Approximation of $f'(x)$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Pay attention to **Numerical Cancellation:** when h is small, $f(x+h) - f(x) \to 0$ Re-formula the formula to avoid numerical cancellation.

Solve Linear System: $AX = b$

GENP:

```
1 function x = genpMyVersion(A,b)
2 % genp.m    Gauss elimination with no pivoting
3 %
4 % input:  A is an n x n nonsingular matrix
5 %         b is an n x 1 vector
6 % output: x is the solution of Ax=b.
7 %
8 n = length(b);
9
10 for k = 1:n-1
11     for i = k+1:n
12         multi = A(i, k)/A(k, k);
13         A(i, k+1 : n) = A(i, k+1 : n) - multi * A(k, k + 1 : n );
14         b(i) = b(i) - multi * b(k);
15         %In fact, the position should be 0 still stays unchanged
16         %As we don't use them for calculation
17         %so we can image they are 0
18     end
19 end
20
21 x = zeros(n,1);
22 x(n) = b(n)/A(n,n);
23
24 for k = n-1:-1:1
25     x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
26 end
```

genpMyVersion.m

For the **first Loop(Line 10):**

We will first consider the **inner Loop(Line 11):**
in each iteration: We have 1 division first (**Line 12**)
We have $(n-k)$ multiplication and $(n-k)$ minus , total $2(n-k)$ (**Line 13**)
We have 1 multiplication and 1 minus (**Line 14**)
Thus, we have

$$\big(1 + 2(n-k) + 2\big)(n-k)$$

Thus for the total loop, we have

$$\sum_{k=1}^{n-1}\big(1 + 2(n-k) + 2\big)(n-k) \approx 2\sum_{k=1}^{n-1}(n-k)^2 = 2 \times \Big(1^2 + 2^2 \ldots + (n-1)^2\Big) \approx \frac{2}{3}n^3$$

For the **second Loop(Line 24):**
in each iteration: We have $(n-k)$ multiplication and $(n-k)$ minus + 1 division, total $2(n-k)+1$
Then for the total Loop, we have

$$\sum_{k=1}^{n-1} 2(n-k) + 1$$

```
1  function x = genpXW(A, b)
2  % genp.m    Gauss elimination with no pivoting
3  %
4  % input:  A is an n x n nonsingular matrix
5  %         b is an n x 1 vector
6  % output: x is the solution of Ax=b.
7  %
8  n = length(b);
9
10 for k = 1:n-1
11     i = k+1:n;
12
13    A(i,k) = A(i,k)/A(k,k);
14    A(i,i) = A(i,i) - A(i,k)*A(k,i);
15    b(i) = b(i) - A(i,k)*b(k);
16 end
17
18
19 x = zeros(n,1);
20 x(n) = b(n)/A(n,n);
21
22 for k = n-1:-1:1
23   x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
24 end
```

genpXW.m

Take our Matrix to be $\begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 8 & 9 \end{bmatrix}$

After **line 13**, we will get $\begin{bmatrix} 2 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 8 & 9 \end{bmatrix}$, which means **line 13** changes both $A(1,2), A(1,3)$

After **line 14**, we will get $\begin{bmatrix} 2 & 2 & 3 \\ 2 & 1 & 0 \\ 3 & 2 & 0 \end{bmatrix}$, which means **line 14** changes both $A(2,2), A(2,3), A(3,2), A(3,3)$

I noticed in Matlab Debugger, it says $i = [2,3]$ when I go into the loop. Here $A(i,i)$ is the matrix $A(k+1:n\ ,\ k+1:n)$ as $i = k+1:n$.

When $k = 1$, this is just $A(2:3\ ,\ 2:3)$.

GEPP:

Question: Why we do partial pivoting?

- GENP will fail $\implies$ as the first element may be 0, there could be cases that divide by 0
- avoid unnecessary loss of accuracy as $GENP$ is numerical stable

```
function x = geppMyVersion(A, b)

n = length(b);

for k = 1 : n-1
    [maxval, maxindex] = max(abs(A(k:n, k)));
    q = maxindex + k - 1;
    if maxval == 0, error("A is singular"), end

    A([k, q], k:n) = A([q, k], k:n);
    % We just need to switch from kth element to nth element
    % Because any elements before them are 0

    b([k, q]) = b([q, k]);
    % Now, we have already switched two rows

    for i = k+1 : n
        multi = A(i, k)/A(k,k);
        A(i, k+1:n) = A(i, k+1:n) - A(k, k+1:n) * multi;
        % We start from column k+1
        % Because any column before that is 0
        b(i) = b(i) - b(k) * multi;
    end

end

x = zeros(n, 1);

x(n) = b(n)/ A(n,n);

for k = n-1:-1:1
    x(k) = ( b(k) - A(k, k+1 : n) * x(k+1:n) ) / A(k,k);
end
```

geppMyVersion.m

COST: Flop cost is the same as before $\frac{2}{3}n^3$ and we need $\frac{1}{2}n^2$

```
function x = geppXW(A,b)
% genp.m    Gauss elimination with partial pivoting
%
% input:  A is an n x n nonsingular matrix
%         b is an n x 1 vector
% output: x is the solution of Ax=b.
%
n = length(b);

for k = 1:n-1
   [maxval, maxindex] = max(abs(A(k:n,k)));
   q = maxindex+k-1;
   if maxval == 0, error('A is singular'), end
   A([k,q],k:n) = A([q,k],k:n);
   b([k,q]) = b([q,k]);
   i = k+1:n;
   A(i,k) = A(i,k)/A(k,k);
   A(i,i) = A(i,i) - A(i,k)*A(k,i);
   b(i) = b(i) - A(i,k)*b(k);
end

x = zeros(n,1);
x(n) = b(n)/A(n,n);
for k = n-1:-1:1
  x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
end
```

geppXW.m

LU Factorization With Partial Pivoting

```
function [L,U,P] = luppMyVersion(A)

n = size(A,1);
P = eye(n);

for k = 1 : n-1
    [maxval, maxind] = max( abs(A(k:n, k)) );
    q = maxind + k - 1;
    if maxval == 0, error("A is singular"),end

    P([k,q], :) = P([q,k],:);
    A([k,q], :) = A([q,k],:);
    %You need to change entire two Rows here

    for i = k+1 : n
        A(i,k) = A(i, k) / A(k,k);
        A(i, k+1 : n) = A(i, k+1 : n) - A(i, k) * A(k, k+1: n);
    end

end

L = tril(A, -1) + eye(n);
U = triu(A);
```

luppMyVersion.m

This is how we use LUPP to solve linear equations:

```
function x = luppSolve(A, b)

n = size(A,1);
P = eye(n);

for k = 1 : n-1
    [maxval, maxind] = max( abs(A(k:n, k)) );
    q = maxind + k - 1;
    if maxval == 0, error("A is singular"),end

    P([k,q], :) = P([q,k],:);
    A([k,q], :) = A([q,k],:);
    %You need to change entire two Rows here
    b([k,q]) = b([q, k]);

    for i = k+1 : n
        A(i,k) = A(i, k) / A(k,k);
        A(i, k+1 : n) = A(i, k+1 : n) - A(i, k) * A(k, k+1: n);
    end

end
L = tril(A,-1) + eye(n);
U = triu(A);

x = zeros(n, 1);


x(1) = b(1);
for k = 2 : n
   x(k) = b(k) - L(k, 1:k - 1) * x(1: k-1);
end

x(n) = x(n)/ U(n,n);
for k = n-1 : -1 : 1
    x(k) = (x(k) -  U(k, k+1:n) * x(k+1:n)  ) / U(k,k);
end
```

luppSolve.m

Theoretical Results:

Accuracy depends on

- Algorithm
- Condition of Problems

Tridiagonal Matrix:

Algorithm

- In class, ideas are from $GENP$
- In midterm, you may be asked to write ideas from $GEPP$

Solving Tridiagonal Systems by GENP

Algorithm for solving

$$\begin{bmatrix} d_1 & c_1 & & & \\ a_1 & d_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & d_{n-1} & c_{n-1} \\ & & & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

for $i = 2 : n$
  $mult \leftarrow a_{i-1}/d_{i-1}$
  $d_i \leftarrow d_i - mult * c_{i-1}$
  $b_i \leftarrow b_i - mult * b_{i-1}$
end
$x_n \leftarrow b_n/d_n$
for $i = n-1 : -1 : 1$
  $x_i \leftarrow (b_i - c_i * x_{i+1})/d_i$
end

Cost: $8n$ flops.

Storage: store only a_i, c_i, d_i and b_i by using 4 1-dimensional arrays. Do not use a 2-dimensional array to store the whole matrix.

Diagonally Dominant Matrices

Let $A = (a_{ij})_{n\times n}$. A is strictly diagonally dominant by column if

$$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}| \; for \; j = 1 : n$$

Let $A = (a_{ij})_{n\times n}$. A is strictly diagonally dominant by row if

$$|a_{ii}| > \sum_{j=1, i\neq j}^{n} |a_{ij}| \; for \; i = 1 : n$$

If a tridiagonal A is strictly diagonally dominant by column, then partial pivoting is not needed, i.e., GENP and GEPP will give the same results.

If A is SDDC, then we have

- $|d_1| > |a_1|$
- $|d_i| > |c_{i-1}| + |a_i|$
- $|d_n| > |c_{n-1}|$

Step 1:
$multi = \frac{a_1}{d_1} < 1$; so there is no pivoting this step
$d_2' = d_2 - \frac{a_1}{d_1} \times c_1$
So, we just need to show the matrix bounded by d_2' is also SDDC.
Obviously, we just need to show $|d_2'| > |a_2|$

$$\begin{aligned}
|d_2'| &= |d_2 - \frac{a_1}{d_1} \times c_1| \\
&\geq |d_2| - |\frac{a_1}{d_1} \times c_1| \\
&\geq |d_2| - |\frac{a_1}{d_1}| \times |c_1| \\
&\geq |d_2| - |c_1| \qquad as\ |d_2| > |a_2| + |c_1| \\
&> |a_2|
\end{aligned}$$

If a tridiagonal A is strictly diagonally dominant by row, then GENP will not fail
If A is SDDR, then we have

- $|d_1| > |c_1|$
- $|d_i| > |a_{i-1}| + |c_i|$
- $|d_n| > |a_{n-1}|$

Step 1:
Since $|d_1| > |c_1| \geq 0$, GENP won't fail in first step

Step 2:
$d_2' = d_2 - \frac{a_1}{d_1} \times c_1$
So, we just need to show the matrix bounded by d_2' is also SDDR.
Obviously, we just need to show $|d_2'| > |c_2|$

$$\begin{aligned}
|d_2'| &= |d_2 - \frac{a_1}{d_1} \times c_1| \\
&\geq |d_2| - |\frac{a_1}{d_1} \times c_1| \\
&\geq |d_2| - |\frac{c_1}{d_1} \times a_1| \\
&\geq |d_2| - |\frac{c_1}{d_1}| \times |a_1| \\
&\geq |d_2| - |a_1| \qquad as\ |d_2| > |a_1| + |c_2| \\
&> |c_2|
\end{aligned}$$

Using GEPP to solve Tridiagonal Matrix

$$\begin{bmatrix} d_1 & c_1 & p_1 & 0 & 0 & 0 \\ a_1 & d_2 & c_2 & p_2 & 0 & 0 \\ 0 & a_2 & d_3 & c_3 & p_3 & 0 \\ 0 & 0 & a_3 & d_4 & c_4 & p_4 \\ 0 & 0 & 0 & a_4 & d_5 & c_5 \\ 0 & 0 & 0 & 0 & a_5 & d_6 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

We will use 5 1-D array to store the matrix d, c, a, p, b
P is initialized to 0 at the beginning, when we change two lines, there will be one more elements which will be stored to p

Algorithm 1 Using GEPP to solve Tridiagonal Matrix

for $i = 2 \to n-1$ **do**
    if $|d_{i-1}| = |a_{i-1}| = 0$ **then**
        "Error"
    end if
    if $|d_{i-1}| < |a_{i-1}|$ **then**
        $d_{i-1} \Leftrightarrow a_{i-1}$
        $d_i \Leftrightarrow c_{i-1}$
        $p_{i-1} \Leftrightarrow c_i$
        $b_i \Leftrightarrow b_{i-1}$
    end if
    $multi = \dfrac{a_{i-1}}{d_{i-1}}$

    $d_i = d_i - multi \times c_{i-1}$
    $c_i = c_i - multi \times p_{i-1}$
    $b_i = b_i - multi \times b_{i-1}$
end for

if $|d_{n-1}| < |a_{n-1}|$ **then**
    $d_{n-1} \Leftrightarrow a_{n-1}$
    $d_n \Leftrightarrow c_{n-1}$
    $b_n \Leftrightarrow b_{n-1}$
end if
$multi = \dfrac{a_{n-1}}{d_{n-1}}$

$d_n = d_n - multi \times c_{n-1}$
$b_n = b_n - multi \times b_{n-1}$

$x_n = \dfrac{b_n}{d_n}$

$x_{n-1} = \dfrac{b_{n-1} - c_{n-1} \times x_n}{d_n}$
for $i = n-2 \to -1$ **do**

    $x_i = \dfrac{b_i - c_i \times x_{i+1} - p_i \times x_{i+2}}{d_i}$

end for