Nonlinear systems

Newton's method

The steepest descent method

Nonlinear systems

A general system of n nonlinear equations in n unknowns

$$F(x_1, x_2, \ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

$f_1, f_2, \ldots, f_n$ are coordinate functions of F

Solving systems of nonlinear equations

Numerical in chemical engineering

Numerical Solution of Large Sets of Algebraic Nonlinear Equations

Nonlinear systems
Multi-scale analysis of complex chemistry systems

Example

$$3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$

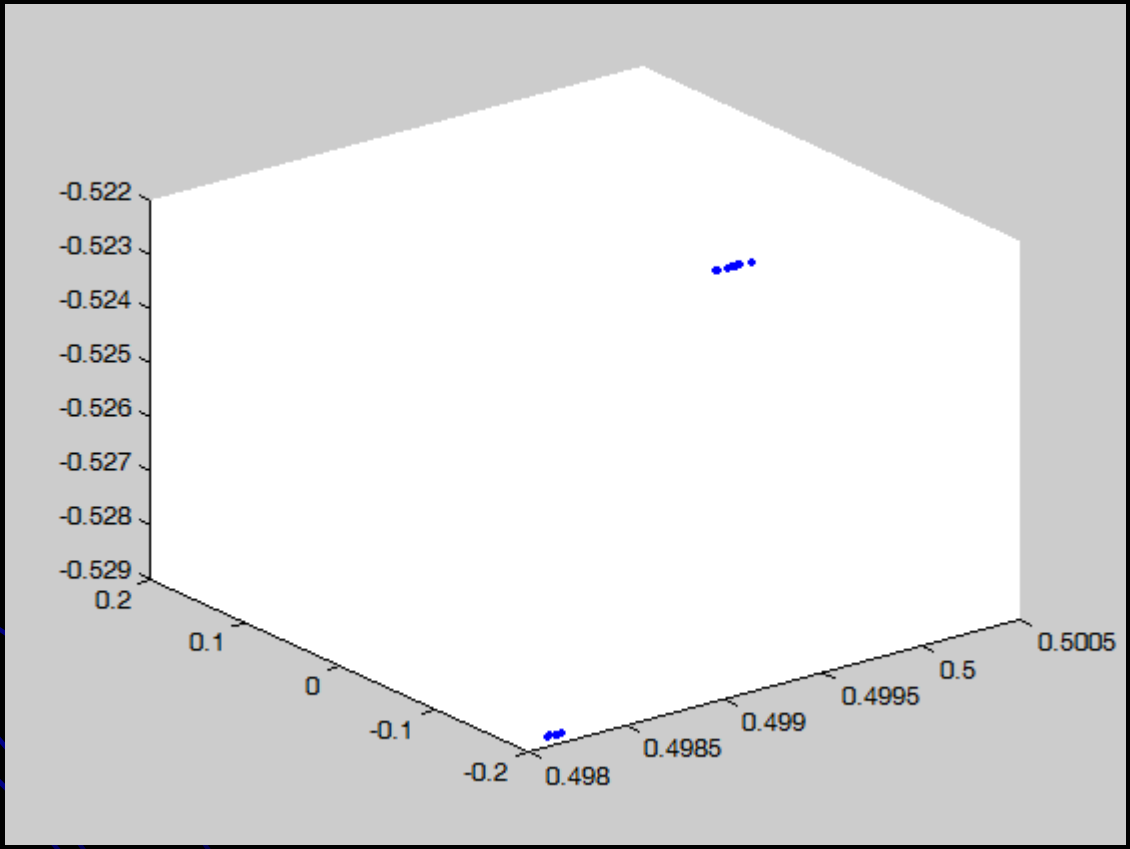
$$x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0$$

$$e^{-x_1 x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

$$\begin{cases} e^{x_1^2+x_2^2} - 1 = 0, \\ e^{x_1^2-x_2^2} - 1 = 0, \end{cases}$$

myfun

```
function F = myfun(x)
    F(1) =  3*x(1)-cos(x(2)*x(3))-1/2;
    F(2) =  x(1).^2 -81*(x(2)+0.1).^2+sin(x(3))+1.06;
    F(3) = exp(-x(1)*x(2))+20*x(3)+1/3*(10*pi-3);
return
```


-0.522
-0.523
-0.524
-0.525
-0.526
-0.527
-0.528
-0.529
0.2
0.1
0
-0.1
-0.2
0.498
0.4985
0.499
0.4995
0.5
0.5005

Solve nonlinear systems

```
x0= rand(1,3)-0.5;
x = lsqnonlin(@myfun,x0);
y=myfun(x);
sum(y.^2);
```

$$f_k(x_1, x_2) = 2 + 2k - e^{kx_1} - e^{kx_2} = 0$$

$$k = 1, \ldots, 10$$

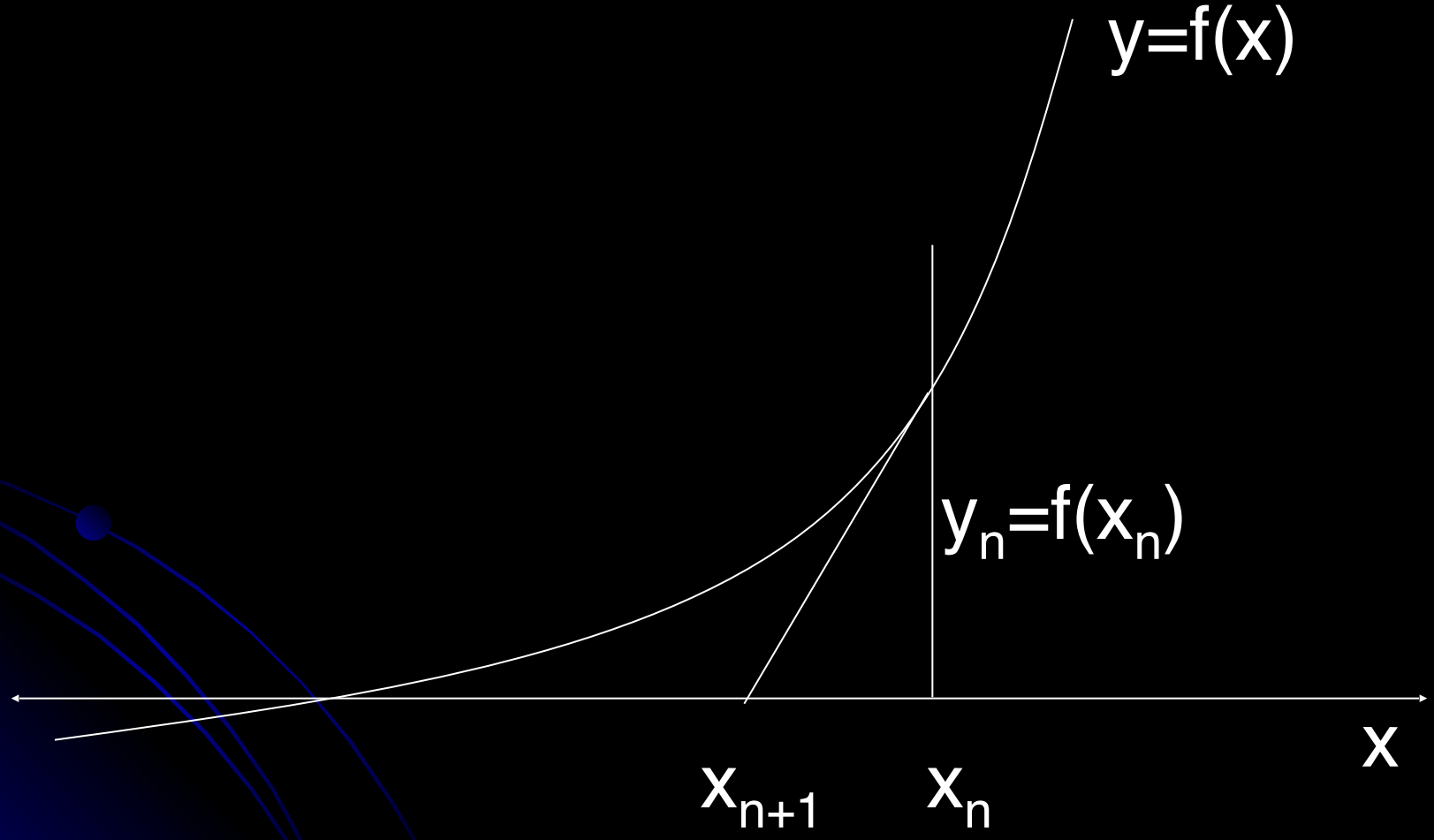
$$E(x_1, x_2) = \sum_{k=1}^{10} [f_k(x_1, x_2)]^2$$

$$X^* = \min_{x_1, x_2} E(x_1, x_2)$$

Demo_ex1

demo_ex1.m

Newton's method -Tangent line



Updating rule

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

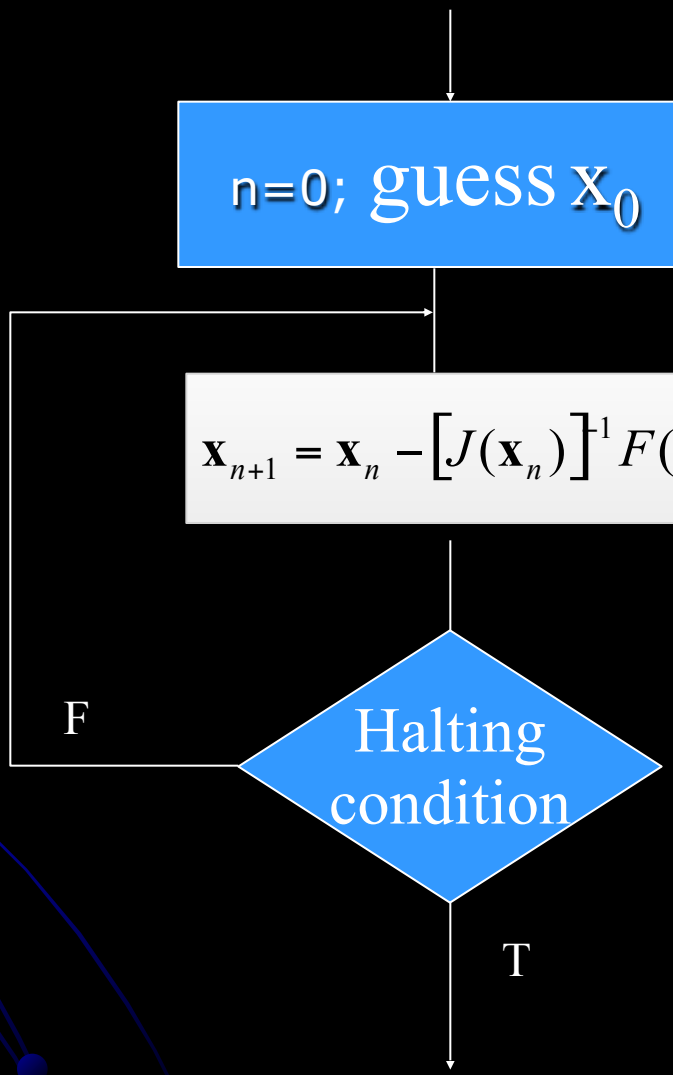
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[J(\mathbf{x})\right]^{-1} F(\mathbf{x}_n)$$

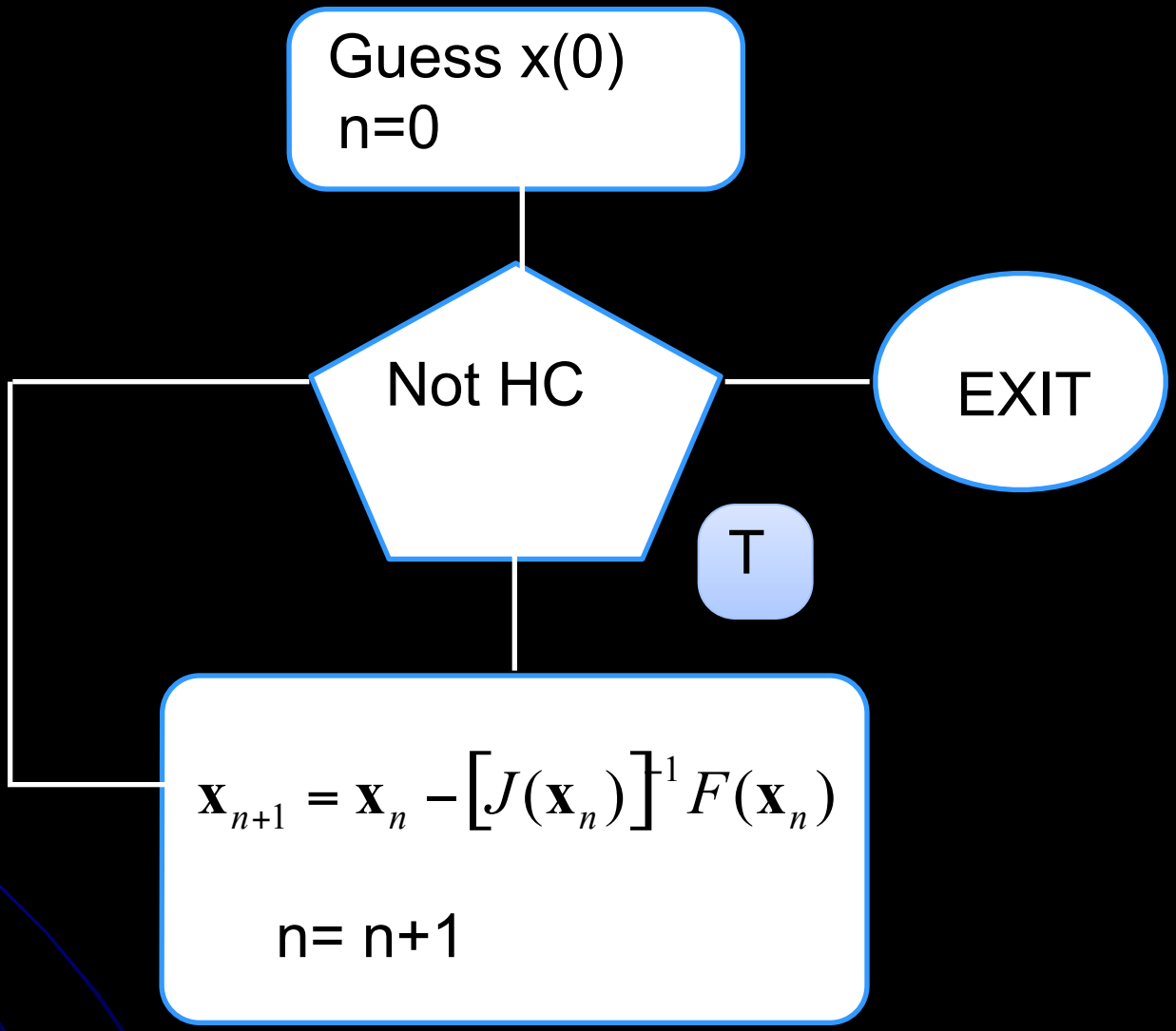
Newton's method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[J(\mathbf{x})\right]^{-1} F(\mathbf{x}_n)$$

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

Flow Chart


n=0; guess x_0
$\mathbf{x}_{n+1} = \mathbf{x}_n - [J(\mathbf{x}_n)]^{-1} F(\mathbf{x}_n)$
Halting condition
F
T


Guess x(0)
n=0
Not HC
EXIT
T
$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[J(\mathbf{x}_n)\right]^{-1} F(\mathbf{x}_n)$
n= n+1

Matlab coding

Matlab modules:

1. 3-variable function evaluation
    1. Detect the number of variables
    2. Translate to 3-variable functions
    3. Function Evaluation
2. Use symbolic differentiation to determine partial derivatives
3. Evaluate partial derivatives
4. Main program: use a while loop to update x iteratively

Function evaluation

```
s1='3*x1-cos(x2*x3)-1/2';
f=inline(s1);
x=rand(1,3);
y=f(x(1),x(2),x(3));
```

Evaluation of vector function

```
s1='3*x1-cos(x2*x3)-1/2';
s2='x1.^2 -81*(x2+0.1).^2+sin(x3)+1.06';
s3='exp(-x1*x2)+20*x3+1/3*(10*pi-3)';
f1=inline(s1); f2=inline(s2);f3=inline(s3);
x=rand(1,3);
Y=f1(x(1),x(2),x(3));
Y=[Y f2(x(1),x(2),x(3))];
Y=[Y f3(x(1),x(2),x(3))];
```

$$\mathrm{F}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{pmatrix}$$

Variant function form

```
s='x1+x2';
f=inline(s);
x=rand(1,3);
y=f(x(1),x(2),x(3));
```

Error message:
Too many input to inline function

Correct codes

```
s='x1+x2';
f=inline(s);
x=rand(1,3);
y=f(x(1),x(2));
```

```
s='x1+x3';
f=inline(s);
x=rand(1,3);
y=f(x(1),x(3));
```

Correct codes

```
s='x2+x3';
f=inline(s);
x=rand(1,3);
y=f(x(2),x(3));
```

Correct codes

```
s='1';
f=inline(s);
x=rand(1,3);
y=f(0);
```

Correct codes

```
s='x1+x2+x3';
f=inline(s);
x=rand(1,3);
y=f(x(1),x(2),x(3));
```

Correct codes

symvar

sc=symvar(f);

- f denotes an inline function
- sc is a cell structure
  - Symbols in f

Symbols in an inline function

```
s='x1+x2';
f=inline(s);
```

```
s='x1+x3';
f=inline(s);
```

```
s='x2+x3';
f=inline(s);
```

```
s='1';
f=inline(s);
```

```
s='x1+x2+x3';
f=inline(s);
```

```
sc=symvar(f);
```

```
sc =
    'x1'
    'x2'
```

```
sc =
    'x1'
    'x3'
```

```
sc =
    'x2'
    'x3'
```

```
sc =
    'x'
```

```
sc =
    'x1'
    'x2'
    'x3'
```

```
y=f(x(1),x(2));
```

```
y=f(x(1),x(3));
```

```
y=f(x(2),x(3));
```

```
y=f(0);
```

```
y=f(x(1),x(2),x(3));
```

3-variable function evaluation

xx=fun3v(f)

- f: an inline function
- xx: a string for 3-variable function evaluation

Newton\fun3v.m

```
function xx=fun3v(f)
s=symvar(f);xx=[];
for i=1:size(s,1)
    if i>1
        xx=[xx ','];
    end
    arg=char(s(i));
    switch arg
        case 'x'
            xx=[xx '0'];
        case 'x1'
            xx=[xx 'x(1)'];
        case 'x2'
            xx=[xx 'x(2)'];
        case 'x3'
            xx=[xx 'x(3)'];
    end
end
xx=['f(' xx ')'];
return
```

example

```
s='x1+x2';
f=inline(s);
```

```
s='x1+x3';
f=inline(s);
```

```
s='x2+x3';
f=inline(s);
```

```
s='1';
f=inline(s);
```

```
s='x1+x2+x3';
f=inline(s);
```

```
xx=fun3v(f)
x=[1 2 3]
eval(xx)
```

3-variable function evaluation

```
function v=feval3v(f,x)
xx=fun3v(f);
eval(xx)
return
```

Matlab codes

feva.m

```
function v=feva(f,x)
s=symvar(f);
xx=[];
for i=1:size(s,1)
    ss=char(s(i));
    switch ss
        case 'x'
            xx=[xx 0];
        case 'x1'
            xx=[xx x(1)];
        case 'x2'
            xx=[xx x(2)];
        case 'x3'
            xx=[xx x(3)];
    end
end
xx=['f(' xx ')'];
v=eval([sprintf(xx)]);
return
```

Partial derivation

```
s='3*x1-cos(x2*x3)-1/2';
fx=inline(s);
x1=sym('x1');
x2=sym('x2');
x3=sym('x3');
J1=inline(diff(s,x1))
J2=inline(diff(s,x2))
J3=inline(diff(s,x3))
```

function [J1,J2,J3]=pdiff(s)

Newton\pdiff.m

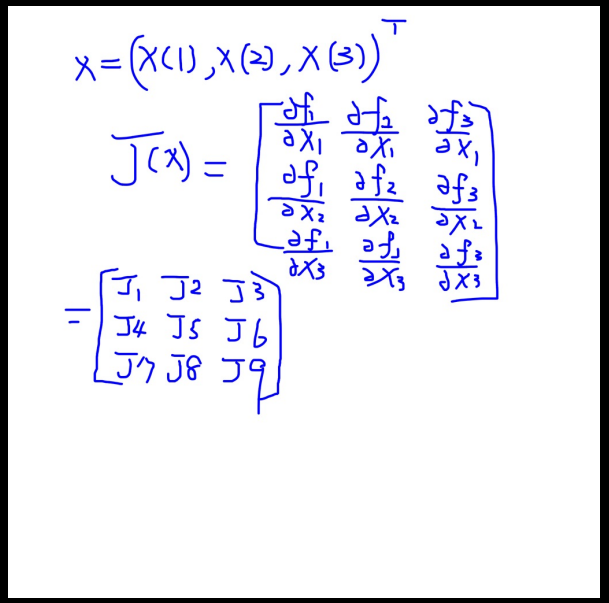
```
function [J1,J2,J3]=pdiff(s)
fx=inline(s);
x1=sym('x1');
x2=sym('x2');
x3=sym('x3');
J1=inline(diff(s,x1));
J2=inline(diff(s,x2));
J3=inline(diff(s,x3));
return
```

Example

```
s='3*x1-cos(x2*x3)-1/2';
[J1 J2 J3]=pdiff(s)
J1 =
    Inline function:
    J1(x) = 3
J2 =
    Inline function:
    J2(x2,x3) = sin(x2.*x3).*x3
J3 =
    Inline function:
    J3(x2,x3) = sin(x2.*x3).*x2
```

Partial derivatives

```
s1='3*x1-cos(x2*x3)-1/2';
s2='x1^2 -81*(x2+0.1)^2+sin(x3)+1.06';
s3='exp(-x1*x2)+20*x3+1/3*(10*pi-3)';
[J1 J2 J3]=pdiff(s1);
[J4 J5 J6]=pdiff(s2);
[J7 J8 J9]=pdiff(s3);
```



Newton\demo.m

```
s1='3*x1-cos(x2*x3)-1/2';
s2='x1^2 -81*(x2+0.1)^2+sin(x3)+1.06';
s3='exp(-x1*x2)+20*x3+1/3*(10*pi-3)';
x0=rand(3,1)-0.5;
x=Newton(s1,s2,s3,x0)
```

Function x=Newton(s1,s2,s3,x0)

s1: string of function f1

s2: string of function f2

s3: string of function f3

x0: initial guess

```
function x=Newton(s1,s2,s3,x0)
f1=inline(s1); f2=inline(s2); f3=inline(s3);
[J1 J2 J3]=pdiff(s1);
[J4 J5 J6]=pdiff(s2);
[J7 J8 J9]=pdiff(s3);
ep=10^(-6); x=x0; it=0;
while sum(abs(y)) > ep & it < 100
    %
    % Determine F(1), F(2), F(3) by substituting x to f1, f2 and f3
    %
    % Determine J by substituting x to J1, J2,…, J9
    %
    % Update x
    y(1)=feva(f1,x); y(2)=feva(f2,x);y(3)=feva(f3,x);
    it=it+1;
    fprintf(' iter=%d criterion=%f\n',it,sum(y.^2));
end
return
```

Determine F(1),F(2) and F(3)
y(1)=F(1);y(2)=F(2);y(3)=F(3)

Exercise

- Write matlab codes to implement the Newton's method for solving a three-variable nonlinear system
- Test your matlab function with the following nonlinear system

$$3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$

$$x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0$$

$$e^{-x_1 x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

The steepest descent method

$$F(x_1, x_2, \ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_1(x_1, x_2, \ldots, x_n) \\ \ldots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

is translated to minimize

$$g(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \left[f_i(x_1, x_2, \ldots, x_n) \right]^2$$

Gradient

$$\nabla g(\mathbf{x}) = \left(\frac{\partial g}{\partial x_1}(\mathbf{x}), \frac{\partial g}{\partial x_2}(\mathbf{x}), \cdots, \frac{\partial g}{\partial x_n}(\mathbf{x}) \right)^T$$

Iterative process

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla g(\mathbf{x}^{(0)})$$

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} - \alpha \nabla g(\mathbf{x}^{(i-1)})$$

$$\nabla g(x_1, x_2, x_3) \equiv \nabla g(\mathbf{x}) = \begin{pmatrix} 2f_1(\mathbf{x})\dfrac{\partial f_1}{\partial x_1}(\mathbf{x}) + 2f_2(\mathbf{x})\dfrac{\partial f_2}{\partial x_1}(\mathbf{x}) + 2f_3(\mathbf{x})\dfrac{\partial f_3}{\partial x_1}(\mathbf{x}) \\ 2f_1(\mathbf{x})\dfrac{\partial f_1}{\partial x_2}(\mathbf{x}) + 2f_2(\mathbf{x})\dfrac{\partial f_2}{\partial x_2}(\mathbf{x}) + 2f_3(\mathbf{x})\dfrac{\partial f_3}{\partial x_2}(\mathbf{x}) \\ 2f_1(\mathbf{x})\dfrac{\partial f_1}{\partial x_3}(\mathbf{x}) + 2f_2(\mathbf{x})\dfrac{\partial f_2}{\partial x_3}(\mathbf{x}) + 2f_3(\mathbf{x})\dfrac{\partial f_3}{\partial x_3}(\mathbf{x}) \end{pmatrix}$$

$$= 2J(\mathbf{x})^T F(\mathbf{x})$$

Newton's method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[J(\mathbf{x})\right]^{-1} F(\mathbf{x}_n)$$

Steepest gradient descent method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla g(\mathbf{x}_n)$$

$$= \mathbf{x}_n - 2\alpha J(\mathbf{x}_n)^T F(\mathbf{x}_n)$$

$$\min_{\alpha} g(\mathbf{x}_{n+1})$$

Steepest descent

- Choose α to minimize

$$h(\alpha) = g(\mathbf{x}^{(i)})$$

$$h(\alpha) = g(\mathbf{x}^{(i-1)} - \alpha \nabla g(\mathbf{x}^{(i-1)}))$$

$$h(\alpha) = g(\mathbf{x}^{(i-1)} - \alpha \nabla g(\mathbf{x}^{(i-1)}))$$

$$\alpha = 0, \Rightarrow h(0) = g(\mathbf{x}^{(i-1)})$$

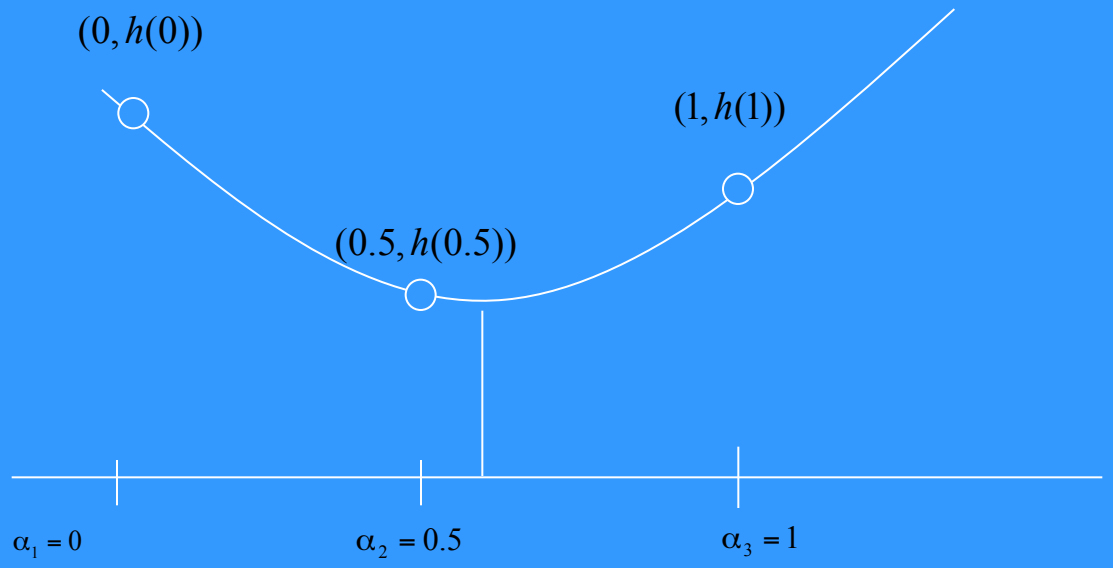
$$\alpha = 1, \Rightarrow h(1) = g(\mathbf{x}^{(i-1)} - \nabla g(\mathbf{x}^{(i-1)}))$$

$$\alpha = \frac{1}{2} \Rightarrow h(\frac{1}{2}) = g(\mathbf{x}^{(i-1)} - \frac{1}{2} \nabla g(\mathbf{x}^{(i-1)}))$$

Quadratic polynomial:

$$P(\alpha) = h(\alpha_1)\frac{\alpha-\alpha_2}{\alpha_1-\alpha_2}\frac{\alpha-\alpha_3}{\alpha_1-\alpha_3} + h(\alpha_2)\frac{\alpha-\alpha_1}{\alpha_2-\alpha_1}\frac{\alpha-\alpha_3}{\alpha_2-\alpha_3}$$

$$+ h(\alpha_3)\frac{\alpha-\alpha_1}{\alpha_3-\alpha_1}\frac{\alpha-\alpha_2}{\alpha_3-\alpha_2}$$



$$P(\alpha) = h(\alpha_1)\frac{\alpha-\alpha_2}{\alpha_1-\alpha_2}\frac{\alpha-\alpha_3}{\alpha_1-\alpha_3} + h(\alpha_2)\frac{\alpha-\alpha_1}{\alpha_2-\alpha_1}\frac{\alpha-\alpha_3}{\alpha_2-\alpha_3}$$

$$+ h(\alpha_3)\frac{\alpha-\alpha_1}{\alpha_3-\alpha_1}\frac{\alpha-\alpha_2}{\alpha_3-\alpha_2}$$

$$\frac{dP(\alpha)}{d\alpha} = h(\alpha_1)\frac{\alpha-\alpha_2+\alpha-\alpha_3}{(\alpha_1-\alpha_2)(\alpha_1-\alpha_3)} + h(\alpha_2)\frac{\alpha-\alpha_1+\alpha-\alpha_3}{(\alpha_2-\alpha_1)(\alpha_2-\alpha_3)}$$

$$+ h(\alpha_3)\frac{\alpha-\alpha_1+\alpha-\alpha_2}{(\alpha_3-\alpha_1)(\alpha_3-\alpha_2)} = 0$$

Function x=Steepest(s1,s2,s3,x0)

1. Create three nonlinear inline functions
2. Create nine inline functions for partial derivatives of the three nonlinear functions
3. Set x to x0
4. While the halting condition is not satisfied
   - Substitute x to all partial derivatives to form a 3x3 matrix J
   - Substitute x to the three nonlinear functions
   - Find optimal
   - Update x by the steepest descent method
5. Return x