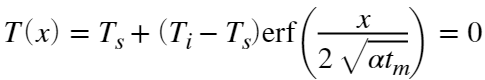
usage: teamwork for week 5

author: 黄哲昊 毛晨光 鲁潇阳

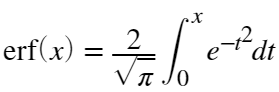
date: 2020.4.4

clear all; close all; clc

**1. 请查阅Ridders方法****， 并应用此算法求解以下方程, 请将你的结果和Matlab内置函数fzero的结果比较：**



**其中****是随位置****(用米计算)变化的温度函数，参数****. 设起始温度****， 极限温度****。时间间隔****天。误差函数****的定义为**



解：

clear all;

format long

fun = @(t)(exp(-t.^2))

fun = *包含以下值的 function\_handle:*

@(t)(exp(-t.^2))

erf = @(x)(2/sqrt(pi)\*integral(fun, 0, x))

erf = *包含以下值的 function\_handle:*

@(x)(2/sqrt(pi)\*integral(fun,0,x))

T = @(x)(-150 + (200 + 150)\*erf(x./(2\*sqrt(0.138\*(10^(-6))\*60\*24\*3600))))

T = *包含以下值的 function\_handle:*

@(x)(-150+(200+150)\*erf(x./(2\*sqrt(0.138\*(10^(-6))\*60\*24\*3600))))

x\_1\_check = fzero(T, 0)

x\_1\_check =

0.676961854481937

通过内置函数fzero求得解为0.676961m，故取Ridders方法的两个初始值分别为

[x\_1, flag\_1] = Ridders(T, 0, 1, 1e-6, 10000)

Ridders method:

the root of f(x)=0 is x = 0.676962, the number of iterations is 10

x\_1 =

0.676961854481937

flag\_1 =

10

e\_r\_1 = relative\_e(x\_1, x\_1\_check)

相对误差为: 0.000000e+00

e\_r\_1 =

0

分析相对误差可以得到，通过Ridders方法求得的解和内置函数fzero求得的解基本相同，均为0.676961m

**2. 一个引人注意的想法是结合二分法和牛顿法 - 先应用二分法， 当达到一个设定的小区间后改用牛顿法来加快收敛速度。**

**请编写函数脚本`bisection\_Newton.m`实现这样的算法： 设置初始试探区间****, 当使用二分法达到的试探区间是原来的一部分** **(即区间长度为****) 后采用牛顿法。 考虑到牛顿法有可能发散， 所以当牛顿法的结果使得试探区间在二分法结果之外时， 应该重新使用二分法。** **将作为函数的输入变量， 它的缺省值为0.1.**

**尝试用你的函数在初始区间****求解****.**

解：

考虑弧度制的区间作图容易发现不仅两个端点符号相同，而且在区间内函数并不连续，无法利用零点存在定理求解。故采用角度值进行求解。

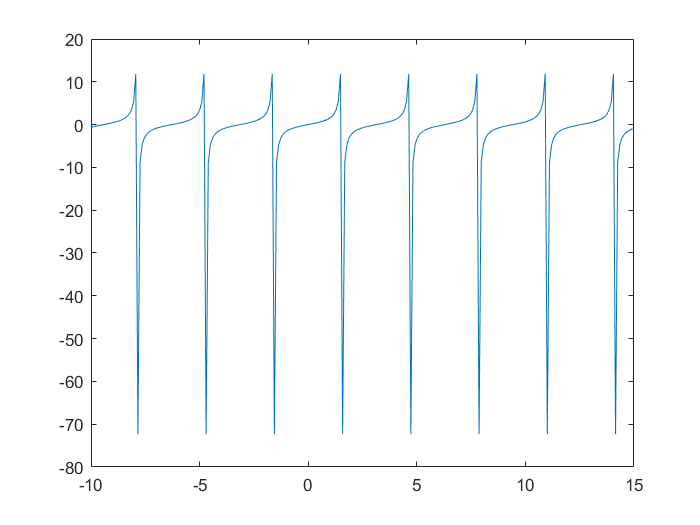
clear all;

sample\_2 = -10:pi/32:15

sample\_2 = 1×255

-10.000000000000000 -9.901825229575319 -9.803650459150639 ⋯

plot(sample\_2, tan(sample\_2))



high = -10 / 180 \* pi

high =

-0.174532925199433

low = 15 / 180 \* pi

low =

0.261799387799149

[x\_2\_1, flag\_2\_1] = bisection\_Newton(@tan, 1e-6, 10000, high, low)

bisection\_Newton:

s: 0.100000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

x\_2\_1 =

-1.664475681382791e-07

flag\_2\_1 =

19

% s取不同值时的迭代次数

plot\_bisection\_Newton\_s(@tan, 1e-6, 10000, high, low, 20)

bisection\_Newton:

s: 0.050000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.100000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.150000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.200000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.250000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.300000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.350000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.400000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.450000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.500000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.550000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.600000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.650000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.700000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.750000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.800000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.850000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 0.900000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

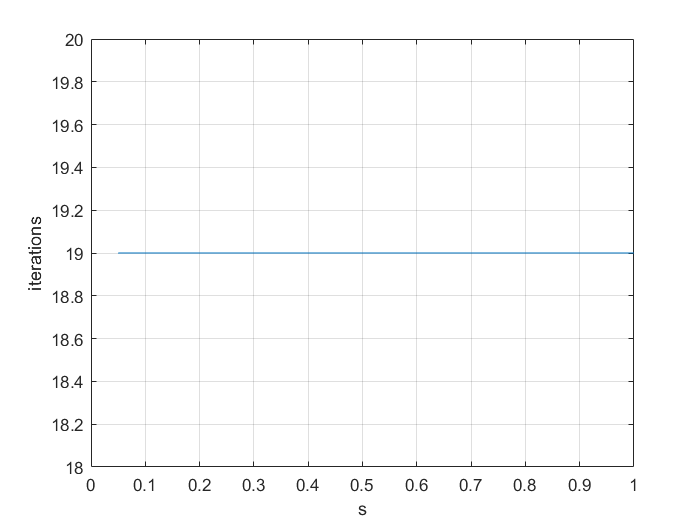
s: 0.950000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19

bisection\_Newton:

s: 1.000000

the root of f(x)=0 is x = -0.000000, the number of iterations is 19



求解得

从图可得事实上s的选取，对在的邻域内求解的迭代次数基本没有影响，均为19次。

**3. 运行以下程序，描述该程序解决的问题， 并在每行加入注释。**

解：

首先绘制两条曲线以及进行观察。

clear all;

sample\_3 = (-3:0.1:2)

sample\_3 = 1×51

-3.000000000000000 -2.900000000000000 -2.800000000000000 ⋯

f\_3\_1 = @(x)-((24.43+1.8\*x.^2)/3.2).^(1/3)

f\_3\_1 = *包含以下值的 function\_handle:*

@(x)-((24.43+1.8\*x.^2)/3.2).^(1/3)

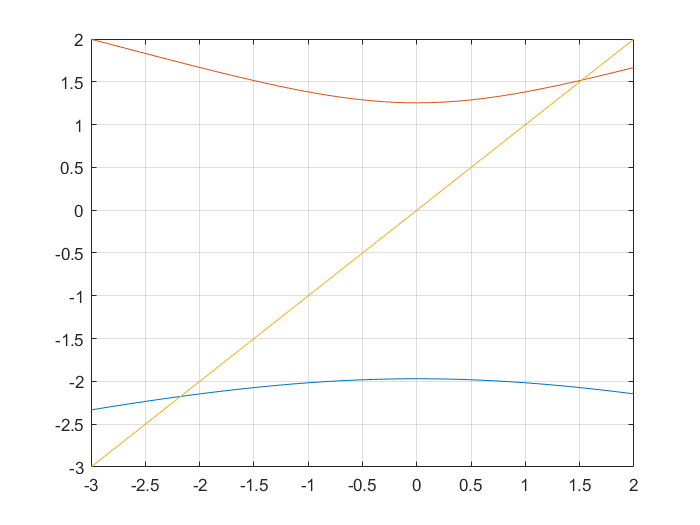
f\_3\_2 = @(x)((5.92+2\*x.^2)/3).^(1/3)

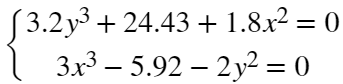
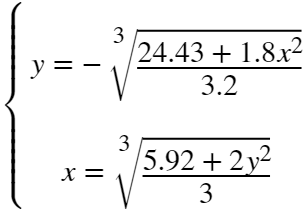
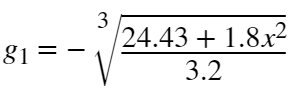
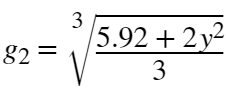
f\_3\_2 = *包含以下值的 function\_handle:*

@(x)((5.92+2\*x.^2)/3).^(1/3)

plot(sample\_3, f\_3\_1(sample\_3), sample\_3, f\_3\_2(sample\_3), sample\_3, sample\_3)

grid on



程序事实上是对非线性方程组，应用不动点迭代的方法求得数值解。首先将两式化为，然后令 ， 通过迭代公式，，并通过误差式 判断是否达到误差上限，或当迭代次数达到最多次数限制时，停止迭代。

白色的地图

描述已自动生成

clear all;

g1 = @(v) -((24.43+1.8\*v(2,1).^2)/3.2).^(1/3) % 迭代式g1

g1 = *包含以下值的 function\_handle:*

@(v)-((24.43+1.8\*v(2,1).^2)/3.2).^(1/3)

g2 = @(v) ((5.92+2\*v(1,1).^2)/3).^(1/3) % 迭代式g2

g2 = *包含以下值的 function\_handle:*

@(v)((5.92+2\*v(1,1).^2)/3).^(1/3)

tol = 1e-3; kmax = 10 % tol为误差上限，kmax为最多迭代次数

kmax =

10

v(:,1) = [-1;-2] % 初始估计值v\_0 = [y\_0, x\_0]

v = 2×1

-1

-2

for k = 1:kmax

% 第k次迭代

% 对应y\_k = g1(x\_k-1), x\_k = g2(y\_k-1)

v(:,k+1) = [g1(v(:,k)); g2(v(:,k))];

% 通过求2-范数判断是否达到误差上限

if norm(v(:,k+1)-v(:,k)) < tol

break

end

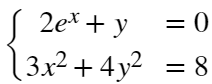
end

% 显示通过定点迭代得到的结果

fprintf('y\_0 = %f, x\_0 = %f', v(1,k+1), v(2, k+1));

y\_0 = -2.099834, x\_0 = 1.700047

**4.应用定点迭代法求解非线性方程组**



**提示： 1) 先绘制函数曲线， 确定根的大致位置。 2) 选择适当的辅助函数。 3) 请将结果与Matlab内置函数fsolve的求解结果比较.**

解：

（1）

clear all;

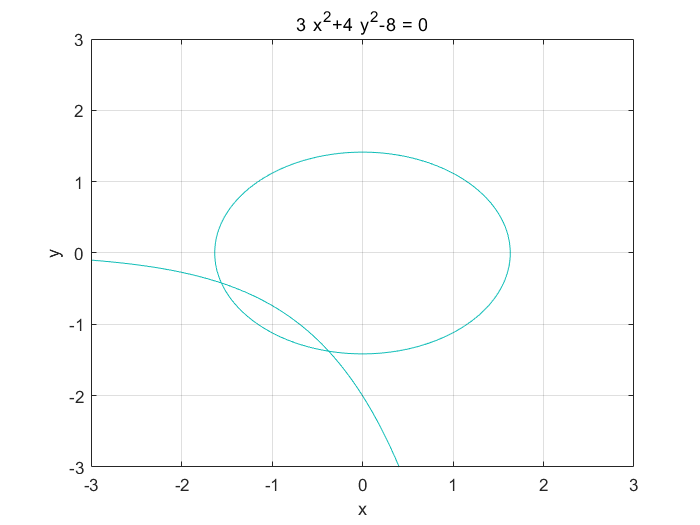
ezplot('2\*exp(x)+y')

hold on

ezplot('3\*x^2+4\*y^2-8')

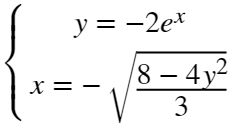
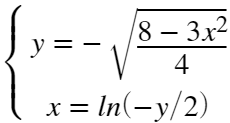
axis([-3 3 -3 3])

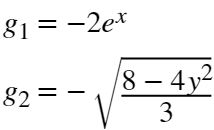
grid on

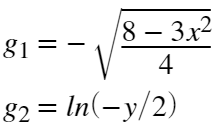


通过绘制两条函数曲线可以得到方程组在区间和各有一解。

（2）

将方程组化为，以及

对第一个解：分别令，选取初始点；

对第二个解：分别令，选取初始点；

通过迭代公式，的定点迭代法求解。

format long

clear all;

g\_y1 = @(v)(-2\*exp(v));

g\_x1 = @(v)(-sqrt((8-4\*v.^2)/3));

G1 = {g\_y1 g\_x1};

x0\_4\_1 = [0;-2];

[x\_4\_1, flag\_4\_1] = fixpoint\_equation\_set(G1, x0\_4\_1, 1e-6, 10000)

fixpoint\_equation\_set:

the root of g(x)=x is

x\_4\_1 = 2×1

-0.420659143653259

-1.559079213743325

flag\_4\_1 =

15

g\_y2 = @(v)(-sqrt((8-3\*v.^2)/4));

g\_x2 = @(v)(log(-v./2));

G2 = {g\_y2 g\_x2};

x0\_4\_2 = [-1, -1];

[x\_4\_2, flag\_4\_2] = fixpoint\_equation\_set(G2, x0\_4\_2, 1e-6, 10000)

fixpoint\_equation\_set:

the root of g(x)=x is

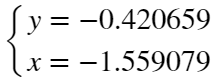
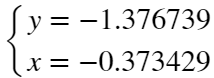
x\_4\_2 = 2×1

-1.376739900121342

-0.373429198846348

flag\_4\_2 =

16

通过定点迭代得到两个解分别为和

（3）

format long

clear all;

options = optimoptions('fsolve','Display',"iter");

fun = @root2d;

x0\_1 = [0,-2];

x\_4\_check\_1 = fsolve(fun, x0\_1, options)

Norm of First-order Trust-region

Iteration Func-count f(x) step optimality radius

0 3 64 128 1

1 6 1.41347 0.559017 14.2 1

2 9 0.00757324 0.156926 0.952 1.4

3 12 1.07007e-06 0.0174655 0.0112 1.4

4 15 3.45203e-14 0.000235482 2e-06 1.4

5 18 7.90833e-29 4.37002e-08 9.74e-14 1.4

Equation solved.

fsolve completed because the vector of function values is near zero

as measured by the value of the function tolerance, and

the problem appears regular as measured by the gradient.

<stopping criteria details>

x\_4\_check\_1 = 1×2

-0.373428741127585 -1.376740074042629

x0\_2 = [-2, 0];

x\_4\_check\_2 = fsolve(fun, x0\_2, options)

Norm of First-order Trust-region

Iteration Func-count f(x) step optimality radius

0 3 16.0733 47.9 1

1 6 0.730132 0.49128 8.54 1

2 9 0.00185126 0.115382 0.403 1.23

3 12 4.01899e-08 0.00772724 0.00187 1.23

4 15 1.66867e-17 3.50118e-05 3.81e-08 1.23

Equation solved.

fsolve completed because the vector of function values is near zero

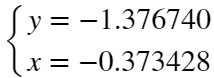
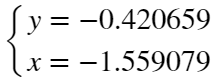
as measured by the value of the function tolerance, and

the problem appears regular as measured by the gradient.

<stopping criteria details>

x\_4\_check\_2 = 1×2

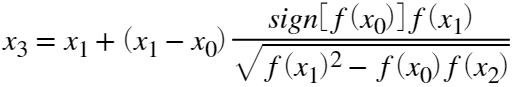
-1.559079091427916 -0.420659352712557

通过matlab内置fsolve函数求得方程组两个解分别为和，证明定点迭代法求得的解基本正确。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*functions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ridders：Ridders方法

given two values  and ，，calculate the midpoint.

a new value between and，，use  and whichever of  and  has function value of opposite sign to  in the next step of the iteration.

args：

f：需要寻根的函数

x0：初始值1

x2：初始值2

epsilon：误差上限

max\_n：最多计算的次数

output：

当，表示可能不收敛或最多迭代次数不足以达到精度，；

当，表示初始区间不存在根，；

其他情况下，表示迭代次数，为寻得的根。

function [answer, flag] = Ridders(f, x0, x2, epsilon, max\_n)

fprintf('Ridders method:\n');

count = 0; % 迭代次数

if(f(x0)\*f(x2) > 0) % 如果符号相同则无法求解

fprintf('区间内不存在根！');

elseif(f(x0) == 0)

count = -1; answer = x0;

elseif(f(x2) == 0)

count = -1; answer = x2;

else %判断迭代次数小于最多计算次数；同时是否满足误差

while(count < max\_n && abs(x0 - x2) > epsilon)

count = count + 1;

x1 = (x0 + x2) / 2;

if(f(x1) == 0)

x0 = x1; x2 = x1; break;

end

x3 = x1 + (x1 - x0)\*sign(f(x0))\*f(x1)/sqrt(f(x1)^2-f(x0)\*f(x2));

if(f(x3) == 0)

x0 = x3; x2 = x3; break;

end

% 判断下一步迭代的x0和x2

if(f(x3)\*f(x0) > 0 && f(x3)\*f(x2) > 0)

if(f(x1)\*f(x0) <= 0)

x2 = x1;

else

x0 = x1;

end

elseif(f(x3)\*f(x0) < 0)

x2 = x3;

elseif(f(x3)\*f(x1) < 0)

x0 = x1;

x2 = x3;

elseif(f(x3)\*f(x2) < 0)

x0 = x3;

end

end

end

%

switch count

case max\_n

flag = -1; answer = NaN;

fprintf('it may not converge or the iterations are not enough!\n')

case 0

flag = -2; answer = NaN;

otherwise

flag = count; answer = (x0 + x2) / 2;

fprintf('the root of f(x)=0 is x = %f, the number of iterations is %d\n', answer, count);

end

end

bisection\_Newton：设置初始试探区间，当使用二分法达到的试探区间是原来的一部分即区间长度为后采用牛顿法。考虑到牛顿法可能发散，所以当牛顿法的结果在二分法试探区间之外时，应该用二分法的结果代替牛顿法。

args：

f：寻根的函数

epsilon：误差上限

max\_n：最多计算步数

high：二分法区间上限

low：二分法区间下限

s：二分法试探程度 default：0.1

output：

当，表示可能不收敛或最多迭代次数不足以达到精度，；

当，表示初始区间不存在根，；

其他情况下，表示迭代次数，为寻得的根。

function [answer, flag] = bisection\_Newton(f, epsilon, max\_n, high, low, s)

fprintf('bisection\_Newton:\n');

if nargin < 6 % nargin:函数输入变量的个数

s = 0.1;

end

fprintf('s: %f\n', s);

%计算一阶导数

syms x;

fun = f(x);

df = matlabFunction(diff(fun));

%求根

count = 0;

length = s\*abs(high - low);

if(f(high)\*f(low) > 0)

fprintf('二分法区间不存在根！');

elseif(f(high) == 0)

count = -1; answer = high;

elseif(f(low) == 0)

count = -1; answer = low;

else

mid = (high + low) / 2;

while(count < max\_n && abs(high - low) > length && f(mid) ~= 0)

count = count + 1;

if(f(high)\*f(mid) < 0)

low = mid;

else

high = mid;

end

mid = (high + low) / 2;

end

while(count < max\_n && abs(high - low) > epsilon && f(mid) ~= 0)

count = count + 1;

mid = mid - f(mid)/df(mid);

% 如果迭代结果在二分范围外则换为(high + low)/2，并根据迭代结果更新high和low

if(mid > high || mid < low)

mid = (high + low) / 2;

end

if(f(high)\*f(mid) < 0)

low = mid;

else

high = mid;

end

end

end

switch count

case max\_n

flag = -1; answer = NaN;

fprintf('it may not converge or the iterations are not enough!\n')

case 0

flag = -2; answer = NaN;

otherwise

flag = count; answer = mid;

fprintf('the root of f(x)=0 is x = %f, the number of iterations is %d\n', answer, count);

end

end

plot\_bisection\_Newton\_s：观察不同s对收敛次数的影响

args：

f：寻根的函数

epsilon：误差上限

max\_n：最多计算步数

high：二分法区间上限

low：二分法区间下限

n：采样点的个数即每个点的间隔为

function plot\_bisection\_Newton\_s(f, epsilon, max\_n, high, low, n)

s = 1/n:(1/n):1;

flag = zeros(size(s));

for i=1:length(s)

[~, flag(i)] = bisection\_Newton(f, epsilon, max\_n, high, low, s(i));

end

plot(s, flag)

xlabel('s')

ylabel('iterations')

grid on

end

计算相对误差的函数，并显示

args:

x\_approx：估计值

x\_real：真值

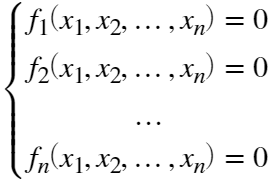
function output = relative\_e(x\_approx, x\_real)

output = (x\_approx - x\_real .\* ones(size(x\_approx)))./x\_real;

fprintf("相对误差为: %e" , output);

end

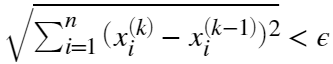
fixpoint\_equation\_set：应用定点迭代法求解非线性方程组



args：

G：一个函数向量（元胞数组），其中函数满足

v：初始迭代向量

epsilon：误差上限，当满足时，停止迭代

max\_n：最多迭代次数

output：

当，表示可能不收敛或最多迭代次数不足以达到精度，；

其他情况下，表示迭代次数，为寻得的根。

function [answer, flag] = fixpoint\_equation\_set(G, x0, epsilon, max\_n)

fprintf('fixpoint\_equation\_set:\n');

count = 0;

v(:,1) = x0;

while(count < max\_n)

count = count + 1;

% 对于每个g函数，分别进行迭代

for k = 1:length(G)

index = 1:length(G);

index(:,k)=[]; % 清除第k列

g = G{k}; % 令g对于g\_k

v(k, count+1) = g(v(index, count)); % v的第k行第c+1列=g\_k(v的第c列除去第k行)

end

if norm(v(:, count+1)-v(:, count)) < epsilon

break

end

end

if(count == max\_n)

flag = -1; answer = NaN;

fprintf('it may not converge or the iterations are not enough!\n');

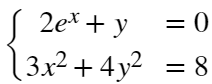
else

flag = count; answer = v(:,count+1);

fprintf('the root of g(x)=x is\n');

end

end

root2d：第四题用于fsolve求解的输入函数

function F = root2d(x)

F(1) = 2\*exp(x(1)) + x(2);

F(2) = 3\*x(1)^2 + 4\*x(2)^2 - 8;

end