Object Oriented Programming
Lecture - 7

Instructor
Prof. S Sundar
and
D Satyaprasad (TA)

PROJECT - 1
Nonlinear PDE Solver

BVP:

$$A(x,y,u)\frac{\partial^2 u}{\partial x^2} + B(x,y,u)\frac{\partial^2 u}{\partial y^2} + C(x,y,u)\frac{\partial u}{\partial x} + D(x,y,u)\frac{\partial u}{\partial y} + E(x,y,u) = 0$$

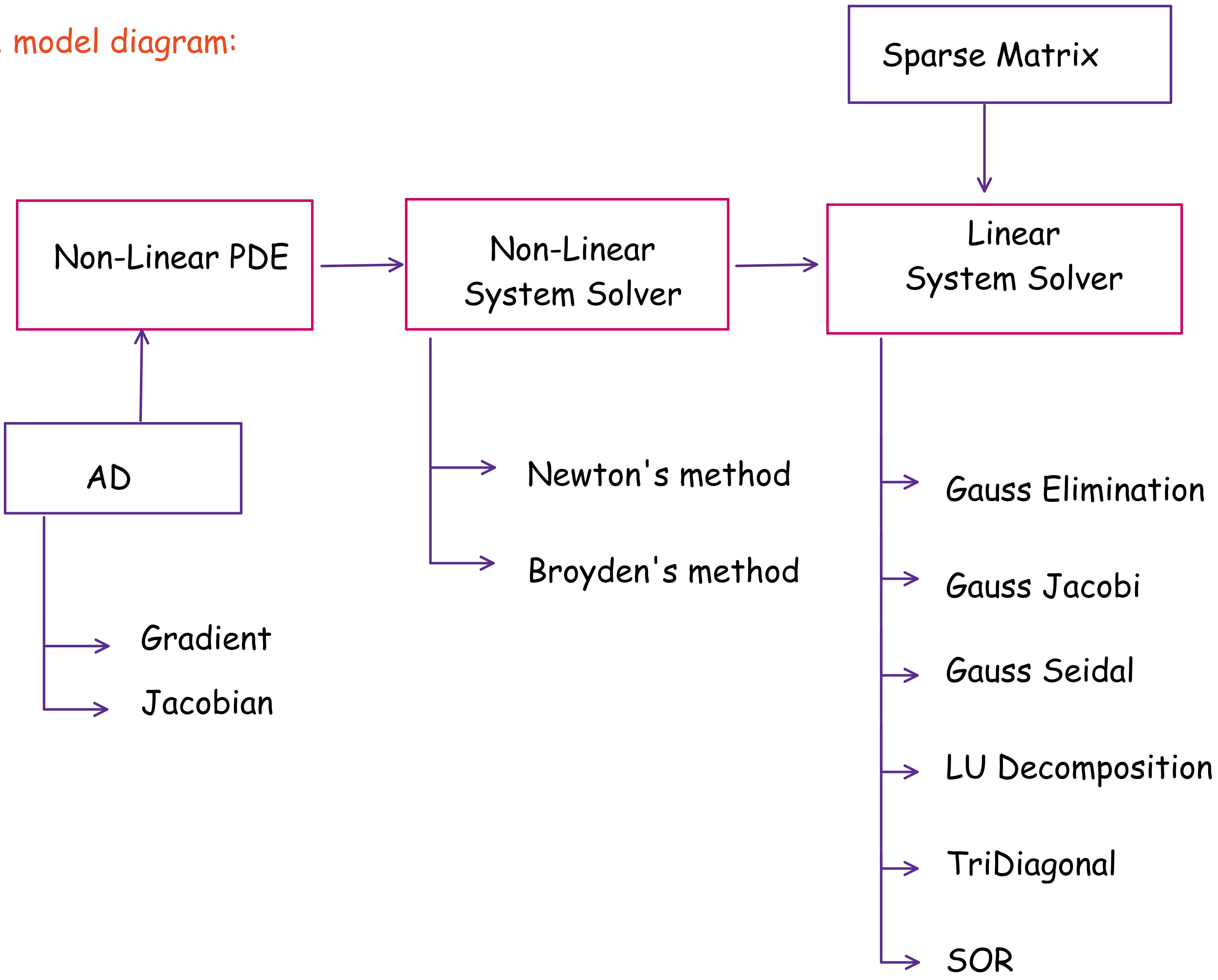
on $[a,b] \times [c,d]$

with boundary conditions

$$u(a,y) = f_1(y), u(b,y) = f_2(y)$$

$$u(x,c) = g_1(x), u(x,d) = g_2(x)$$

A model diagram:



Solving given PDE of above type involve the following steps:

Step - 1 : Input the PDE with boundary conditions.

Step -2 : Discretization of PDE using finite difference method.
Generate the system of nonlinear algebraic equations.

Step-3 : Solve the system of nonlinear algebraic equations using Newton's or Broyden's method.

Step-4 : Solve the system of linear equations.

Step-5: Plot the results. (Typically a surface plot)

Illustration:

Given problem:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 2 \text{ on } (0,1) \times (0,1)$$

$$u(0,y) = u(1,y) = 0, \text{ on } y \in [0,1]$$

$$u(x,0) = u(x,1) = 0, \text{ on } x \in [0,1]$$

Step-1: (Input)

A = 1, B = 1, C = 0, D = 0, E = -2

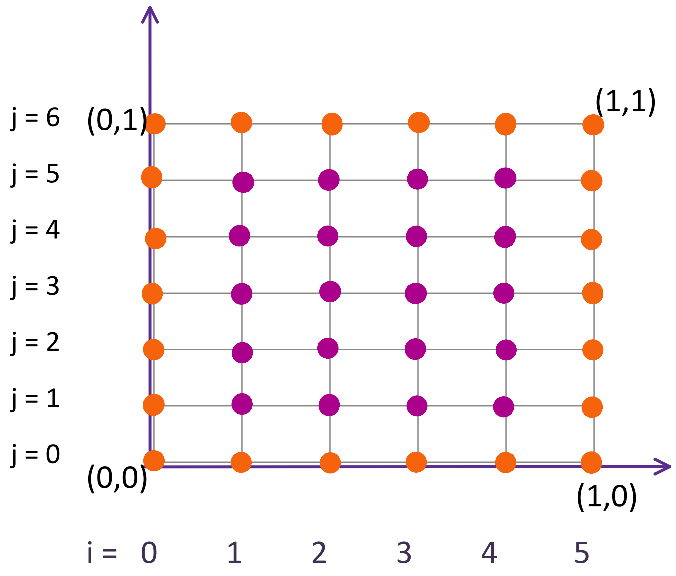
a = 0, b = 1
c = 0, d = 1

$f_1(y) = f_2(y) = g_1(x) = g_2(x) = 0$

Step-2: (Discretization)

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 2$$

for, i = 0, 1, 2, …… m
j = 0, 1, 2, …… n



Substituting, i from 1 to 4 and j from 1 to 5 in the above discretization results a system of 20 nonlinear algebraic equations.

Let us name the nonlinear algebraic equations as

$$\left.\begin{aligned} f_1(u_1, u_2, \ldots u_{20}) &= 0 \\ f_2(u_1, u_2, \ldots u_{20}) &= 0 \\ &\,. \\ &\,. \\ &\,. \\ &\,. \\ f_{20}(u_1, u_2, \ldots u_{20}) &= 0 \end{aligned}\right\} \equiv \boldsymbol{F}(\boldsymbol{x})$$

(Output of the step-2

&

Input for step-3)

Step - 3: (Nonlinear algebraic system solver)

Solving $\boldsymbol{F}(\boldsymbol{x}) = 0$

Newton's method:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} - \frac{\boldsymbol{F}(\boldsymbol{x}^{(k-1)})}{J(\boldsymbol{x}^{(k-1)})}$$

where,

k = 1, 2, . . . , m represents the iteration,

$\mathbf{x} \in \mathbb{R}^n$,

$\boldsymbol{F}$ is a vector function,

$J(\boldsymbol{x})$ is Jacobian matrix

Inparticular for above discretization, here $n = 4 \times 5 = 20$

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

Let $\boldsymbol{y}^{(k-1)} = -\frac{\boldsymbol{F}(\boldsymbol{x}^{(k-1)})}{J(\boldsymbol{x}^{(k-1)})}$

$$J(\boldsymbol{x}^{(k-1)})\boldsymbol{y}^{(k-1)} = -\boldsymbol{F}(\boldsymbol{x}^{(k-1)})$$

This is in the form Ax = b (linear system) then the Newton iteration scheme,

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{y}^{(k-1)}$$

Now we describe the steps of Newton's method:

Step - 1:

Let $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ be a given initial vector.

Step - 2:

Calculate $J(\mathbf{x}^{(0)})$ and $\mathbf{F}(\mathbf{x}^{(0)})$.

Step - 3:

In order to find $\mathbf{y}^{(0)}$, we solve the linear system $J(\mathbf{x}^{(0)})\mathbf{y}^{(0)} = -\mathbf{F}(\mathbf{x}^{(0)})$

Step - 4:

Once $\boldsymbol{y}^{(0)}$ is found, we can now proceed to finish the first iteration by solving $\boldsymbol{x}^{(1)}$.

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \mathbf{y}^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} + \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \vdots \\ y_n^{(0)} \end{bmatrix}$$

Step - 5:

Repeat the process again, until $\boldsymbol{x}^{(k)}$ converges to $\overline{\boldsymbol{x}}$.

i.e., $\left\| \boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)} \right\| < \varepsilon$

This indicates we have reached the solution to $\boldsymbol{F}(\boldsymbol{x}) = 0$, where $\overline{\boldsymbol{x}}$ is the solution to the system.

Step - 4 : (Linear system solver Ax = b)

This step (solving Ax = b) invokes while running every iteration of Newton's method in Step - 3.

There are several methods to solve system of linear equations such as

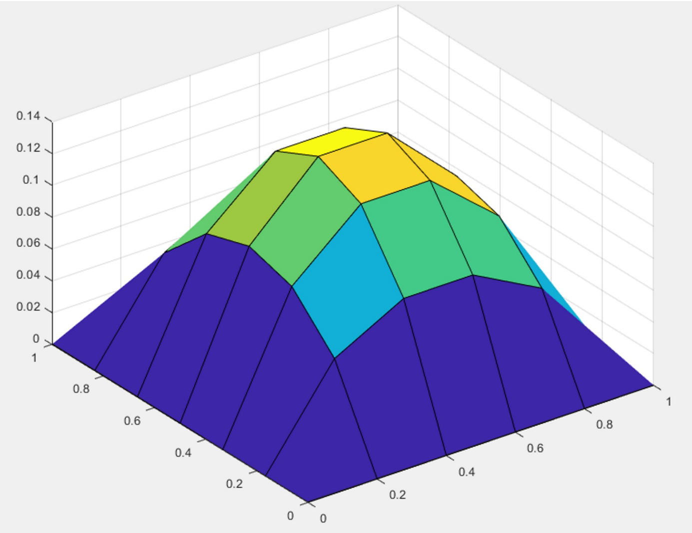
Gauss Elimination
Gauss Jacobi

Gauss Seidal
LU Decomposition
TriDiagonal
SOR
etc.

Step - 5 : (Plotting results)

Plot the results from the solution $\overline{\boldsymbol{x}}$ of $\boldsymbol{F}(\boldsymbol{x}) = 0$

That is, $\overline{\boldsymbol{x}} = (u_1, u_2, \ldots u_{20})$ at grid points.



Automatic Differentiation (AD) class with gradient and Jacobian:

```
#ifndef AD_H
#define AD_H
#include<cmath>
#include "vector.h"
#include "matrix.h"

int varCount = 0;  // to keep track number of independent variables used.

class AD{
private:
   double f;
   vector df;
   int id;

public:
   AD();
   AD(double);
   void setIndVar();
   double getf();
   double getDf(int);
```

```
    vector getGradient();
    friend matrix getJacobian(AD*);

    AD operator *(AD);
    AD operator +(AD);
    AD operator*(double);

    friend AD sin(AD);
    friend AD cos(AD);

};



AD :: AD(){
    f = 0;
}

AD :: AD(double value){
    this->f = value;
    this->id = varCount;
    varCount++;
}

void AD :: setIndVar(){
    this->df = vector(varCount);
    for(int i=0 ; i<this->id ; i++)
```

```
        this->df[i] = 0;
    this->df[this->id] =  1;
    for(int i = this->id + 1 ; i < varCount ;i++)
        this->df[i] = 0;
}

double AD :: getf(){
   return this->f;
}
double AD :: getDf(int index){
    return df[index];
}

vector AD :: getGradient(){
    vector gradient(varCount);
    for(int i=0 ;i<varCount;i++)
        gradient[i] = this->df[i];
    return gradient;
}

matrix getJacobian(AD funList[]){
    int n = varCount;
    matrix M = matrix(n,n);
      for(int i = 0; i<n ; i++)
       for(int j = 0 ; j< n ; j++)
         M(i,j) = funList[i].getDf(j);
    return M;
```

```
}

//Binary operators

AD AD :: operator +(AD g){
   AD h;
   h.f = this->f + g.f;
   h.df = vector(varCount);
   for(int i = 0 ;i<varCount ;i++)
     h.df[i] = this->df[i] + g.df[i];
   return h;
}

AD AD :: operator *(AD g){
   AD h;
   h.f = this-> f * g.f;
   h.df = vector(varCount);
   for(int i = 0 ; i <varCount ;i++)
      h.df[i] = (this->f * g.df[i]  + g.f * this->df[i]);
   return h;
}

AD AD :: operator *(double s){
   AD h;
   h.f = s*(this->f);
   h.df = vector(varCount);
   for(int i = 0; i<varCount ;i++)
```

```
        h.df[i] = s*this->df[i];
    return h;
}

AD sin(AD g){
    AD h;
    h.f = sin(g.f);
    h.df = vector(varCount);
    for(int i=0 ;i < varCount ; i++)
        h.df[i] = cos(g.f)*g.df[i];
    return h;
}

AD cos(AD g){
    AD h;
    h.f = cos(g.f);
    h.df = vector(varCount);
    for(int i=0 ;i < varCount ; i++)
        h.df[i] = -sin(g.f)*g.df[i];
    return h;
}


#endif
```

Test Program:

```
#include <stdio.h>
#include "vector.h"
#include "AD.h"
#include "matrix.h"
int main()
{
    AD x(3), y(8), z(-1);

    // set x,y,z as independent variables.
    x.setIndVar();
    y.setIndVar();
    z.setIndVar();

    // Input f,g,h as functions of our interest.
    AD f,g,h;
    f = x*y*z + sin(x*y) * 2 + x*y*cos(z);
    g = x*x +y*y + z*z + x*y*z;
    h = x*y + y*z + z*x;
    AD funArray[] = {f,g,h};

    // Evaluate Gradient of f
    cout<<"Gradient of f is :  ";
    f.getGradient().print();

    // Evaluate Jacobian
```

```
    matrix J = getJacobian(funArray);
    cout<<"Jacobian matrix : \n"<<endl;
    J.print();

    return 0;
}
```

OUTPUT:

```
Gradient of f is :  3.10928     1.16598 44.1953

Jacobian matrix :

3.10928 1.16598 44.1953
-2      13      22
7       2       11
```

Thank you