Solving a system of nonlinear equations

Newton's method

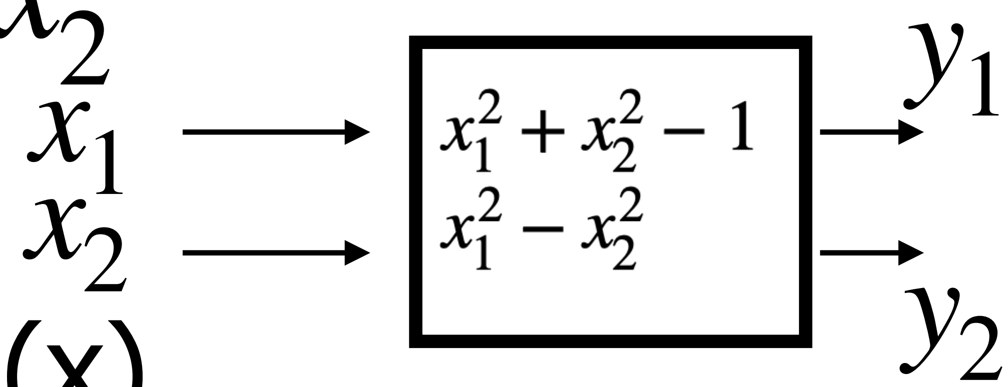
Levenberg-Marquardt method

Outline

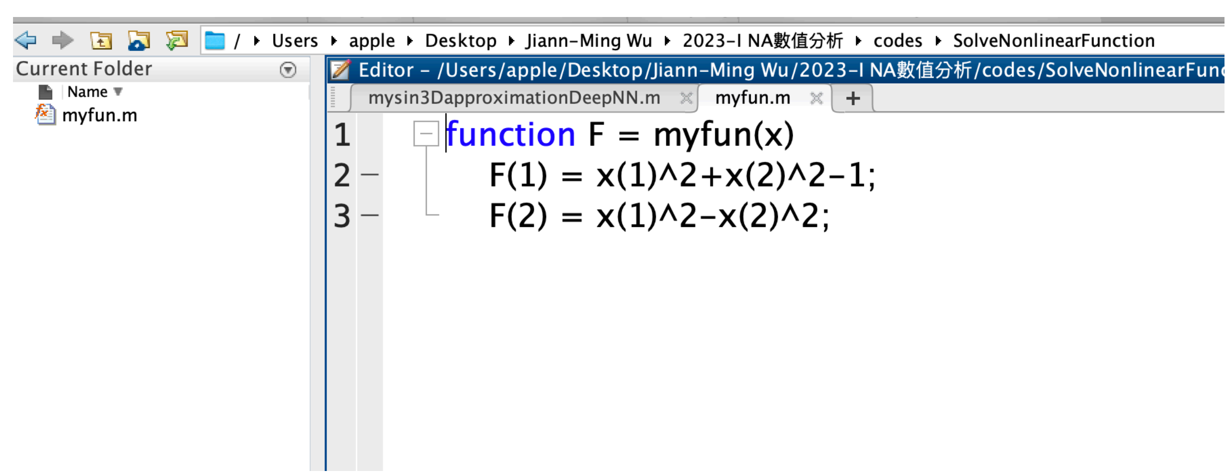
- A system of nonlinear equations
- Matlab toolbox: fsolve
- fsolve by the Levenberg-Marquardt method
- Newton's method for nonlinear system solving
  - Updating rule
  - Matlab implementation

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

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




```
function F = myfun(x)
   F(1) =  x(1)^2 + x(2)^2-1;
   F(2) =  x(1)^2 - x(2)^2;
return
```



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])
```

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0$$
$$f_2(x_1, x_2) = x_1^2 - x_2^2 = 0$$

```
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

$$e^{-e^{-(x_1+x2)}} - x_2(1 + x_1^2) = 0$$

$$x_1 cos(x_2) + x_2 sin(x_1) - 0.5 = 0$$

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. Hillstrom, *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

Nonlinear systems $\frac{df_i}{dx_j}$

MIMO: multiple input
Multiple output

A system of nonlinear equations

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

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

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
```

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

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

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

```
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
```

$$e^{-e^{-(x_1+x2)}} - x_2(1 + x_1^2) = 0$$

$$x_1 cos(x_2) + x_2 sin(x_1) - 0.5 = 0$$

Command Window

```
<stopping criteria details>

ans =

  1.0e-12 *

  -0.1875   -0.0540
```

```
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
```

	Base	Levenberg-Marquardt
Absolute sum of y	2.7894E-07	2.4147E-13
	Mean: 2.7894e-07 Var: 2.1972e-14	Mean: 3.0549e-13 Var: 1.1369e-25

effectiveness

reliability

Ten executions

Example

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

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

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

$$y = (a - bx_1^2 + x_1^{\frac{4}{3}}) * x_1^2 + x_1 x_2 + (-c + cx_2^2)x_2^2)$$

Given a, b and c, find x that minimizes y

a = 4; b = 2.1; c = 4;

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 commands at 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

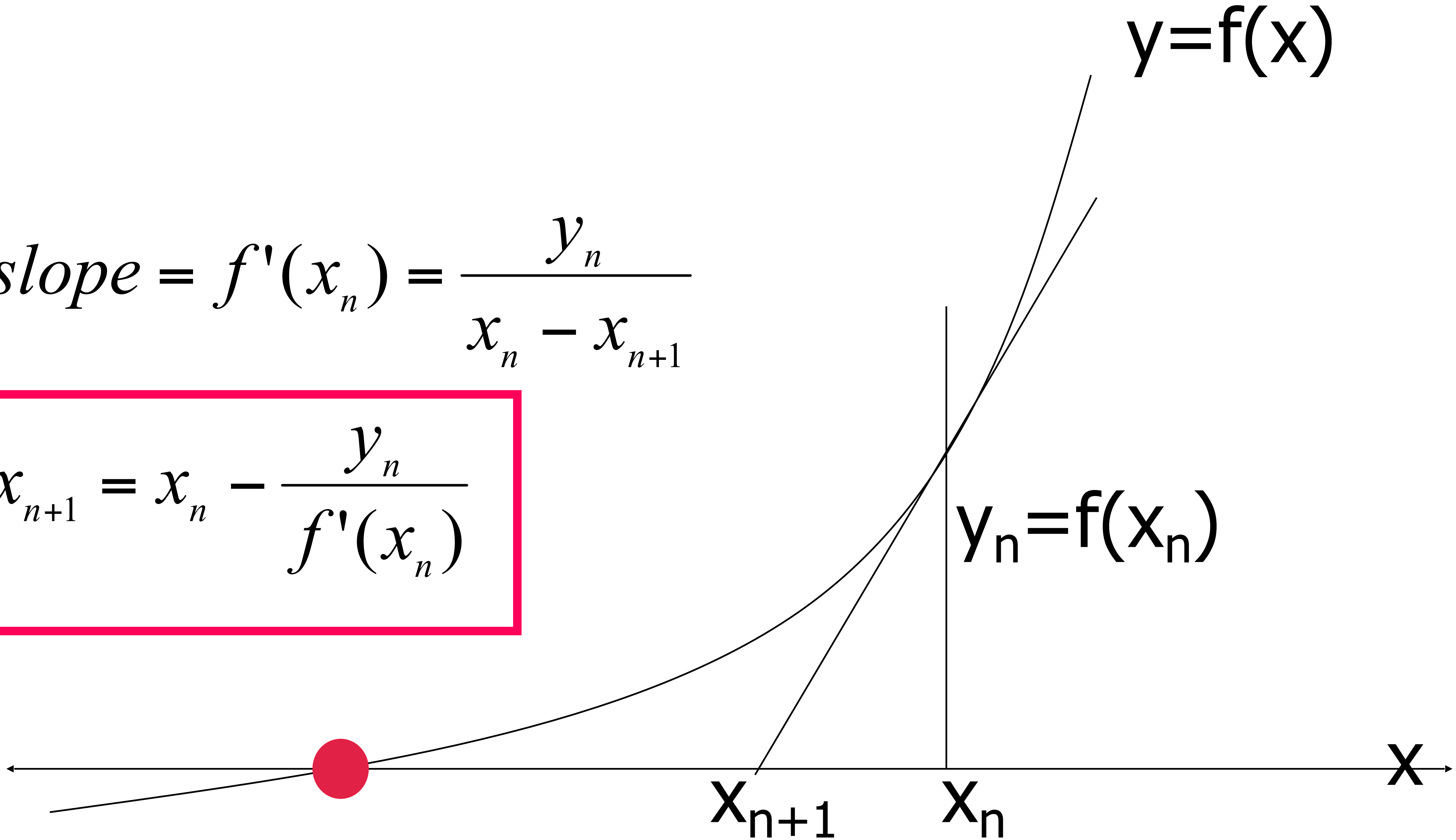
$$x_i = \tanh(a_i^T \mathbf{x}), \;\; \text{for all } i$$

Thermal
Equilibrium

Newton's method -Tangent line

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

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



Updating rule

x : scalar

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

x : vector

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

Newton Method

Taylor series

Second order expansion at x= x_n

$$F(\mathbf{x}+\Delta\mathbf{x}) \approx F(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\begin{bmatrix} \Delta^T\mathbf{x}H_1(\mathbf{x})\Delta\mathbf{x} \\ \Delta^T\mathbf{x}H_2(\mathbf{x})\Delta\mathbf{x} \\ \cdots \\ \Delta^T\mathbf{x}H_n(\mathbf{x})\Delta\mathbf{x} \end{bmatrix}$$

Jacobi matrix

Hessian matrix

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

$$F(\mathbf{x}) \approx F(\mathbf{x}_n) + \mathbf{J}(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) + \frac{1}{2}\begin{bmatrix} (\mathbf{x}-\mathbf{x}_n)^T H_1(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) \\ (\mathbf{x}-\mathbf{x}_n)^T H_2(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) \\ \cdots \\ (\mathbf{x}-\mathbf{x}_n)^T H_n(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) \end{bmatrix}$$

Jacobi matrix

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

Hessian Matrix

$$H_n(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f_n(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f_n(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_2 \partial x_n} \\ \cdots & \cdots & & \cdots \\ \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f_n(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

First order expansion

Set zero to

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

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

x : vector

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

Newton's method

$$3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$
$$x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0$$
$$e^{-x_1 x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

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

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

Problem:
1. Find the second order Taylor expansion of $E(x_1, x_2, x_3)$ in terms of Newton-Gauss Hassian matrix

$$E(x_1, x_2, x_3) = \sum_{i=1}^{3} f_i^2(x_1, x_2, x_3)$$

Try Newton-Gauss Hessian

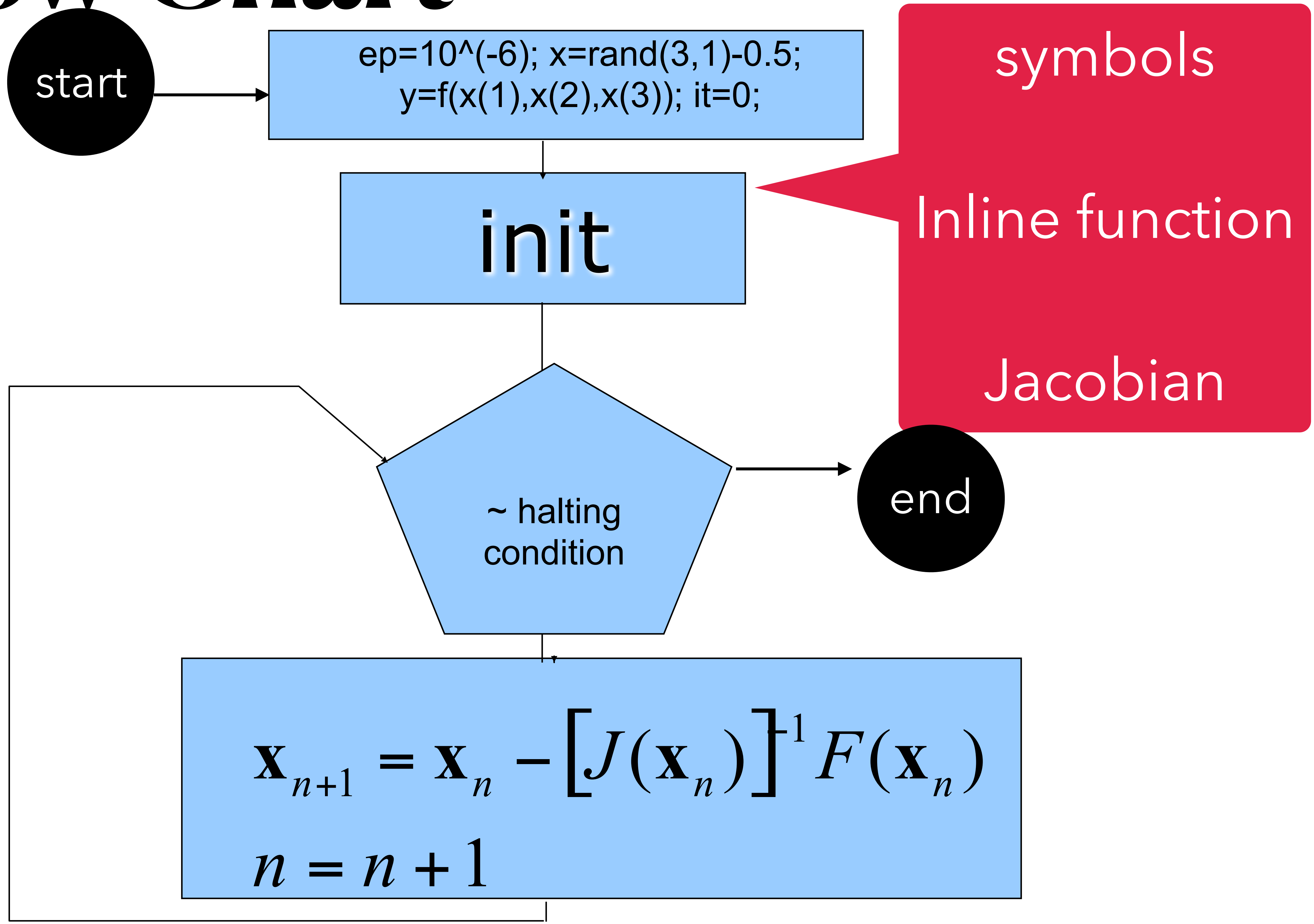
$$E(x_1, x_2, x_3) = \sum_{i=1}^{3} f_i^2(x_1, x_2, x_3)$$

Find the minimum of $E(x_1, x_2, x_3)$

Assignment:
2. State the Newton-Gauss method for solving a nonlinear system based on the approximation in problem 1.

Flow Chart

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



symbols

```
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')
```

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

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

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

inline Function

f=inline([str2sym(s1);str2sym(s2) ;str2sym(s3)]);

f(0,0,0)

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

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

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

Jaconian

```
A=jacobian([str2sym(s1);str2sym(s2) ;str2sym(s3)],[x1 x2 x3]);
j=inline(A);
j(1,1,1)
```

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

Symbols, inline and Jacobian

symbols

```
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([str2sym(s1);str2sym(s2) ;str2sym(s3)])

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

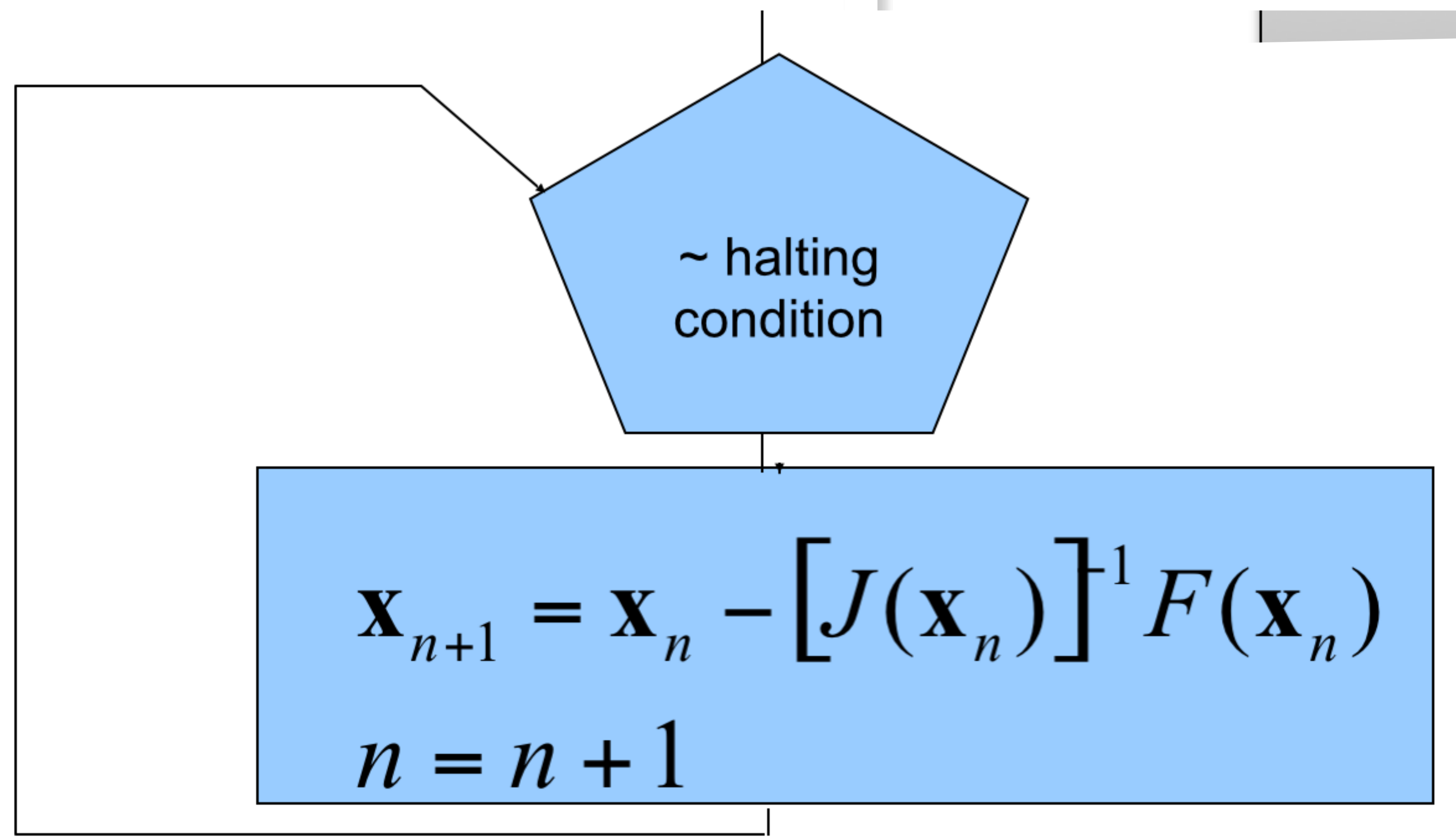
Inline function

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
```



Problem sets

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

$$3x_1 - \cos(x_2 x_3) - \frac{1}{2} = 0$$
$$x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0$$
$$e^{-x_1 x_2} + 20x_3 + \frac{1}{3}(10\pi - 3) = 0$$

$$x_1^2 + x_2^2 + x_3^2 = 4$$
$$2x_1 - x_2 + x_3 = 1$$
$$x_1 + 3x_2 - x_3 = 3$$

2. Write Matlab codes to calculate the mean square error of solving a nonlinear system

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2023-I NA數值分析 ▸ codes ▸

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/codes/demo_newton2023.m

myfun.m | root2d_a.m | my_test.m | demo_fsolve_0.m | demo_newton2023.m

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

    ep=10^(-8); 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;
```

Command Window

```
it : 1, abs sum of y :0.7417431573
 it : 2, abs sum of y :0.0820978432
 it : 3, abs sum of y :0.0018912820
 it : 4, abs sum of y :0.0000011021
 it : 5, abs sum of y :0.0000000000
```

Try Newton-Gauss Hessian

$$E(x_1, x_2, x_3) = \sum_{i=1}^{3} f_i^2(x_1, x_2, x_3)$$

Find the minimum of $E(x_1, x_2, x_3)$

Unconstrained Optimization

Nonlinear systems

A system of nonlinear equations

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

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

Write codes to calculate

$$E(x) = \frac{1}{n}\sum_{i} f_i^2(x)$$

for x derived by the Newton method

3. A. Derive the gradient of $E(x)$ with respect to each x_i.

$$\frac{dE(x)}{dx_i} = ?$$

3.B. How to express $\frac{dE(x)}{dx_i}$ in terms of elements in $J(x)$, where $J(x)$ denotes the Jacobi matrix.

$$\Delta x \propto -\frac{\partial E}{\partial x}$$

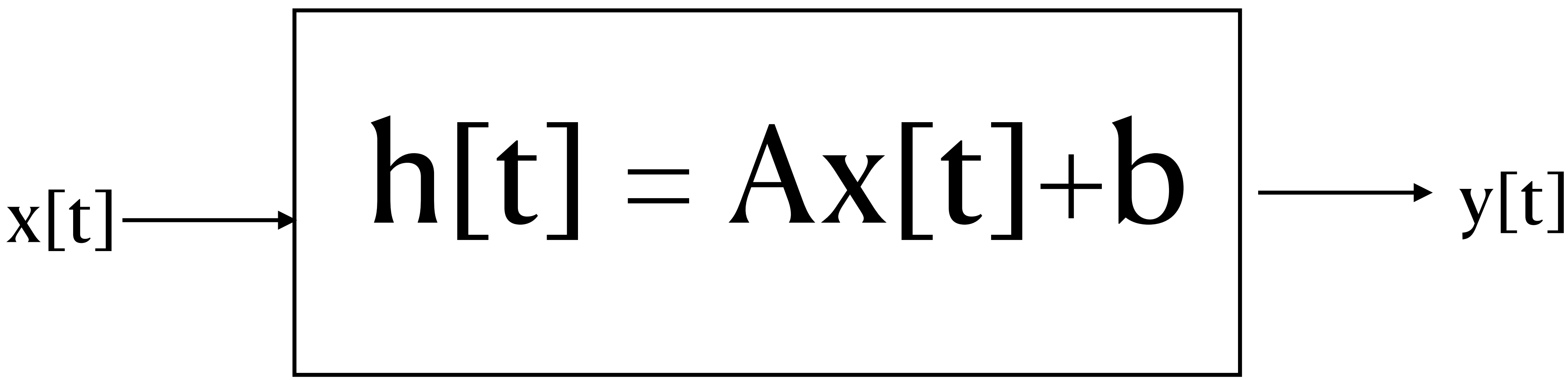
$$\Delta x = -\eta\frac{\partial E}{\partial x}, \eta > 0$$

4. Express the rule used by the gradient descent method for minimizing the mean square error $E(x)$.

5. Write codes to implement the gradient descent method for minimizing $E(x)$.

gradient descent method

Linear Transformation



Nonlinear transformation I

x[t]

$h[t] = Ax[t] + b$

$u[t] = \exp(h[t])$

$y[t] = r^T u[t] + r_0$

y[t]

Nonlinear transformation II

x[t]

↓

$h[t] = Ax[t] + b$ → $u[t] = \exp(h[t])$ → $y_i[t] = \dfrac{u_i[t]}{\sum_j u_j[t]}$

↓

y[t]

Softmax: normalization

Expectation of δ

$$= [y_1, \ldots, y_i, \ldots, y_n]^T$$

$$= [\frac{exp(u_1)}{\sum_j \exp(u_j)}, \ldots, \frac{exp(u_i)}{\sum_j \exp(u_j)}, \ldots, \frac{exp(u_n)}{\sum_j \exp(u_j)}]^T$$

Nonlinear transformation III

x[t]

$h[t] = Ax[t] + b$

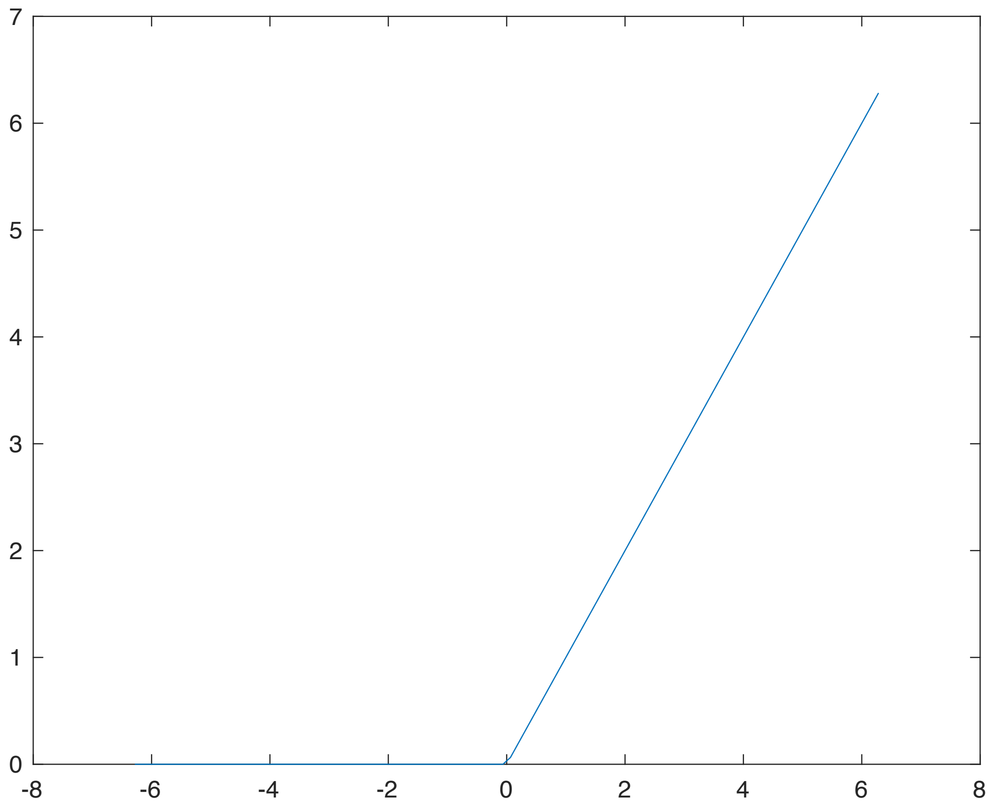
$u[t] = \tanh(h[t])$

$y[t] = r^T u[t] + r_0$

y[t]

Perceptrons

Nonlinear transformations I-III are all typical neural functions



Relu

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.

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

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

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

Try Newton-Gauss Hessian

$$E(x_1, x_2, x_3) = \frac{1}{2}\sum_{i=1}^{3} f_i^2(x_1, x_2, x_3)$$

Find the minimum of $E(x_1, x_2, x_3)$

$$E(x_1, x_2, x_3) = \frac{1}{2}\sum_{i=1}^{3} f_i^2(x_1, x_2, x_3)$$

$$\frac{dE}{dx_j} = \sum_{i=1}^{3} f_i(x)\frac{df_i}{dx_j}$$

$$x = \left[x_1, x_2, x_3\right]^T$$

gradient $\quad g(x) = \left[\frac{dE}{dx_1}, \frac{dE}{dx_2}, \ldots \frac{dE}{dx_k}, \ldots, \frac{dE}{dx_n}\right]^T$

NG-Hessian approximates

$$\frac{\partial^2 E}{\partial x_j \partial x_k} \text{ by } \frac{dE}{dx_j}\frac{dE}{dx_k}$$

$$\text{NG-Hessian Matrix} = \begin{bmatrix} \frac{dE}{dx_1} \\ \frac{dE}{dx_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{dE}{dx_j} \\ \cdot \\ \cdot \\ \cdot \\ \frac{dE}{dx_n} \end{bmatrix}$$

$$\left[\frac{dE}{dx_1}, \frac{dE}{dx_2}, \ldots \frac{dE}{dx_k}, \ldots, \frac{dE}{dx_n}\right]$$

$$\frac{\partial x_1 x_2}{\partial x_1 \partial x_2} = 1 = \frac{\partial x_2}{\partial x_2}$$

$$\frac{dx_1 x_2}{dx_1}\frac{dx_1 x_2}{dx_2} = x_2 x_1$$

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

The joint entry of the jth row and the kth column of NG-Hessian Matrix is

$$\frac{dE}{dx_j}\frac{dE}{dx_k}$$

Let $g = \left[\frac{dE}{dx_1}, \frac{dE}{dx_2}, \ldots \frac{dE}{dx_k}, \ldots, \frac{dE}{dx_n}\right]^T$ denote the gradient and H denote NG-Hessian matrix

$$E(x + \Delta x) \approx E(x) + g^T \Delta x + \frac{1}{2}\Delta x^T H \Delta x$$

Set $L(x) = E(x) + g\Delta x + \frac{1}{2}\Delta x^T H \Delta x = 0$

$$H\Delta x = -g$$

$$\Delta x = \boxed{-H^{-1}g}$$

Newton-Gauss method

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

$$\frac{dE}{dx_j} = \sum_{i=1}^{3} f_i(x) \frac{df_i}{dx_j}$$

g(1:3) = F*J(:,1:3)

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2023-I NA數值分析 ▸ codes ▸

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/codes/demo_NG_Newton.m*

myfun.m | root2d_a.m | my_test.m | demo_fsolve_0.m | demo_newton2023.m | demo_NG_Newton.m*

Newton-Gauss method

```
y=f(x(1),x(2),x(3)); it=0;
while sum(abs(y)) > ep & it < 100
    g = y * j(x(1),x(2),x(3));
    H = transpose(g) * g;
    delta_x = -inv(H) * g;
    % x = x - inv(j(x(1),x(2),x(3))) * y;
    x = x + delta_x;
    y = f(x(1),x(2),x(3));
    it = it + 1;
    fprintf(' it : %d, abs sum of y :%12.10f\n ', it, sum(abs(y)))

end
```

$$\Delta x = -(H + \lambda I)^{-1} g$$

Levenberg-Marquardt method

alpha

- $\alpha = \dfrac{E(x) - E(x + \Delta x)}{E(x) - L(x + \Delta x)}$
- $E(x) - L(x + \Delta x) = E(x) - (E(x) + g\Delta x + \dfrac{1}{2}\Delta x^T H \Delta x)$
- $= -g\Delta x - \dfrac{1}{2}\Delta x^T H \Delta x)$

Actual cost reduction

Predicted cost reduction

$$\Delta x = -(H + \lambda I)^{-1} g$$

High α

Reduce λ

improve efficiency

Force to Newton-Gauss method

$$\Delta x = -(H + \lambda I)^{-1} g$$

Low α

Increase λ

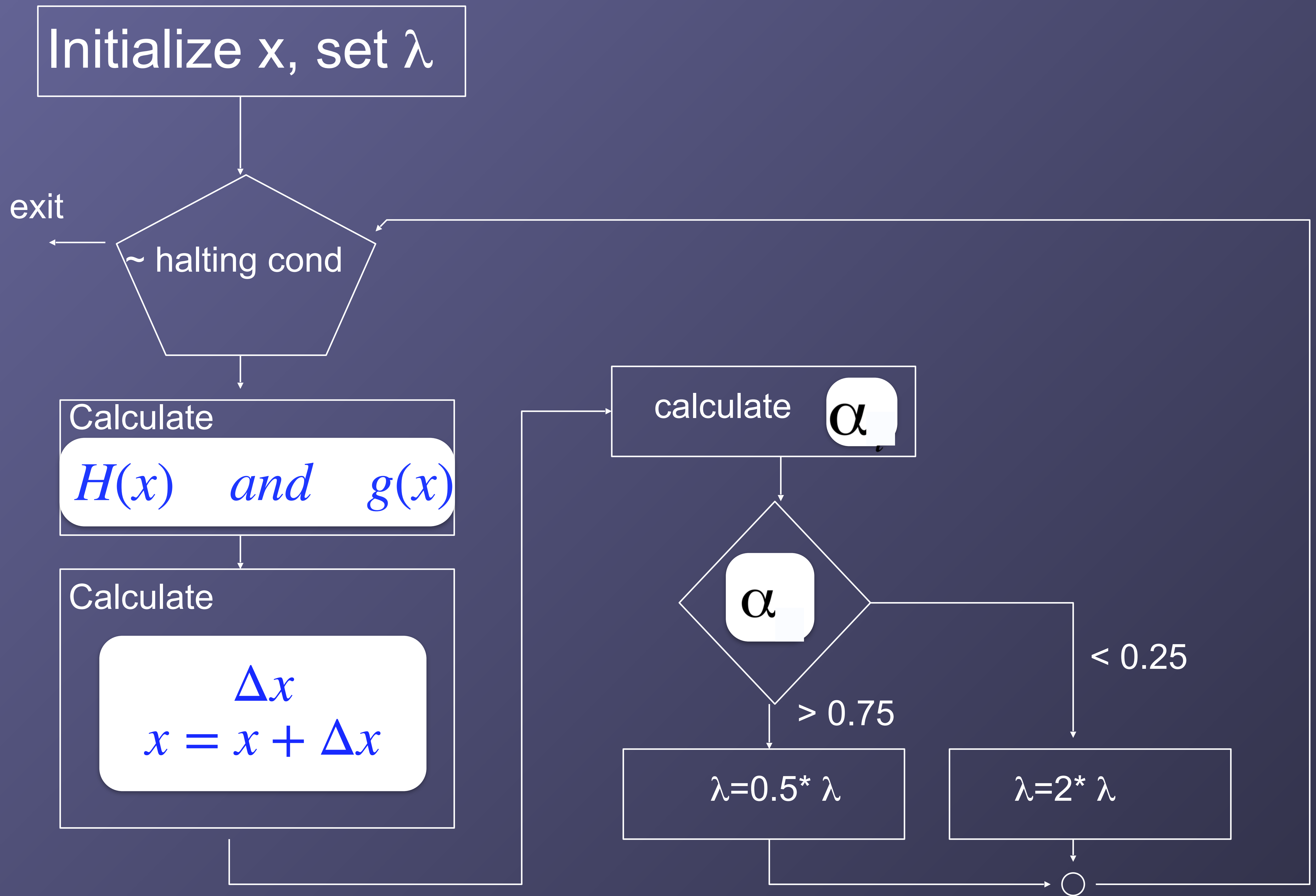
Improve reliability

Force to gradient method

Heuristic adaption

(a) If $\alpha > 0.75$, $\lambda \leftarrow 0.5\lambda$.

(b) If $\alpha < 0.25$, $\lambda \leftarrow 2\lambda$.



Initialize x, set λ
exit
~ halting cond
Calculate
$H(x)$ and $g(x)$
Calculate
Δx
$x = x + \Delta x$
calculate α
α
> 0.75
< 0.25
λ=0.5* λ
λ=2* λ

Rejection

- If $E(x) \leq E(x + \Delta x)$, move to $x + \Delta x$ is rejected
- Define matlab function to calculate E(x)
- Define matlab function to calculate α

Implement the LM method for solving a nonlinear system