

A system of nonlinear equations

Newton's method

Levenberg-Marquardt method

Outline

- A system of nonlinear equations
- Matlab toolbox: fsolve
- fsolve by the Levenberg-Marquardt method
- Newton's method for nonlinear system solving
 - Updating rule
 - Matlab implementation

```
function F = myfun(x)
```

$$F(1) = x(1)^2 + x(2)^2 - 1;$$

$$F(2) = x(1)^2 - x(2)^2;$$

```
return
```

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 1$$
$$f_2(x_1, x_2) = x_1^2 - x_2^2$$

symbols

```
s1='x1^2+x2^2-1';
```

```
s2='x1^2-x2^2';
```

```
x1=sym('x1')
```

```
x2=sym('x2')
```

Inline Function

```
f=inline([str2sym(s1);str2sym(s2)]);  
f(0,0)
```

fsolve

```
x=fsolve(@(x) [x(1)^2+x(2)^2-1  
x(1)^2-x(2)^2],[1 1])
```

```
x =
```

```
0.7071
```

```
0.7071
```

```
s1='x1^2+x2^2-1';
```

```
s2='x1^2-x2^2';
```

```
x1=sym('x1')
```

```
x2=sym('x2')
```

```
f=inline([sym(s1);sym(s2)]);
```

```
f(x(1),x(2))
```

```
ans =  
  
1.0e-11 *  
  
0.2282  
0
```

zeros

Jacobian

```
A=jacobian([str2sym(s1);str2sym(s2)],[x1  
x2]);
```

```
j=inline(A);
```

```
j(1,1)
```

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 1$$

$$f_2(x_1, x_2) = x_1^2 - x_2^2$$

```
A =
```

```
[ 2*x1,  2*x2]
```

```
[ 2*x1, -2*x2]
```


fsolve

<https://www.mathworks.com/help/optim/ug/fsolve.htm>
[mathwork](#)

$$\begin{aligned}e^{-e^{-(x_1+x_2)}} - x_2(1+x_1^2) &= 0 \\ x_1 \cos(x_2) + x_2 \sin(x_1) - \frac{1}{2} &= 0.\end{aligned}$$

Write a function that computes the left-hand side of these two equations.

```
function F = root2d(x)
```

```
F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
```

```
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

```
function demo_fsolve_0()
```

```
fun = @root2d;
```

```
x0 = [0,0];
```



```
x = fsolve(fun,x0)
```

```
root2d(x)
```

```
function F = root2d(x)
```

```
F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
```

```
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

- 
- [1] Coleman, T.F. and Y. Li, "An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds," *SIAM Journal on Optimization*, Vol. 6, pp. 418-445, 1996.
 - [2] Coleman, T.F. and Y. Li, "On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization Subject to Bounds," *Mathematical Programming*, Vol. 67, Number 2, pp. 189-224, 1994.
 - [3] Dennis, J. E. Jr., "Nonlinear Least-Squares," *State of the Art in Numerical Analysis*, ed. D. Jacobs, Academic Press, pp. 269-312.
 - [4] Levenberg, K., "A Method for the Solution of Certain Problems in Least-Squares," *Quarterly Applied Mathematics* 2, pp. 164-168, 1944.
 - [5] Marquardt, D., "An Algorithm for Least-squares Estimation of Nonlinear Parameters," *SIAM Journal Applied Mathematics*, Vol. 11, pp. 431-441, 1963.
 - [6] Moré, J. J., "The Levenberg-Marquardt Algorithm: Implementation and Theory," *Numerical Analysis*, ed. G. A. Watson, Lecture Notes in Mathematics 630, Springer Verlag, pp. 105-116, 1977.
 - [7] Moré, J. J., B. S. Garbow, and K. E. Hillstom, *User Guide for MINPACK 1*, Argonne National Laboratory, Rept. ANL-80-74, 1980.
 - [8] Powell, M. J. D., "A Fortran Subroutine for Solving Systems of Nonlinear Algebraic Equations," *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz, ed., Ch.7, 1970.
- 



QUARTERLY
OF
APPLIED
MATHEMATICS

BROWN UNIVERSITY

Online ISSN 1552-4485; Print ISSN 0033-569X

A method for the solution of certain non-linear problems in least squares

An Algorithm for Least-Squares Estimation of Nonlinear Parameters

Donald W. Marquardt

Journal of the Society for Industrial and Applied Mathematics, Vol. 11, No. 2 (Jun., 1963), 431-441.

Stable URL:

<http://links.jstor.org/sici?sici=0368-4245%28196306%2911%3A2%3C431%3AAAFLEO%3E2.0.CO%3B2-7>

Journal of the Society for Industrial and Applied Mathematics is currently published by Society for Industrial and Applied Mathematics.



Nonlinear systems

A system of nonlinear equations

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

f_1, f_2, \dots, f_n are coordinate functions of F

Example

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

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

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

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
```



```
options = optimoptions('fsolve')  
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
```

```
function demo_fsolve_1()
% problem setting
problem.x0 = [0,0];

display('fsolve by levenberg-marquardt');
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')

FUN=@(x)root2d(x);
tic
x = fsolve(FUN,problem.x0,options);
toc
root2d(x)

end

function F = root2d(x)
F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
end
```

```
function demo_fsolve_2()
% problem setting
problem.x0=[0,0];
options = optimoptions('fsolve')
```

```
FUN=@(x)root2d(x);
tic
x = fsolve(FUN,problem.x0,options);
toc
root2d(x)

end
```

```
function F = root2d(x)
F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
end
```

Example

$$x_1^2 + x_2^2 + x_3^2 = 4$$

$$2x_1 - x_2 + x_3 = 1$$

$$x_1 + 3x_2 - x_3 = 3$$

To pass parameters using anonymous functions:

Write a file containing the following code:

```
function y = parameterfun(x,a,b,c)
y = (a - b*x(1)^2 + x(1)^4/3)*x(1)^2 + x(1)*x(2) + ...
    (-c + c*x(2)^2)*x(2)^2;
```

Assign values to the parameters and define a function handle f to an anonymous function by entering the following code in the MATLAB® prompt:

```
a = 4; b = 2.1; c = 4; % Assign parameter values
x0 = [0.5,0.5];
f = @(x)parameterfun(x,a,b,c);
```

Call the solver fminunc with the anonymous function:

```
[x,fval] = fminunc(f,x0)
```

```

function demo_fsolve3()
% problem setting
x0 = [0,0,0];
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')

display('fsolve by levenberg-marquardt');
c=[1 1 1 2 -1 1 1 3 -1];
f = @(x)root3d(x,c);
tic
x = fsolve(f,x0,options);
toc
root3d(x,c)

end

function F = root3d(x,c)
F(1) =c(1)*x(1)^2+ c(2)*x(2)^2+c(3)*x(3)^2-4;
F(2) =c(4)*x(1)+ c(5)*x(2)+c(6)*x(3)-1;
F(3) =c(7)*x(1)+ c(8)*x(2)+c(9)*x(3)-3;

end

```

$$x_1^2 + x_2^2 + x_3^2 = 4$$

$$2x_1 - x_2 + x_3 = 1$$

$$x_1 + 3x_2 - x_3 = 3$$

Neural dynamics

$$x_i = \tanh(a_i^T \mathbf{x}), \text{ for all } i$$

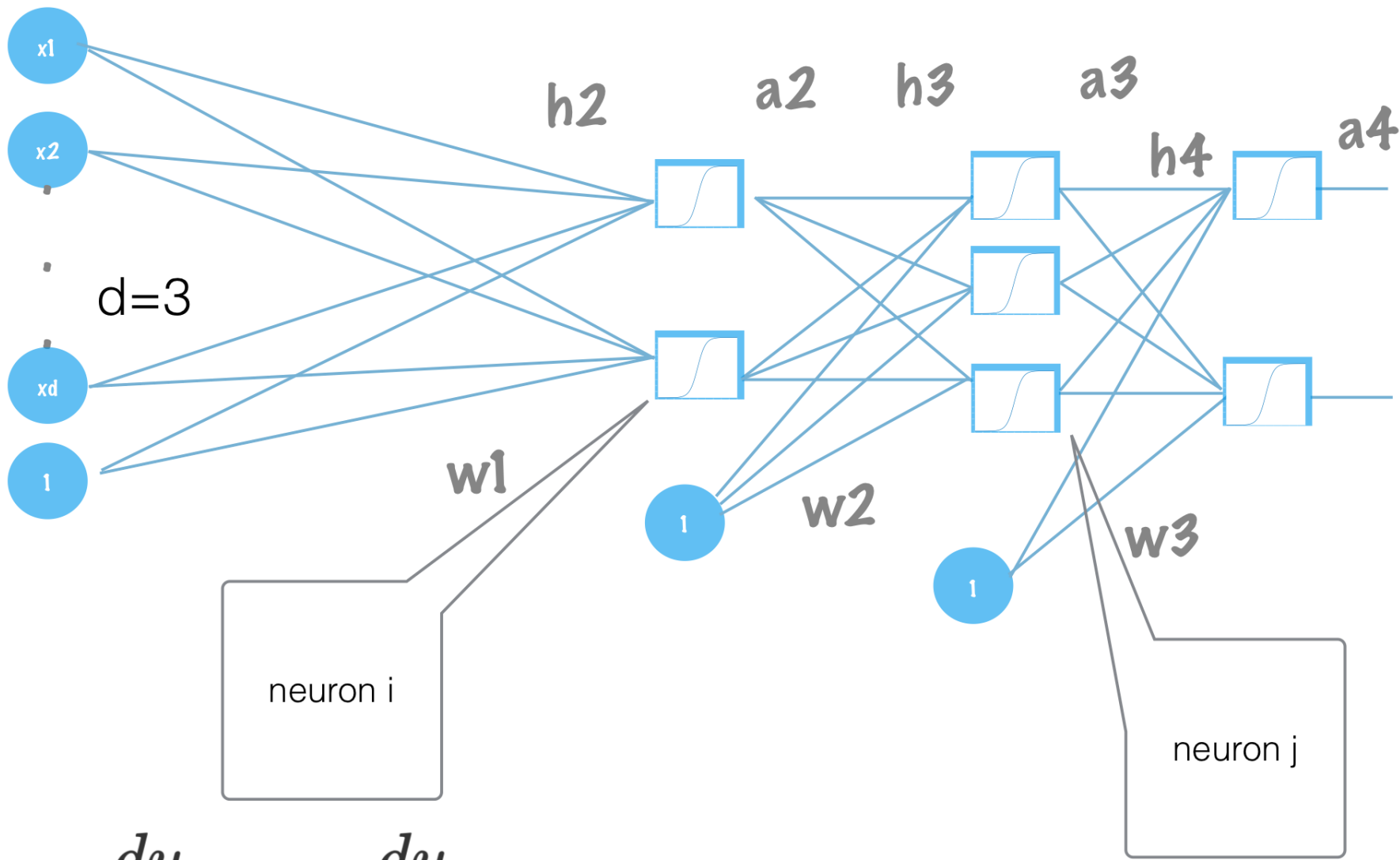
$$x_i = \tanh(a_i^T \mathbf{x}), \text{ for all } i$$

j \circ \backslash a_{ij} \circ i \oplus line r combination
 \circ \backslash \dots \circ

$$v_i = \sum_{j \neq i} a_{ij} v_j$$

$$\textcircled{2} \quad v_i = \tanh\left(\sum_{j \neq i} a_{ij} v_j\right)$$

a1



$$\frac{dy}{dh_2[i]} = ? \quad \frac{dy}{da_2[i]} = ?$$

$$\frac{dy}{dh_3[i]} = ? \quad \frac{dy}{da_3[i]} = ?$$

MatConvNet

Caffe

Deep learning framework
by [BAIR](#)

Created by

[Yangqing Jia](#)

Lead Developer

[Evan Shelhamer](#)



[View On GitHub](#)

Caffe

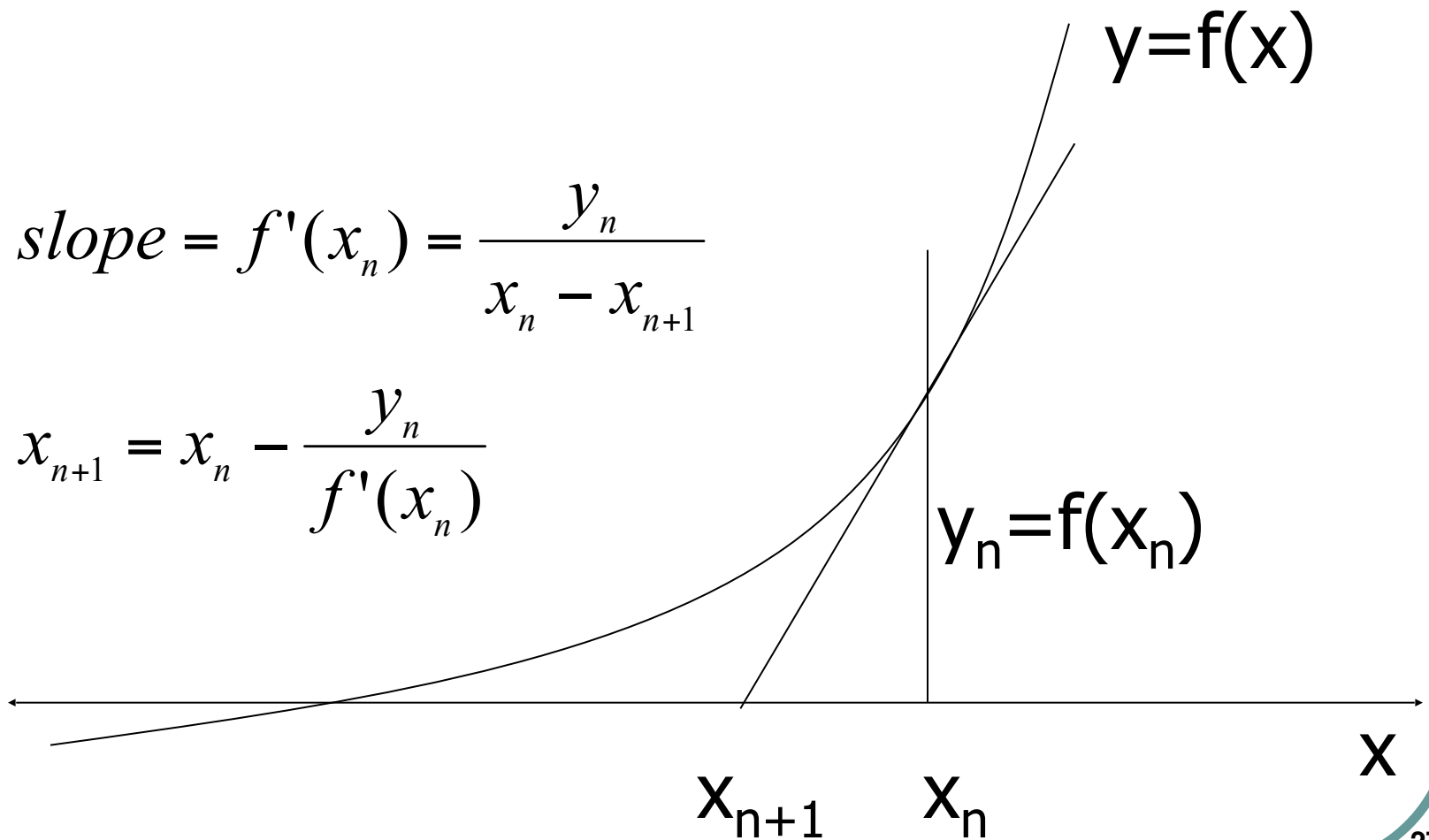
Caffe is a deep learning framework made with expression, speed and accuracy. It was developed by Berkeley AI Research ([BAIR](#)) and by community project during his PhD at UC Berkeley. Caffe is released under the BSD license.

Check out our web image classification [demo!](#)

Why Caffe?

Expressive architecture encourages application and innovation defined by configuration without hard-coding. Switch between

Newton's method -Tangent line



Updating rule

x : scalar

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

\mathbf{x} : vector

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

Taylor series

- Second order expansion at $\mathbf{x} = \mathbf{x}_n$

$$F(\mathbf{x} + \Delta\mathbf{x}) \approx F(\mathbf{x}) + J(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T H(\mathbf{x})\Delta\mathbf{x}$$

Jacobi matrix

Hessian matrix

$$\mathbf{x} \leftarrow \mathbf{x}_n, \quad \Delta\mathbf{x} \leftarrow \mathbf{x} - \mathbf{x}_n,$$

$$F(\mathbf{x}) \approx F(\mathbf{x}_n) + J(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_n)^T H(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n)$$

Hessian Matrix

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f_1(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f_1(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_1(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f_2(\mathbf{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f_2(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f_2(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

Jacobi matrix

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

First order expansion

- *Set zero to*

$$F(\mathbf{x}) \approx F(\mathbf{x}_n) + J(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n)$$

$$F(\mathbf{x}_n) + J(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n) = 0 \implies \mathbf{x} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

Newton's method

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

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

Flow Chart

```
ep=10-6; x=rand(3,1)-0.5;  
y=f(x(1),x(2),x(3)); it=0;
```

function x=Newton2(x0,s1,s2,s3)

init

~ halting
condition

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

symbols

Inline function

Jacobian

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)';

x1=sym('x1')

x2=sym('x2')

X3=sym('x3')

$$3x_1 - \cos(x_2x_3) - \frac{1}{2} = 0$$

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

$$e^{-x_1x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

inline Function

```
f=inline([str2sym(s1);str2sym(s2) ;str2sym(s  
3)]);  
f(0,0,0)
```

Jacobian

```
A=jacobian([str2sym(s1);str2sym(s2) ;str2sym(s3)],[x1 x2 x3]);
```

```
j=inline(A);
```

```
j(1,1,1)
```

Symbols, inline and Jacobian

symbols

Inline function

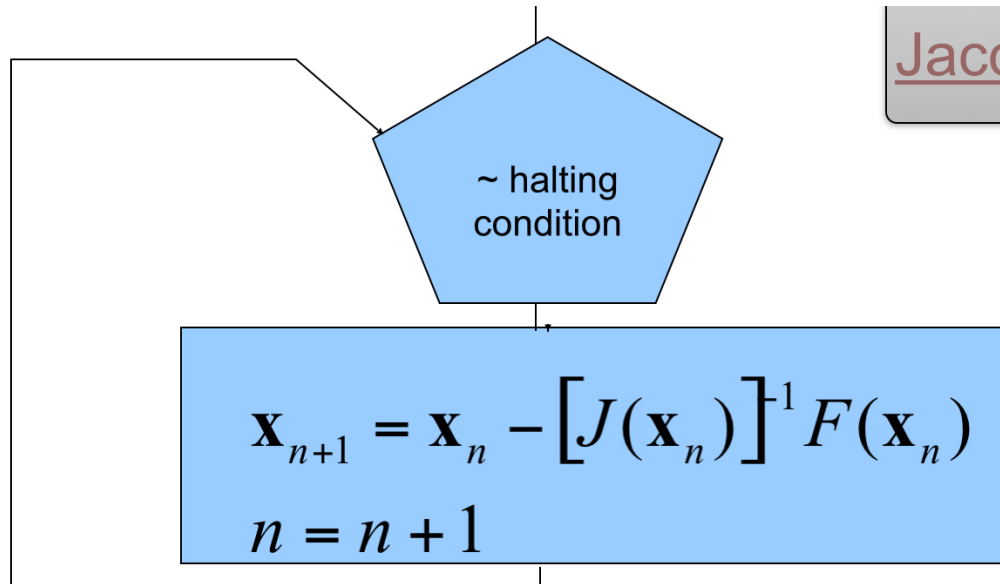
```
s1='3*x1-cos(x2*x3)-1/2';  
s2='x1^2 -81*(x2+0.1)^2+sin(x3)+1.06';  
s3='exp(-x1*x2)+20*x3+1/3*(10*pi-3)';  
x1=sym('x1');x2=sym('x2');x3=sym('x3');  
f=inline([sym(s1);sym(s2) ;sym(s3)]);  
A=jacobian([sym(s1);sym(s2) ;sym(s3)],[x1 x2 x3]);  
j=inline(A);
```

Jacobian

Init

```
ep=10-6; x=rand(3,1)-0.5;  
y=f(x(1),x(2),x(3)); it=0;
```

```
while sum(abs(y)) > ep & it < 100
x=x-inv(j(x(1),x(2),x(3)))*y;
y=f(x(1),x(2),x(3))
it=it+1
end
x
```



- Implement the Newton's method for solving a three-variable nonlinear system
- Test your matlab codes with the following nonlinear system

$$3x_1 - \cos(x_2x_3) - \frac{1}{2} = 0$$

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

$$e^{-x_1x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

$$x_1^2 + x_2^2 + x_3^2 = 4$$

$$2x_1 - x_2 + x_3 = 1$$

$$x_1 + 3x_2 - x_3 = 3$$

problem set 1

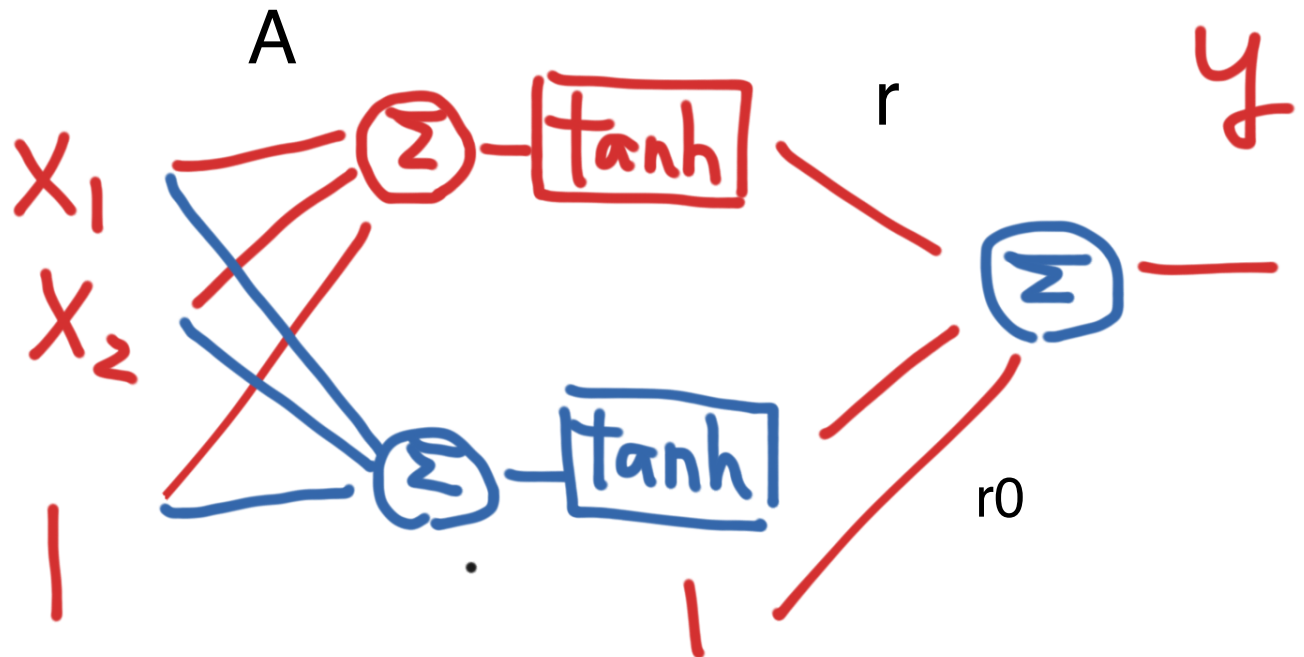
Write a matlab function to translate an input matrix to an output matrix by an MLP neural network.

Write a matlab function to implement a system of nonlinear error functions of an MLP neural network.

Try to solve nonlinear error functions by fsolve based on Levenber-Marquardt

$$Y = r * \tanh(AX) + r0$$

$$y = g(x_1, x_2) \\ = \tanh(x_1 + x_2 - 1) \\ + \tanh(x_1 - x_2 + 1)$$



$$X = \text{rand}(50, 2) * 2 * \pi - \pi$$

$$y = g(x) \% X : 2 \text{ by } 50$$

$$\% y : 1 \text{ by } 50$$

— — — — — — — — —

Let $y=f(x)$ denote a mapping realized by a deep neural network

$$f(x) = W_3 * \tanh(W_2 * \tanh(W_1 x))$$

where W_1 , W_2 and W_3 denote matrixes, x denotes a stimulus vector and y denotes an output vector. For example, x is a handwritten digit and y is a unit vector for representing a label. Consider training and testing sets of MNIST. Discuss how to train W_1 , W_2 and W_3 by the Newton method.

```

load mnist_uint8;
train_x = double(train_x) / 255;
test_x = double(test_x) / 255;
train_y = double(train_y);
test_y = double(test_y);
X=[];
for i=1:10
A = train_x;
b = train_y(:,i)==1;
AA=A'*A;
bb=A'*b;
x=CG_lin(AA,bb);
b_hat=A*x>0.5;
er_train=b-b_hat;
b_test=(test_x*x)>0.5;
er_test=b_test-test_y(:,i)==1;
fprintf('%d er_train %d er_test %d\n',i-1,sum(abs(er_train))/length(er_train),sum(abs(er_test))/
length(er_test));
X=[X x];
end
train_y_hat=train_x*X;
r=count_er(train_y,train_y_hat)

```

```

test_y_hat=test_x*X;
r=count_er(test_y,test_y_hat)

```

```
function r=count_er(y,y_hat)
[v ind_y]=max(y');
[v ind_y_hat]=max(y_hat');
r=sum(ind_y~=ind_y_hat)/length(ind_y);
```