# 1. 画出函数图像

clc,clear

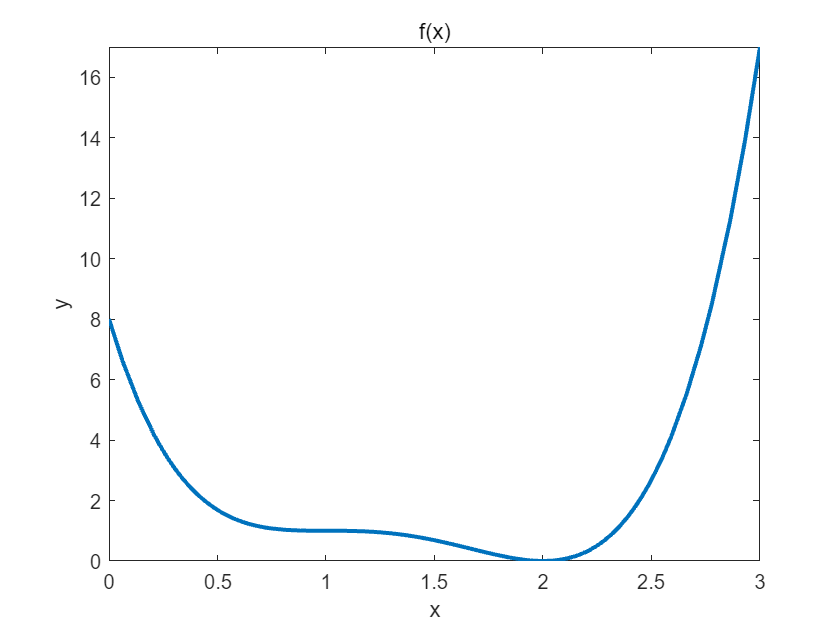
f=@(x)3\*x.^4-16\*x.^3+30\*x.^2-24\*x+8;

fplot(f,[0 3],'LineWidth',2)

title('f(x)')

xlabel('x');

ylabel('y');



# 2. 用二分法求该函数极小值

a=0;b=3;tol=1e-6;iter\_max=1e+5;

syms x;

df=diff(f,x);

df=matlabFunction(df);

[x,f\_val,iter\_time,tol\_list]=binary\_search\_min(f,df,a,b,iter\_max,tol);

disp("二分法：")

二分法：

fprintf('找到的近似零点: %.6f\n', x);

找到的近似零点: 2.000000

fprintf('函数极小值: %.6f\n', f\_val);

函数极小值: 0.000000

fprintf('迭代次数: %d\n', iter\_time);

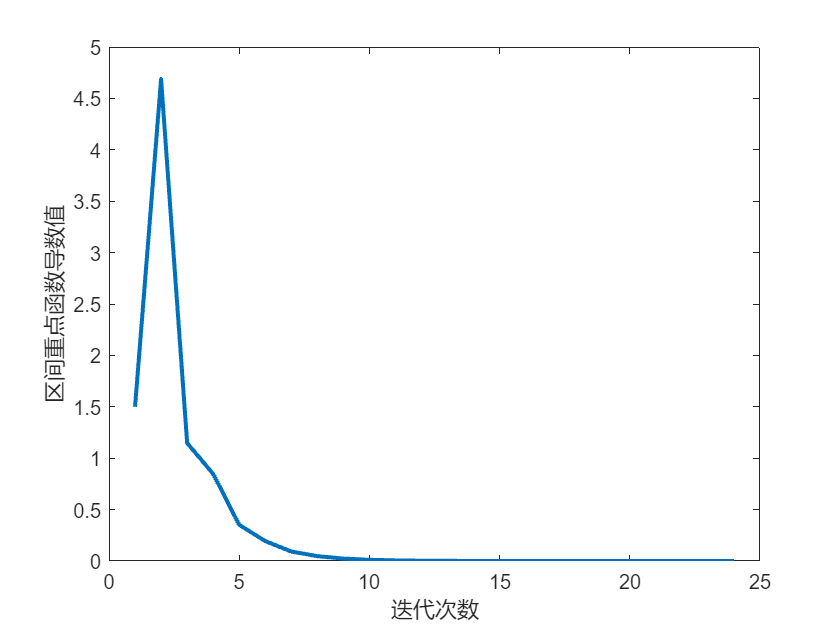
迭代次数: 24

%展示算法的极小化过程

plot([1:iter\_time],tol\_list,'LineWidth',2)

xlabel("迭代次数")

ylabel("区间重点函数导数值")



# 3. 用三分法求该函数极小值

[x,f\_val,iter\_time,tol\_list]=ternarySearchMin(f,a,b,iter\_max,tol);

disp("三分法")

三分法

fprintf('找到的近似零点: %.6f\n', x);

找到的近似零点: 2.000000

fprintf('函数极小值: %.6f\n', f\_val);

函数极小值: 0.000000

fprintf('迭代次数: %d\n', iter\_time);

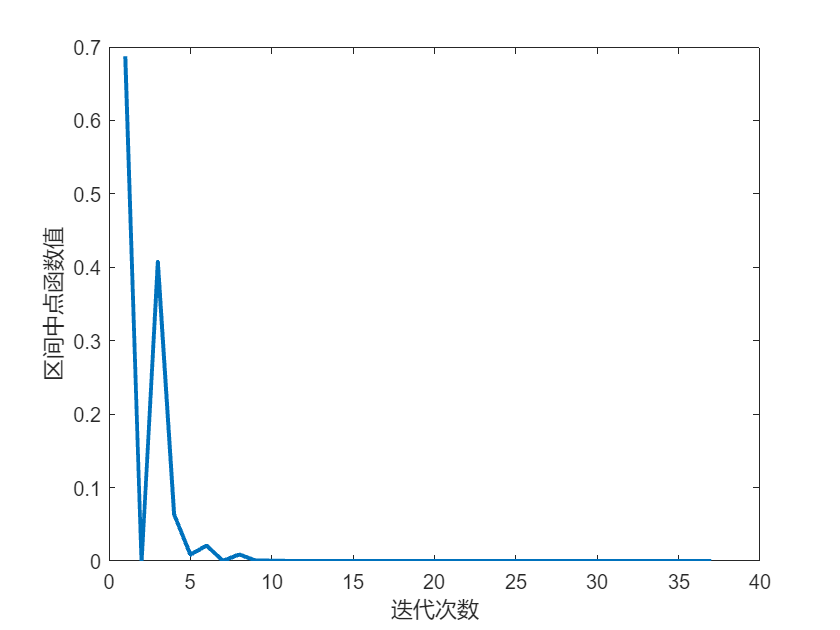
迭代次数: 37

%展示算法的极小化过程

plot([1:iter\_time],tol\_list,'LineWidth',2)

xlabel("迭代次数")

ylabel("区间中点函数值")



# 4.用黄金分割法求该函数极小值

[x,f\_val,iter\_time,tol\_list]=goldenSectionMin(f,a,b,iter\_max,tol);

disp("黄金分割法")

黄金分割法

fprintf('找到的近似零点: %.6f\n', x);

找到的近似零点: 2.000000

fprintf('函数极小值: %.6f\n', f\_val);

函数极小值: 0.000000

fprintf('迭代次数: %d\n', iter\_time);

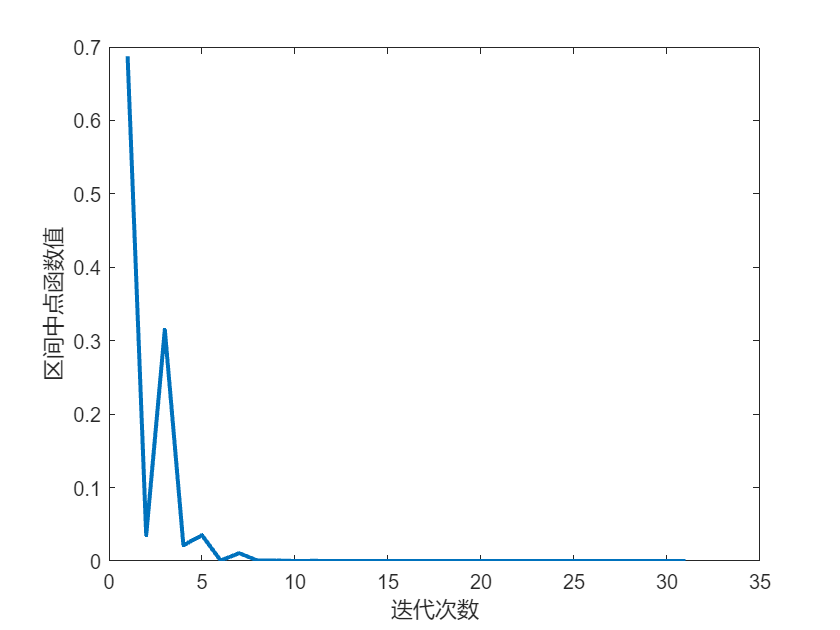
迭代次数: 31

%展示算法的极小化过程

plot([1:iter\_time],tol\_list,'LineWidth',2)

xlabel("迭代次数")

ylabel("区间中点函数值")



# 5. 探索 Matlab 内置命令求该函数极小值

%fminbnd

%查找单变量函数在定区间上的最小值

x1 = 0;

x2 = 3;

options = optimset('Display','iter');

x = fminbnd(f,x1,x2,options)

Func-count x f(x) Procedure

1 1.1459 0.988937 initial

2 1.8541 0.104232 golden

3 2.2918 0.731378 golden

4 1.76686 0.233608 parabolic

5 1.94397 0.0174569 parabolic

6 2.07683 0.0391491 golden

7 1.99429 0.000193919 parabolic

8 1.99709 5.04873e-05 parabolic

9 2.00035 7.55957e-07 parabolic

10 2.00001 4.95803e-10 parabolic

11 1.99998 3.53495e-09 parabolic

12 2.00004 1.08143e-08 parabolic

优化已终止:

当前的 x 满足使用 1.000000e-04 的 OPTIONS.TolX 的终止条件

x = 2.0000

function [x\_min,f\_min,iter\_time,tol\_list]=binary\_search\_min(f,df,a,b,iter\_max,tol)

[x\_min,iter\_time,tol\_list]=binarySearchZero(df,a,b, iter\_max, tol);

f\_min=f(x\_min);

end

function [root, iterations,tol\_list] = binarySearchZero(func, a, b, iter\_max, tol)

if nargin < 5

tol = 1e-6; % 默认容差

end

if nargin < 4

iter\_max = 1000; % 默认迭代次数

end

fa = func(a);

fb = func(b);

tol\_list=[];

if fa \* fb > 0

error('函数在给定区间端点处没有根');

end

iterations = 0;

while iterations < iter\_max

iterations = iterations + 1;

c = (a + b) / 2;

fc = func(c);

tol\_list=[tol\_list,abs(fc)];

if abs(fc) < tol

root = c;

return;

end

if fa \* fc < 0

b = c;

fb = fc;

else

a = c;

fa = fc;

end

end

root = (a + b) / 2;

end

function [xmin,fmin,iterations,tol\_list] = ternarySearchMin(func, a, b, iter\_max, tol)

if nargin < 5

iter\_max = 1000; % 默认迭代次数

end

if nargin < 4

tol = 1e-6; % 默认容差

end

tol\_list=[];

iterations = 0;

while (b - a) > tol && iterations < iter\_max

iterations = iterations + 1;

% 计算两个切分点

m1 = a + (b - a) / 3;

m2 = b - (b - a) / 3;

% 计算函数在切分点处的值

fm1 = func(m1);

fm2 = func(m2);

tol\_list=[tol\_list,abs(func((m1+m2)/2))];

if fm1 < fm2

b = m2; % 更新搜索区间

else

a = m1; % 更新搜索区间

end

end

xmin = (a + b) / 2;

fmin = func(xmin);

end

function [xmin, fmin, iterations,tol\_list] = goldenSectionMin(func, a, b, iter\_max, tol)

goldenRatio = (sqrt(5) - 1) / 2; % 黄金比例

if nargin < 5

iter\_max = 1000; % 默认迭代次数

end

if nargin < 4

tol = 1e-6; % 默认容差

end

tol\_list=[];

iterations = 0;

while (b - a) > tol && iterations < iter\_max

iterations = iterations + 1;

m1 = b - goldenRatio \* (b - a);

m2 = a + goldenRatio \* (b - a);

fm1 = func(m1);

fm2 = func(m2);

tol\_list=[tol\_list,abs(func((m1+m2)/2))];

if fm1 < fm2

b = m2;

else

a = m1;

end

end

xmin = (a + b) / 2;

fmin = func(xmin);

end

# 6. 比较各算法的精度，效率

显然，由上面结果可以看到在设置容忍误差为1e-6时，各个算法均能在规定最大迭代次数能达到容忍误差，其中算法效率：二分法＞黄金分割法＞三分法

改变参数对结果的影响：不难估计，当最大容忍误差越小时，迭代次数越多。改变最大容忍误差，将tol设置为1e-9，则二分法需要34次，三分法需要54次迭代，黄金分割法需要46次迭代才能达到容忍误差。

改变容忍误差为1e-16 ，设置最大迭代次数为10，结果为：

二分法：

找到的近似零点: 1.999512

函数极小值: 0.000001

迭代次数: 10

三分法

找到的近似零点: 1.995986

函数极小值: 0.000096

迭代次数: 10

黄金分割法

找到的近似零点: 1.994015

函数极小值: 0.000213

迭代次数: 10

不难分析，精度：二分法＞三分法＞黄金分割法