equations, remembering about the column swaps we made. It gives us following result:

$$x = \begin{bmatrix} -0.6 \\ 0.2 \\ 2.2 \end{bmatrix}$$

We can check the result by multiplying A and x and checking if it's really b:

$$\begin{bmatrix} 2 & 0 & 1 \\ -2 & -4 & 3 \\ 0 & 4 & 1 \end{bmatrix} \begin{bmatrix} -0.6 \\ 0.2 \\ 2.2 \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix}$$

As described in "Numerical Linear Algebra" by Trefthen and Bau, the general form of GECP is:

$$PAQ = LU$$

where P and Q are permutation matrices (rows and columns respectively), such that PAQ are equal to product of lower and upper triangluar matrices LU, where U can be used to determine result of our equation.

Using an aforementioned example, the general form for A would look like:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 1 \\ -2 & -4 & 3 \\ 0 & 4 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0.25 & 1 \end{bmatrix} \begin{bmatrix} -4 & 3 & -2 \\ 0 & 4 & -2 \\ 0 & 0 & -2.5 \end{bmatrix}$$

To solve our problem, I will first calculate this general form, but already on an augmented matrix, to modify B and A at once. Then I will use U and modified B to compute entries of X and then insert them into solution in a correct order.

2 Description of a program

Usage and explanation of the code is provided in the comments

```
function [xfin, U, L, P, Q] = gecp(A, B)
% Usage of the gecp function to calculate X from the equation AX=B:
% Function takes two arguments - matrices A and B from the equation AX=B
% The output may vary, depending which results we want to extract from the
% function. We can call it for example:
% X = gecp(A, B)
% which will return just the calculated X matrix, or we can call it even:
% [X, U, L, P, Q] = gecp(A, B)
% which returns X, as well as all matrices from PAQ = LU equation
% error is printed on the output as well
[n, n2] = size(A); % extracts A dimensions
[n1, m] = size(B); % extracts B dimensions
if n~= n2 || n1~=n || m > n % checks whether dimensions are valid
    disp('Dimensions not correct'); % if not returns the function
    return;
end
p = 1:n; % for P and Q evaluation
q = 1:n;
A_aug = [A, B]; % creating augmented matrix
col_per = zeros(n, 1); % vector for keeping track of column swaps
for i = 1 : 1 : n
    col_per(i, 1) = i;
end
for k = 1:n-1
    [maxc, rowindices] = max( abs(A_aug(k:n, k:n)) ); % looking for pivot
    [maxm, colindex] = max(maxc);
    row = rowindices(colindex)+k-1; col = colindex+k-1;
    A_aug( [k, row], : ) = A_aug( [row, k], : ); % row swap
    A_aug( :, [k, col] ) = A_aug( :, [col, k] ); % column swap
    temp_col = col_per(k, 1); % column swap tracking
    col_per(k, 1) = col_per(col, 1);
    col_per(col, 1) = temp_col;
    p( [k, row] ) = p( [row, k] ); % P and Q updates
    q( [k, col] ) = q( [col, k] );
    if A_aug(k,k) == 0 % if pivot is 0 there is no sense to add rows
      break
    end
    A_aug(k+1:n,k) = A_aug(k+1:n,k)/A_aug(k,k); % adding rows
    i = k+1:n;
    j = k+1:n+m; % we don't zero values below pivot to save them for L
    A_aug(i,j) = A_aug(i,j) - A_aug(i,k) * A_aug(k,j);
end
```

```
A = A_aug(1:n, 1:n); % extracting left side (A) of an augmented matrix
B = A_aug(1:n, n+1:n+m); % extracting right side (B) of an augmented matrix
L = tril(A,-1) + eye(n); % extracting L from changed A
U = triu(A); % extracting U from changed A
P = eye(n);
P = P(p,:); % create P matrix based on 'tracking' vector
Q = eye(n);
Q = Q(:,q); % the same as above
% calculating x with entries not on right places:

x = zeros(n,m);
for c = 1 : 1 : m
    res = B(:, c);
    for j = n : -1 : 1
        if(U(j,j)==0)
            error('singular matrix');
        end
        x(j, c) = res(j)/U(j,j);
        res(1:j-1) = res(1:j-1) - U(1:j-1, j)*x(j, c);
    end
end

xfin = zeros(n, m); % correcting it thanks to col_per vector
for i = 1 : 1 : n
    for j = 1 : 1 : m
        xfin(col_per(i), j) = x(i, j);
    end
end
```

3 Numerical examples

In order to compute errors, we use following formulas, where X is our solution of $AX = B$, and Z is the exact solution:

$$error1(relative) = \frac{||X - Z||}{||Z||}$$

$$error2(forward) = \frac{||X - Z||}{||Z||cond(A)}$$

$$error3(backward) = \frac{||B - AX||}{||A||||X||}$$

To check the function I have run following examples:

Example №1 The first one is the very same example I solved „by hand", so to speak.

The matrices are:

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -2 & -4 & 3 \\ 0 & 4 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix}$$

Now I entered following commands on MATLAB shell:

```
>> A = [2 0 1; -2 -4 3; 0 4 1];
>> B = [1; 7; 3];
>> Z = [-0.6; 0.2; 2.2];
>> [X, U, L, P, Q] = gecp(A, B)

X =

   -0.6000
    0.2000
    2.2000


U =

   -4.0000    3.0000   -2.0000
         0    4.0000   -2.0000
         0         0    2.5000


L =

    1.0000         0         0
   -1.0000    1.0000         0
         0    0.2500    1.0000


P =

     0     1     0
     0     0     1
     1     0     0


Q =

     0     0     1
```

```
     1         0         0
     0         1         0

>> cond_A = cond(A)

cond_A =

    2.7572

>> error1=norm(X-Z)/norm(Z)

error1 =

   7.2750e-17

>> error2=error1/cond(A)

error2 =

   2.6386e-17

>> error3=norm(B - A*X)/(norm(A)*norm(X))

error3 =

   6.5418e-17
```

Computed value of X:

$$X = \begin{bmatrix} -0.6 \\ 0.2 \\ 2.2 \end{bmatrix}$$

Example №2 Now let's compute something that involves B and X that are something more complicated than just a simple vector:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 5 & 2 & 3 \\ 9 & 2 & 3 \end{bmatrix}, B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 9 & 3 \end{bmatrix}$$

Calculation performed by our function:

```
>> A = [1 3 4; 5 2 3; 9 2 3];
>> B = [1 4; 2 5; 9 3];
>> Z = [1.75 -0.5; 24.75 -16.5; -18.75 13.5];
>> [X, U, L, P, Q] = gecp(A, B)
```

```
X =

    1.7500   -0.5000
   24.7500  -16.5000
  -18.7500   13.5000


U =

    9.0000    3.0000    2.0000
         0    3.6667    2.7778
         0         0   -0.1212


L =

    1.0000         0         0
    0.1111    1.0000         0
    0.5556    0.3636    1.0000


P =

     0     0     1
     1     0     0
     0     1     0


Q =

     1     0     0
     0     0     1
     0     1     0

>> cond_A = cond(A)

cond_A =

  144.9142

>> error1=norm(X-Z)/norm(Z)

error1 =
```

```
    1.0108e-15

>> error2=error1/cond(A)

error2 =

    6.9748e-18

>> error3=norm(B - A*X)/(norm(A)*norm(X))

error3 =

     0
```

The result in second example is:

$$X = \begin{bmatrix} 1.75 & -0.5 \\ 24.75 & -16.5 \\ -18.75 & 13.5 \end{bmatrix}$$

Example №3 This time we check, if our function return an identity matrix when both matrices on the input would be the same:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 5 & 2 & 3 \\ 9 & 2 & 3 \end{bmatrix}, B = \begin{bmatrix} 1 & 3 & 4 \\ 5 & 2 & 3 \\ 9 & 2 & 3 \end{bmatrix}$$

Code implementation:

```
>> A = [1 3 4; 5 2 3; 9 2 3];
>> B = [1 3 4; 5 2 3; 9 2 3];
>> Z = eye(3);
>> [X, U, L, P, Q] = gecp(A, B)

X =

     1     0     0
     0     1     0
     0     0     1


U =

    9.0000    3.0000    2.0000
         0    3.6667    2.7778
```

```
          0              0     -0.1212


L =

     1.0000              0          0
     0.1111         1.0000          0
     0.5556         0.3636     1.0000


P =

     0     0     1
     1     0     0
     0     1     0


Q =

     1     0     0
     0     0     1
     0     1     0

>> cond_A = cond(A)

cond_A =

   144.9142

>> error1=norm(X-Z)/norm(Z)

error1 =

     0

>> error2=error1/cond(A)

error2 =

     0

>> error3=norm(B - A*X)/(norm(A)*norm(X))

error3 =
```

```
    0
```

Example №4 In the next example, let's see how function behaves when we enter two quite big matrices i.e. Pascal matrix as A, and product of A and magic matrix as B:

```
>> n=10;
>> A = pascal(n);
>> Z = magic(n);
>> B = A * Z;
>> [X, U, L, P, Q] = gecp(A, B)

X =

   92.0000   99.0000    1.0000    8.0000   15.0000   67.0000
74.0000   51.0000   58.0000   40.0000
   98.0000   80.0000    7.0000   14.0000   16.0000   73.0000
55.0000   57.0000   64.0000   41.0000
    4.0000   81.0000   88.0000   20.0000   22.0000   54.0000
56.0000   63.0000   70.0000   47.0000
   85.0000   87.0000   19.0000   21.0000    3.0000   60.0000
62.0000   69.0000   71.0000   28.0000
   86.0000   93.0000   25.0000    2.0000    9.0000   61.0000
68.0000   75.0000   52.0000   34.0000
   17.0000   24.0000   76.0000   83.0000   90.0000   42.0000
49.0000   26.0000   33.0000   65.0000
   23.0000    5.0000   82.0000   89.0000   91.0000   48.0000
30.0000   32.0000   39.0000   66.0000
   79.0000    6.0000   13.0000   95.0000   97.0000   29.0000
31.0000   38.0000   45.0000   72.0000
   10.0000   12.0000   94.0000   96.0000   78.0000   35.0000
37.0000   44.0000   46.0000   53.0000
   11.0000   18.0000  100.0000   77.0000   84.0000   36.0000
43.0000   50.0000   27.0000   59.0000

>> cond_A = cond(A)

cond_A =

    4.1552e+09

>> error1=norm(X-Z)/norm(Z)

error1 =
```

```
    5.0181e-09

>> error2=error1/cond(A)

error2 =

    1.2077e-18

>> error3=norm(B - A*X)/(norm(A)*norm(X))

error3 =

    7.5337e-17
```

Example №5 In the end let's consider seemingly simple case, which turns out to be a tricky one. Let A and B be:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, B = \begin{bmatrix} 4 \\ 9 \end{bmatrix}$$

This is an example which doesn't work using the regular Gaussian elimination. However in this case it works just well:

```
>> A = [0  1;  1  1];
>> B = [4;  9];
>> Z = [5;  4];
>> [X,  U,  L,  P,  Q] = gecp(A,  B)

X =

     5
     4


U =

     1     1
     0     1


L =

     1     0
     0     1
```

```
P =

     0     1
     1     0


Q =

     1     0
     0     1

>> cond_A=cond(A)

cond_A =

    2.6180

>> error1=norm(X-Z)/norm(Z)

error1 =

     0

>> error2=error1/cond(A)

error2 =

     0

>> error3=norm(B - A*X)/(norm(A)*norm(X))

error3 =

     0
```

The result is:

$$X = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

Which after checking turns out to be the correct one.

4 Analysis of the result

Gaussian elimination with complete pivoting provides a very stable and reliable method of solving $AX = B$ equations. As we can see, every example provided us with a correct result, even there where other, faster methods fail.

GECP turns out to be quite sluggish when dealing with matrices of smaller size like in Example №1. As we can check by tic and toc commands on the sixth example, it is significantly slower than MATLAB implementation:

```
>> A = [2 0 1; -2 -4 3; 0 4 1];
>> B = [1; 7; 3];
>> tic; X = gecp(A,B); toc
Elapsed time is 0.004980 seconds.
>> tic; X = A\B; toc
Elapsed time is 0.001383 seconds.
```

However, when we consider Example №4, my function turn out to be remarkably faster:

```
>> n=10;
>> A = pascal(n);
>> Z = magic(n);
>> B = A * Z;
>> tic; X = gecp(A,B); toc
Elapsed time is 0.005824 seconds.
>> tic; X = A\B; toc
Elapsed time is 0.164473 seconds.
```

So as we can see, GECP comes in handy when we want to compute bigger matrices.

Errors in most cases are relatively low, however as we can see in fourth example, both the sizes of matrices, as well as floating point values, make the error reasonably bigger.
As we can see in the table 1, the condition number of matrix A is corelated to the realtive error. The higher the $cond(A)$, the bigger relative error we get, as in the example №4. Since the condition number is a measure how much can output change in a result of input change. Since we are dealing with a quite big matrix, the condition number can be very big as well, which results in high realtive error. On the other hand, in the example №5, $cond(A)$ is so small, that the relative error is negligible. Example №3 is a special case, since we are dealing with two identical matrices, then despite the condition number being the same as in Example №2, the relative error is negligible.

Tablica 1: Cond and error

№	$cond(A)$	relative error
1	2.7572	$7.2750e-17$
2	144.9142	$1.0108e-15$
3	144.9142	0
4	$4.1552e+09$	$5.0181e-09$
5	2.6180	0

Reference

[1] L.N. Trefethen, D. Bau III, Numercial Linear Algebra, 1st edition, SIAM, 1997.