Applied Programming

Gaussian Elimination
Matrix Factorizations
and
Pivoting

More details in: U. Ascher and C. Grief, "A First Course in Numerical Methods", chapters 5,

Gaussian Elimination - Reminder

$$\begin{aligned} 2x_1 + 8x_2 + 6x_3 &= 20 \\ 4x_1 + 2x_2 - 2x_3 &= -2 \\ 3x_1 - x_2 + x_3 &= 11 \end{aligned}$$

matrix form: $\mathbf{A}\,\boldsymbol{x} = \mathbf{b}$

$$\underbrace{\begin{pmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{pmatrix}}_{\mathbf{A}} \times \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} 20 \\ -2 \\ 11 \end{pmatrix}}_{\mathbf{b}}$$

Final Result of GE - Reminder

- $\mathbf{U}\,\boldsymbol{x} = \boldsymbol{c}$

$$\underbrace{\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} 20 \\ -42 \\ 20 \end{bmatrix}}_{\boldsymbol{c}}$$

So: $x_1 = 2$, $x_2 = -1$, $x_3 = 4$

Gaussian Elimination Revisited

- Gaussian ***Elimination*** reduces a *dense system* of ***n*** linear ***equations*** in ***n unknowns***, $\boldsymbol{Ax = b}$ to an ***upper triangular*** *system* $\boldsymbol{Ux = c}$ in $\mathcal{O}(n^3)$
  - The ***solution x***, is obtained by ***back-substitution***

- In many cases we need to solve systems of equations with the same coefficient matrix ***A***, but different right hand values ***b*.**
  - Solve for **U** and reuse it!

- *Gaussian Elimination induces a Lower-Upper factorization* called ***LU*** *decomposition* or ***LU*** *factorization*

Factorization Induced by Elimination

- Applying GE to ***just the coefficient matrix*** **A**
  - No b vector

$$\begin{aligned} 2x_1 + 8x_2 + 6x_3 &= 20 \\ 4x_1 + 2x_2 - 2x_3 &= -2 \\ 3x_1 - \ \ x_2 + \ \ x_3 &= 11 \end{aligned}$$

implies ->

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{bmatrix}$$

- Produces an ***LU*** factorization of ***A***

$$\begin{bmatrix} 2 & 8 & 6 \\ 2 & -14 & -14 \\ 3/2 & 13/14 & 5 \end{bmatrix}$$

What is **L** ?

A=LU

Factorization Induced by Elimination

$$\begin{bmatrix} 2 & 8 & 6 \\ 2 & -14 & -14 \\ 3/2 & 13/14 & 5 \end{bmatrix}$$

- The ***lower triangular*** matrix L is formed using the multipliers stored "in-place" and adding a diagonal of ones as shown below

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}$$

When working with LU factorizations verify that A=LU

L matrix insight

- The ***lower triangular*** matrix ***L*** contains all the multipliers used to manipulate the original "A" and "b" matrices .
  - If we are give a new "b" matrix then we can apply all the multipliers used in the original A matrix solution!
  - **We don't have to recalculate the A matrix**
- The L matrix is upper triangular
  - Easy to solve!
  - Similar to a lower triangular matrix

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix}$$

Solving Equations using ***LU Fact.***

Algorithm to Solve the system $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$

Input: ***A, b*** Output: ***L, U, x***

1. ***Apply GE to A*** (**not [A b]**) to obtain ***L, U***

$$\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U} \quad \Rightarrow \quad \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{L}\,(\,\boldsymbol{U}\,\boldsymbol{x}\,) = \boldsymbol{b} \quad \text{but } \boldsymbol{U}\,\boldsymbol{x} = \boldsymbol{c}$$

so $\boldsymbol{L}\,\boldsymbol{c} = \boldsymbol{b}$

2. Solve for ***x*** in ***two steps***: (instead of one)
   a. Use *forward substitution* to solve $\boldsymbol{L}\,\boldsymbol{c} = \boldsymbol{b}$ for ***c***
   b. Use ***back substitution*** to solve $\boldsymbol{U}\,\boldsymbol{x} = \boldsymbol{c}$ for ***x***

*It appears that this is more complex than applying GE to the augmented matrix **G**=[**A b**] but in fact it has the same complexity*

Reminder: Back substitution

- After Gaussian Elimination

$$\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ -42 \\ 20 \end{bmatrix}$$

- Solve for x_3, then x_2, and then x_1

So: $x_1 = 2$, $x_2 = -1$, $x_3 = 4$

Forward Substitution

- Same as back substitution but using the L matrix
  - ***Starting at the top***

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 20 \\ -2 \\ 11 \end{bmatrix}$$

$$L \qquad c \qquad = \qquad b$$

- So $\qquad c_1 = 20$
- $2(20)+1C_2 = -2 \;\; -> \;\; 40 + c_2 = -2 => \qquad c_2 = -42$
- $3/2(20) + 13/14(-42) + c_3 = 11 => -9 + c_3 = 11 => \;\; c_3 = 20$

L Summary

- Applying the L matrix to the original "b" vector results in "c", the same final "b" vector after classical Gaussian
- Applying the "L" matrix $$\begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 20 \\ -42 \\ 20 \end{bmatrix}$$
- Using Gaussian elimination $$\underbrace{\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 20 \\ -42 \\ 20 \end{bmatrix}}_{c}$$
- Conclusion:

We can solve a Gaussian system once, record L and then solve for multiple values of "b"

In–place Forward substitution

- Assumes the data is in lower form WITH an assumed 1 diagonal

- Process columns left to right, skip the last column

  – Process rows starting at the current column+1

    - Subtract the matrix entry multiplied by the current b column entry from the b row entry

Algorithm FS.1 (Similar to bsubs)

- **Forward substitution** (solving $\mathbf{L}\boldsymbol{c} = \boldsymbol{b}$)

  Preconditions: ($\mathbf{L}$ $n \times n$ lower triangular)

  (vectorized "in-place" pseudo-code)

```
% j is column index of L, j=1…n
for j= 1:n-1                % iterate over columns
    b(j) = b(j)/L (j,j)
    rows=j+1:n
    b(rows) = b(rows) –b(j)L(rows,j)
end
b(n) = b(n)/L(n,n)
```

$$\boxed{FLOPs = \mathcal{O}\left(n^2\right)}$$

In this algorithm, the *solution* **c** *overwrites* ***b***. It works for arbitrary lower triangular matrices, e.g., it does not assume ones on diagonal of *L*

Solve for x

- $\boldsymbol{U}\,\boldsymbol{x} = \boldsymbol{c}$

$$\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 20 \\ -42 \\ 20 \end{bmatrix}$$

$$\text{U} \qquad \text{x} \quad = \quad \text{c}$$

- Using back substitution
- $x_3 = 4$
- $-14x_2 - 14(4) = -42 \;\; \text{->} \;\; -14x_2 - 56 = -42 \Rightarrow \;\; x_2 = -1$
- $2x_1 + 8(-1) + 6(4) = 20 \Rightarrow 2x_1 + 16 = 20 \Rightarrow \;\; x_1 = 2$

The right Answer!

Example: ***LU Factorization, new b***

- Given: ***(from before)***

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}$$

- Solve ***b=[1; 2; 3]*** NOT the original b = [20; -2; 11]

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

L c = b

Example: ***LU Factorization***

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

L c = b

So: c = 1, 0, 1.5

$$\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1.5 \end{bmatrix}$$

U x = c

So: x = .8, -0.3, .3

We solved for a new "b" without any Gaussian mathematics, saving CPU time

Verify using Matlab

- verify that ***A=LU***

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}$$

- Now solve ***A x=b*** but for ***b=[1; 2; 3]***
    - NOT the original b = [20; -2; 11]

1. Solve **L** ***c*** **=** ***b*** for ***c*** (*forward substitution)*
2. Solve **U** ***x*** **=** ***c*** for ***x*** (*back substitution)*

Always verify your solutions, e.g. check if Ax=b or Ax-b=0

Example: LU Factorization

- "Solution" (Using Matlab/Octave)

```
% ex1_lu –LU factorization induced by GE

%% LU factors of A
A=[2 8 6;4 2 -2;3 -1 1];
L=[1 0 0;2  1 0;3/2 13/14 1];
U=[2 8 6;0 -14 -14;0 0 5];

%% Verify that A=LU
A-L*U

%% Solve Ax=b using LU factors
b = [1;2;3];
c=fsubs(L,b) % instead of using c=inv(L)*b
x=bsubs(U,c) % instead of using x=inv(U)*c

%% Verify solution
A*x-b
```

Example: LU Factorization

```
Verify that A=LU - result should be very close to zero (4.4409e-16)
ans =    0.0000e+00   0.0000e+00   0.0000e+00
         0.0000e+00   0.0000e+00   0.0000e+00
         0.0000e+00   4.4409e-16   4.4409e-16
c =  1.00000
     0.00000
     1.50000
x =  0.80000
    -0.30000
     0.30000

Verify that A*x-b - result should be very close to zero (4.4409e-16)
ans =   0
        0
        0
```

LU Summary

- Gaussian Elimination can be used to simplify a series of equations

- LU factorization can solve the equations for different data values “b”
  - Assumes the order of the “A” matrix did not change
  - If we rearrange the “A” matrix for some reason, then we will need to rearrange any “b” vector

Gaussian Elimination Failure

Solve

$$\begin{bmatrix} 2 & 8 & 6 \\ 4 & 16 & -2 \\ 3 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ -2 \\ 11 \end{bmatrix}$$

After pass one:

$$\begin{bmatrix} 2 & 8 & 6 \\ 0 & 0 & -14 \\ 0 & -13 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ -42 \\ -19 \end{bmatrix}$$

- New pivot is 0 ! – cannot continue !
- Solution: ***Interchange eqs. 2 and 3***

Pivoting & Gaussian Elimination Failure

After **interchanging equations** 2 and 3

$$\begin{bmatrix} 2 & 8 & 6 \\ 0 & -13 & -8 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ -19 \\ -42 \end{bmatrix}$$

After the row swap (interchange)

- New *pivot* is now -*13*
- ***Ordering of variables (x_1, x_2, x_3) is not affected***
  - ***Notice: "b" vector order changed!***
  - We need to record the row exchanges for future LU factorizations!

Swapping rows *allowed us to* ***complete the Elimination Process***

Partial Pivoting

- Interchanging rows during Gaussian Elimination is called ***pivoting***
- ***Partial pivoting*** is a strategy to choose the pivot for stability of the algorithm
- Pivoting is necessary for two reasons:
  1. To ***avoid zero pivots*** (and thus ***division by zero***)
  2. To ***prevent overflow*** (caused by ***division by elements near zero***) thus avoiding inaccurate results.

Partial Pivoting Strategy

Choose as k^{th} pivot the element of largest magnitude (in the k^{th} sub column)

GE with Partial Pivoting

- Use GE with partial pivoting to solve

$$\begin{bmatrix} -20 & 55 & -10 \\ -10 & -10 & 50 \\ 30 & -20 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 200 \\ 0 \end{bmatrix}$$

- Form Augmented Matrix ***G=[A b]***

$$\begin{bmatrix} -20 & 55 & -10 & 0 \\ -10 & -10 & 50 & 200 \\ 30 & -20 & -10 & 0 \end{bmatrix}$$

GE with Partial Pivoting

Pass 1: **Find pivot** (largest **absolute value**)

Original row order: r =[1 2 3]

$$\begin{bmatrix} -20 & 55 & -10 & 0 \\ -10 & -10 & 50 & 200 \\ 30 & -20 & -10 & 0 \end{bmatrix}$$

Perform Pivoting $\rightarrow$ **swap rows** 1 and 3

Keep track of swaps: **r =[1 2 3] → r =[3 2 1]**

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -10 & -10 & 50 & 200 \\ -20 & 55 & -10 & 0 \end{bmatrix}$$

Use index vector to keep track of swaps.

GE with Partial Pivoting

Perform Elimination on new submatrix

$$\begin{bmatrix} 30 & -20 & -10 & 0 \\ -10 & -10 & 50 & 200 \\ -20 & 55 & -10 & 0 \end{bmatrix}$$

Memory snapshot at **end of pass 1**

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{1}{3} & -\frac{50}{3} & \frac{140}{3} & 200 \\ -\frac{2}{3} & \frac{125}{3} & -\frac{50}{3} & 0 \end{bmatrix}$$

r =[3 2 1]

GE with Partial Pivoting

Pass 2: Find pivot (largest **absolute value**)

current row order: r =[3 2 1]

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{1}{3} & -\frac{50}{3} & \frac{140}{3} & 200 \\ -\frac{2}{3} & \frac{125}{3} & -\frac{50}{3} & 0 \end{bmatrix}$$

Perform Pivoting $\rightarrow$ swap rows 2 and 3

***Keep track of swaps*: r =[3 2 1] → r =[3 1 2]**

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{2}{3} & \boldsymbol{\frac{125}{3}} & -\frac{50}{3} & 0 \\ -\frac{1}{3} & -\frac{50}{3} & \frac{140}{3} & 200 \end{bmatrix}$$

GE with Partial Pivoting

Perform Elimination on new ***submatrix***

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{2}{3} & \boldsymbol{\frac{125}{3}} & -\frac{50}{3} & 0 \\ -\frac{1}{3} & -\frac{50}{3} & \frac{140}{3} & 200 \end{bmatrix}$$

Memory snapshot at **end of pass 2**

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{2}{3} & \boldsymbol{\frac{125}{3}} & -\frac{50}{3} & 0 \\ -\frac{1}{3} & -\frac{2}{5} & 40 & 200 \end{bmatrix}$$

r =[3 1 2]

GE with Partial Pivoting

$$\begin{bmatrix} \mathbf{30} & -20 & -10 & 0 \\ -\frac{2}{3} & \mathbf{\frac{125}{3}} & -\frac{50}{3} & 0 \\ -\frac{1}{3} & -\frac{2}{5} & 40 & 200 \end{bmatrix}$$

r =[3 1 2]

With **partial pivoting** all the **multipliers** are **bounded by 1**

Finally find ***x*** by "back substitution"

$$\begin{bmatrix} \mathbf{30} & -20 & -10 \\ & \mathbf{\frac{125}{3}} & -\frac{50}{3} \\ & & 40 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 200 \end{bmatrix}$$

GEPP - Pseudo code

- Initialize the p vector to 0, 1, 2…
- Process all the rows -1
  - Find largest pivot point by scanning the remaining rows in the current column.
    - Save the largest pivot AND the index
  - Check for zero pivot, error out if necessary
  - Swap the largest pivot point row and adjust the permutation vector
    - Store scaling factors in place
      - Adjust rest of the row
- Check if final diagonal value is a 0, error out if necessary

Algorithm GEPP.1

- Gaussian Elimination with Partial Pivoting ($n \times n$ **A** matrix)
  (vectorized "in-place" pseudo-code)

```
% Initialization of working matrix (G) and pivot vector
 G = [A b]; p=1:n
% k is "pass" index, j=1…n-1
for k= 1: n-1          % pass loops
   % Find pivot and swap columns (q is index of pivot found)
    q=max( find( abs(G(k:n,k) == max(abs(G(k:n,k)) ) + k-1
    G(k,1:n)↔G(q,1:n)          % Swap rows
    p(k)↔ r(q)                 % Update pivot vector r (swap indices)
    pivot = G(k,k)             % set current pivot
  % Perform "regular" Elimination step
    rows = k+1:n               % row index set of entries below pivot
    cols = k+1:n+1  % col index set of entries right of pivot
   % Scale all entries below k-th pivot
    G(rows,k) = G(rows,k) /pivot
   % Apply elimination to compete k-th pass
    G(rows,cols) = G(rows,cols)- G(rows,k) G(k,cols)
end
```

GE with Partial Pivoting

- At each pass a ***new pivot*** is chosen as the element of ***largest magnitude***.
- This choice defines a ***row interchange*** (keep track in index vector *r*)
- Perform ***regular Elimination*** after "row interchange"
- Repeat until finished

Fact: *Gaussian Elimination with Partial Pivoting (GEPP)* induces a *Permuted LU factorization* (***PLU*** factorization)

GEPP and PLU Factorization

- Since partial pivoting just rearranges the equations, GEPP induces a ***PLU Factorization of the Coefficient Matrix***

$$PA = LU$$

- The matrix ***P*** is called a ***permutation matrix***. It can be obtained by applying the same sequence of row swaps (encoded in r) to the identity matrix (examples in next slide)

Permutation Matrices

- A permutation matrix P(*r*) is the matrix obtained from the identity matrix after *performing the row interchanges* encoded in *r*
- ***Examples:***
  - r=[3 1 2], $P(\begin{bmatrix}3 & 1 & 2\end{bmatrix}) = \begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$
  - r=[3 2 1], $P(\begin{bmatrix}3 & 2 & 1\end{bmatrix}) = \begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}$
- We **won't** use this in C!

Permutation Matrix Secrets

- In “C” a permutation matrix just results in swapping array rows

$$P(\begin{bmatrix} 3 & 1 & 2 \end{bmatrix}) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$ (math is origin 1)

- ***Example:*** r=[2 0 1] (C is origin 0)

  New[0] = Old[2];
  New[1] = Old[0];
  New[2] = Old[1];

C Permutation Mapping

- P= [3 1 2] b= [10 200 50]

  Entry= [1 2 3] Entry[1 2 3]

  – Entry 3 moves to position 1

    Pb= [50 . .]

  – Entry 1 moves to position 2

    Pb= [50 10 .]

  – Entry 2 moves to position 3

- Permutated vector Pb= [50 10 200]

GEPP and PLU Factorization

- Write the PLU matrices corresponding to:

$$\begin{bmatrix} 30 & -20 & -10 & 0 \\ -\dfrac{2}{3} & \dfrac{125}{3} & -\dfrac{50}{3} & 0 \\ -\dfrac{1}{3} & -\dfrac{2}{5} & 40 & 200 \end{bmatrix}$$

r =[3 1 2]

$$L=\begin{bmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ -\frac{1}{3} & -\frac{2}{5} & 1 \end{bmatrix} \quad U=\begin{bmatrix} 30 & -20 & -10 \\ 0 & \frac{125}{3} & -\frac{50}{3} \\ 0 & 0 & 40 \end{bmatrix} \quad P=\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- $\boldsymbol{PA=LU}$

Partial Pivoting Example

- Given: $\begin{bmatrix} -20 & 55 & -10 \\ -10 & -10 & 50 \\ 30 & -20 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 200 \\ 0 \end{bmatrix}$

- And GEPP $\begin{bmatrix} 30 & -20 & -10 \\ -\frac{2}{3} & \frac{125}{3} & -\frac{50}{3} \\ -\frac{1}{3} & -\frac{2}{5} & 40 \end{bmatrix}$

  r =[3 1 2]

- Find **x** given "b" $\begin{bmatrix} 10 \\ 200 \\ 0 \end{bmatrix}$

Pivoting is Row exchange

- From GEPP $L=\begin{bmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ -\frac{1}{3} & -\frac{2}{5} & 1 \end{bmatrix}$ $U=\begin{bmatrix} 30 & -20 & -10 \\ 0 & \frac{125}{3} & -\frac{50}{3} \\ 0 & 0 & 40 \end{bmatrix}$

- Use r =[3 1 2] to rearrange "b"

  [3 1 2] (map) $\begin{bmatrix} 10 \\ 200 \\ 0 \end{bmatrix}$ to $\begin{bmatrix} 0 \\ 10 \\ 200 \end{bmatrix}$

  [1 2 3]

- Solve using LU
  - Lc=b implies c = [0 10 204]
  - Ux=c implies x = [3.22 2.28 5.1] (right answer)

Example

```
% ex2_palu –LU factorization induced by GE tracking P

A=[-20 55 -10; -10 -10 50; 30 -20 -10];
L=[1 0 0; -2/3 1 0; -1/3 -2/5 1];
U=[30 -20 -10; 0 125/3 -50/3; 0 0 40];
P=[0 0 1; 1 0 0; 0 1 0];

disp('should be ~zero')
P*A-L*U
```

```
should be ~zero
ans =
   0.0000e+00   0.0000e+00   0.0000e+00
   0.0000e+00   0.0000e+00   1.7764e-15
   0.0000e+00   0.0000e+00   0.0000e+00
```

Solving Equations from *PLU*

Solve a system of linear ***equations for different b's***

$$\mathbf{A}\,x = b_1,\ \mathbf{A}\,x = b_2,\ \ldots$$

Solution: Use ***PLU*** factorization

- Apply ***GE with pivoting*** to **A** (not **[A b]**) to obtain the factors ***L, U*** *and* ***P***

$$P\,\mathbf{A} = L\,U \quad \Rightarrow \quad \mathbf{P}\,\mathbf{A}\,x = L\,(\,U\,x\,) = P\,b$$

- Solve for ***x*** in two steps (forward substitution and back substitution) replacing the coefficient vector ***b*** for ***P b*** (that is, permuting its entries)

In practice ***P is never found explicitly*** *and the vector* ***Pb*** *is obtained by* ***indexing***

Application: Solving Matrix Equations

Solve the matrix equation: $\mathbf{A}\,\boldsymbol{X} = \boldsymbol{B}$

Solution: Use PLU factorization

- Apply GE with pivoting to **A** to obtain the factors **L, U** (and **P**)

$$\boldsymbol{P}\,\mathbf{A} = \boldsymbol{L}\,\boldsymbol{U} \quad \Rightarrow \quad \mathbf{P\,A\,X} = \boldsymbol{L}\,(\,\boldsymbol{U}\,\boldsymbol{X}\,) = \boldsymbol{P}\boldsymbol{b}$$

- Find ***a column at a time***: x_i=*X(:,i)*, b_i=*B(:,i)*,
- Find x_i in two steps:

  1. Use *forward substitution to* solve $\mathbf{L}\, c_i = \mathbf{P} b_i$ for c_i
  2. Use *back substitution to* solve $\mathbf{U}\, x_i = c_i$ for x_i

*We can solve **matrix equations** by solving a **sequence of linear equations** with different right hand sides*

Application: the Inverse of a Matrix

Problem: Find the Inverse of an *nxn* matrix **A**

Hint: *Formulate the problem as a set of linear equation with different right hand sides*

Solution: If **A** is invertible its inverse ***X*** solves **A *X* = I**

- Solve **A *X* = I** a column at a time:
  - Find *PLU* factorization of A => PA=LU and note that

    PA X = PI => LU X = P $\mathcal{O}(n^3)$
  - For i=1 to n
    - **Solve *L* c_i = *P(:,i)*** $\mathcal{O}(n^2)$
    - **Solve *U X(:,i)* = c_i** $\mathcal{O}(n^2)$

Summary: Practical G. Elimination

- Partial Pivoting is the most common pivoting strategy. It involves "switching rows".

- Theoretically ***partial pivoting*** does not guarantee the "stability" of Gaussian Elimination, that is, for some (***ill-posed***) problems it may give ***inaccurate results***

- In practice, the ***probability*** of getting inaccurate results with GEPP is ***very small*** (**but not zero**)

- If GEPP fails to give accurate results need to use other pivoting strategies

Summary: Practical G. Elimination

- When applying ***partial pivoting*** (correctly) ***all the multipliers*** stored in the sub-diagonals of the ***L*** matrix will have ***magnitude less or equal to one***
  - ***Example:***

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{bmatrix}$$

L factors of A

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix}$$

(a) without pivoting

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 3/4 & -5/14 & 1 \end{bmatrix}$$

(b) with partial pivoting

This increases the robustness of this algorithm to round-off errors.

Reference: Practical G. Elimination

- Other pivoting strategies (not discussed here) with guaranteed "stability" include ***scaled pivoting*** and ***total pivoting*** (switching rows and columns). Their overhead does not justify their use.

- From a practical point of view, ***partial pivoting is almost always sufficient.***

- In practice ***pivoting*** is used to ***avoid singularities*** (zero pivots) ***and reduce error propagation*** (small pivots) during Gaussian Elimination.

Problem 1

Represent the following in **Ax=b** form

$$
\begin{aligned}
2x_1 + 8x_2 + 6x_3 &= 20 \\
4x_1 + 2x_2 - 2x_3 &= -2 \\
3x_1 - x_2 + x_3 &= 11
\end{aligned}
$$

$$
\underbrace{\begin{pmatrix} 2 & 8 & 6 \\ 4 & 2 & -2 \\ 3 & -1 & 1 \end{pmatrix}}_{\mathbf{A}} \times \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 20 \\ -2 \\ 11 \end{pmatrix}}_{\mathbf{b}}
$$

Problem 2

- Convert the following to LU form:

$$G=\begin{bmatrix} 2 & 8 & 6 & 1 \\ 2 & -14 & -14 & 2 \\ 3/2 & 13/14 & 5 & 3 \end{bmatrix}$$

$$L=\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3/2 & 13/14 & 1 \end{bmatrix} \quad U=\begin{bmatrix} 2 & 8 & 6 \\ 0 & -14 & -14 \\ 0 & 0 & 5 \end{bmatrix}$$

Problem 3

- Create a mathematical permutation “matrix” from an origin one permutation vector.

  r =[3 1 2]

- The vector digit indicates the placement of the “1”

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Problem 4

- Give a "b" vector of $\begin{bmatrix} 10 \\ 200 \\ 0 \end{bmatrix}$ and an origin ONE permutation vector of: r =[3 1 2]
  - Find the resulting "b" vector

- Use "C Permutation mapping" to rearrange "b"

  [3 1 2] 3 moves to 1, 1 moves to 2, 2 moves to 3

  [1 2 3]

- Answer: b = $\begin{bmatrix} 10 \\ 200 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 10 \\ 200 \end{bmatrix}$

Problem 4

- Why is LU factorization used in Gaussian elimination?

- LU factorization is used to solve the Ax=b equation for multiple sets of “b” without having to re-execute the entire Gaussian elimination process.
- **We solve a Gaussian system once, record L and then solve for multiple values of “b”**

Problem 5

- Why use partial pivoting in Gaussian elimination?

- It avoids division by zero
  - And tends to reduce overall errors

Problem 6

- What are L & U and the general family of LU equations?
  - L is "lower matrix only", made of Gaussian divisors, with a diagonal of 1
  - U is "upper matrix only" after Gaussian simplification
  - $Ax = b$ - General matrix solution
  - $Ux = c$ - Upper matrix equation
  - $Lc = b$ - Intermediate, partial solution
  - $A = LU$ - Identity

Problem 7

You are given the following

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 4 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 3 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & -2 \end{bmatrix}$$

Find $Ax = b$ for $b = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$

without finding A explicitly.

Formulas: Ax=b, Ux=c, A=LU (no pivoting permutation's applied), PA = LU (with pivoting)

Problem 7

$Ax = b \Rightarrow LUx = Pb.$

(in the generation of L and U we used pivoting so we have to pivot b too)

Convert the P matrix to $P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ a "p" vector by inspection:

$$P = [1\ 3\ 2]$$

Given $b = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$ so $Pb = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$

Problem 7

Ax = b => LUx = Pb.

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

L c = Pb

Solve using forward substitution

$$\begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

Problem 7

$$\begin{bmatrix} 3 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

$$U \qquad x = c$$

Solve using backward substitution

$$\begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1 \\ -1 \end{bmatrix}$$

Appendix

Rdnparse.c

```
/************************************************
 * rndparse.c - Read and parse a text file
 ************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NUMARGS 2
#define BUFSIZE 4096

int main (int argc, char *argv[])
{
   char sep[] = " ";    /* parsing separator is space */
   char *p,*buffer;     /* buffer and pointer to scan buffer */
   FILE *ifp;           /* source file */
   unsigned buflen;     /* length of line read */
   unsigned lineno;     /* length of line read */
   unsigned count;      /* coune entries read */
   double x;            /* to hold number read */

   /* first check to see if have required number for argc */
   /* if not, error has occurred, don't do anything else */
   if (NUMARGS != argc)
   {
     printf("%s was expecting one argument\n",argv[0]);
     printf("usage: %s file_to_parse\n",argv[0]);
     printf("    file should include lines of numbers, separated by spaces\n");
   }
   else {
     /* correct number of arguments used, try to open both files */
      buffer = (char*) malloc(BUFSIZE);     /* allocate buffer */
      ifp = fopen (argv[1], "r");           /* open file for reading */
```

```
if (ifp && buffer) {                    /* if successful proceed */
       /* read file line by line */
        lineno=0;
        while ( fgets(buffer, BUFSIZE, ifp) != NULL ) {
          /* defensive/secure programming */
          buflen = strlen(buffer);
          if ( 0 == buflen || '\n' != buffer[buflen-1]){
            printf ("ERROR: Wrong buffer !!\n");
            return 1;
          }

          /* print line read */
          lineno++;
          printf("l%2d: %s",lineno,buffer);

          /* parse line */
          count=0;
          p = strtok(buffer,sep);
         while ( p != NULL) {
           ++count;                /* update counter */
           x = atof(p);            /* Convert the string to double */
           printf("% 10.4f", x);
           p = strtok(NULL, sep); /* find next number */
         }
         putchar('\n');
         printf("Number of entries: %d\n",count);
        }
        fclose (ifp);
     }
     else  /* otherwise, there was a problem */
       printf ("ERROR: Could not open file | buffer not alloc\n");
   }

   return 0;
}
```

Use_qsort.c

```
/* Applied Programming Examples: sorting.c
 * Uses qsort() to sort an array of random doubles
 * Use compiler directive -DN=size to change the size of the array
 * Reference: A. Kelley and I Pohl "A book on C" 4th Ed.
 * Revised: 3/31/2015 (JCCK)
 */
 #include <stdio.h>
 #include <stdlib.h> /* for qsort()    */
 #include <time.h>   /* to seed rand() */

/* Size of array to be sorted */
 #ifndef N
   #define N   13
 #endif
/* Verbatim flag */
 #ifndef VERB
   #define VERB 0
 #endif

 /* Function prototypes */
 int cmpdbl(const void *p1,const void *p2); /* for qsort() */
 void fill_array(double *a, int n,int verb);
 void print_array(double *a, int n);

/*
 Initialize an array of doubles of size N, with random numbers
 between -50 and 50, sort it and print it
*/
 int main(void) {
    double darray[N];
    int verb=-1;
```

```
verb=(VERB ? 1 : 0);

    fill_array(darray , N, verb);
     printf("Before Sorting\n");
     print_array(darray , N);
    qsort(darray, N, sizeof(double), cmpdbl);
     printf("\nAfter Sorting\n");
     print_array(darray , N);
    return 0;
}
int cmpdbl(const void *p1, const void *p2) {
  const double *p = p1;
  const double *q = p2;
  double             diff = *p - *q;
  /* return -1 - The element pointed to by p1 goes before the element pointed to by p2
     return +1 - The element pointed to by p2 goes before the element pointed to by p1
     return  0 - The element pointed to by p1 and p2 are equivalent (equal)           */
  return ((diff>=0.0) ? ((diff >0.0) ? -1:0 ) : +1 );
}
void fill_array(double *a, int n,int verb) {
  int i;
  if (verb) {
    printf("filling array with %d random numbers\n",N);
   }
  srand(time(NULL)); /* seed */
  for( i=0 ; i<n ; ++i)
    a[i] = (rand() % 1001) /10.0 - 50.0;
}
void print_array(double *a, int n) {
  int i;
  for( i=0 ; i<n ; ++i) {
    if (i % 6 == 0) { printf("\n");}
    printf("% 10.1f", a[i]);
  }
  printf("\n");
}
```