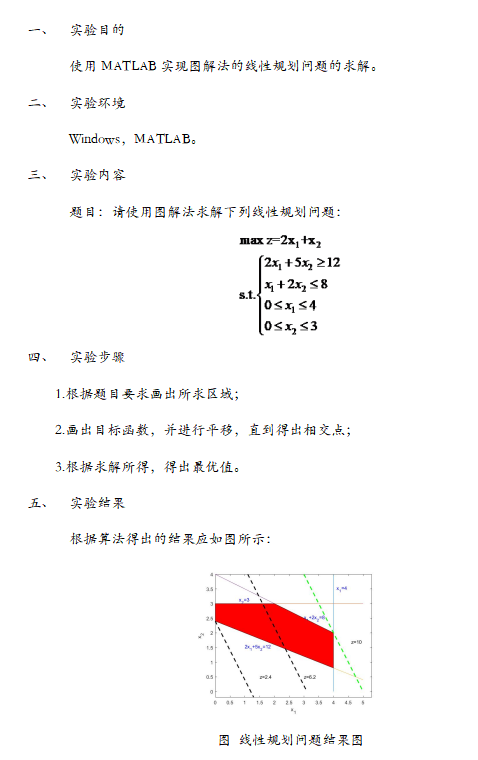
**最优化实验报告**

**第一次实验**

1. 问题概述

****

1. 方法设计

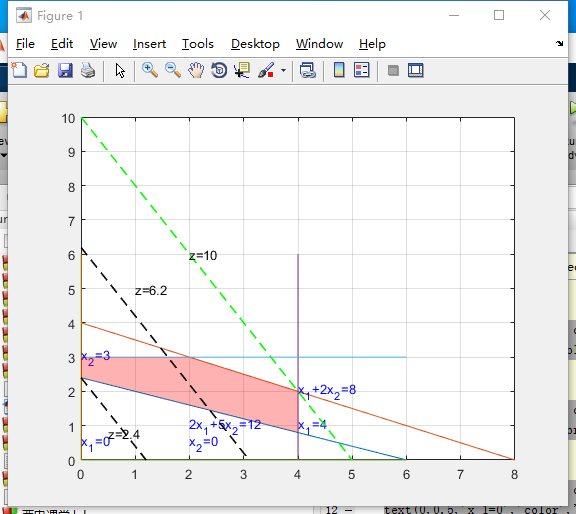
%% 直线  
L1=[0,12/5;6,0];  
plot(L1(:,1),L1(:,2));hold on  
text(2,1,'2x\_1+5x\_2=12','color','b');  
  
L2=[0,4;8,0];  
plot(L2(:,1),L2(:,2));hold on  
text(4,2,'x\_1+2x\_2=8','color','b');  
  
L3=[0,0;0,6];  
plot(L3(:,1),L3(:,2));hold on  
text(0,0.5,'x\_1=0','color','b');  
  
L4=[4,0;4,6];  
plot(L4(:,1),L4(:,2));hold on  
text(4,1,'x\_1=4','color','b');  
  
L5=[0,0;6,0];  
plot(L5(:,1),L5(:,2));hold on  
text(2.0,0.5,'x\_2=0','color','b');  
  
L6=[0,3;6,3];  
plot(L6(:,1),L6(:,2));hold on  
text(0,3,'x\_2=3','color','b');  
grid on  
  
%% 填充  
[X1,X2]=meshgrid(0:0.01:7,0:0.01:3);  
idX1=(2\*X1+5\*X2>=12)&(X1+2\*X2<=8)&(X1>=0)&(X1<=4)&(X2>=0)&(X2<=3);  
X1=X1(idX1);  
X2=X2(idX1);  
k=convhull(X1,X2);  
h=fill(X1(k),X2(k),'r');  
set(h,'edgealpha',0,'facealpha',0.3)  
  
%% 目标函数  
z0=[0,2.4;1.2,0];  
plot(z0(:,1),z0(:,2),'k--','LineWidth',1.2);  
text(0.5,0.8,'z=2.4');  
z1=[0,6.2;3.1,0];  
plot(z1(:,1),z1(:,2),'k--','LineWidth',1.2);  
text(1,5,'z=6.2');  
z2=[0,10;5,0];  
plot(z2(:,1),z2(:,2),'g--','LineWidth',1.2);  
text(2,6,'z=10');

首先用L1,L2,L3,L4,L5,L6分别定义六个直线，他们分别是2x1+5x2=12;x1+x2=8;x2=0;x2=4;x1=0;x1=3;然后根据计算及平移，求得x1=2,x2=6时，2x1+x2有最大值。确定六条直线L1、L2、L3、L4、L5和L6的坐标点，这些直线由它们在x轴和y轴上的截距定义。随后，使用plot函数在图形上绘制这些直线，并通过text函数在每个直线旁添加相应的线性约束方程。

接下来，为了绘制由这些线性约束条件确定的填充区域，创建一个网格。这个网格的x值范围从0到7，y值范围从0到3，步长为0.01。通过评估每个网格点是否满足所有约束条件，筛选出满足条件的点，并将这些点的坐标存储在变量idX1中。然后，使用convhull函数来计算这些点构成的凸包的边界。最后，使用fill函数将凸包内部区域填充为红色，以表示满足所有约束条件的可行解区域。

之后，定义目标函数的等高线。选取三个不同的等高线高度，分别对应目标函数z的三个不同取值。使用plot函数，将这些等高线以虚线的形式绘制在图形上。同时，使用text函数在每个等高线旁添加对应的目标函数表达式。

最终，得到一个二维图形，其中包含了表示线性约束条件的直线和表示目标函数不同取值水平的等高线。填充的红色区域展示了满足所有约束条件的可行解空间，而等高线则展示了目标函数在解空间中的变化。通过观察这个图形，可以大致确定目标函数最大值点可能所在的位置。结果



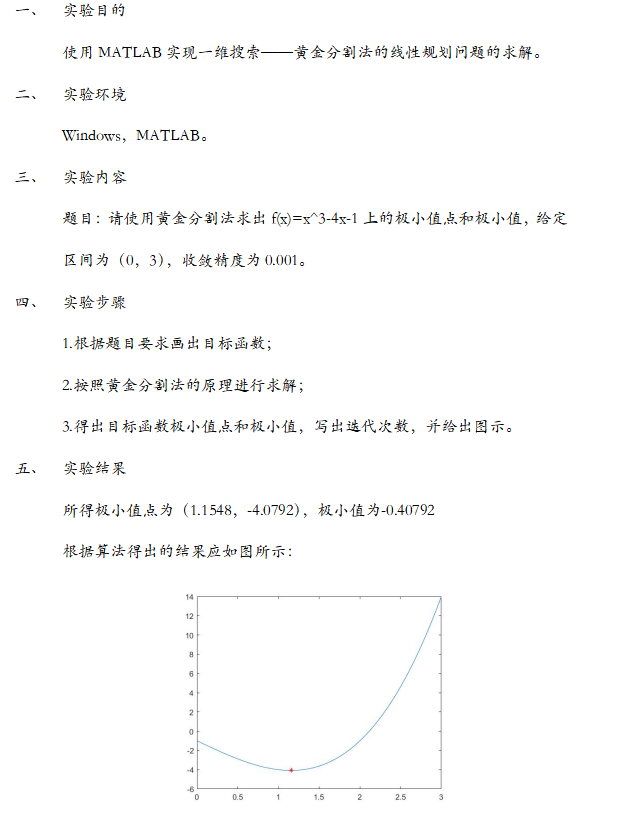
结果如图所示，红色区域即为可行解，绿色直线与红色区域交点即为最优解

1. 讨论

求解线性规划问题最重要的是需要把图画出来，当可行解的区域画出来，根据直线的平移，求出两端的临界直线，可求得最优解，即求出最大值

**第二次实验**

1. 问题概述

****

1. 方法设计

function goldensection

g=@(x)x^3-x\*4-1;% 定义要求的g（x）函数

i=0;% 定义初始次数，用来表示迭代次数

a=0;

b=3;

k=b-a;

x1=a+0.382\*(b-a);

x2=a+0.618\*(b-a);% 第一次区间缩短计算

while abs(k)>0.001 % 开始进行迭代计算

i=i+1;%迭代次数加一

y1=g(x1);

y2=g(x2);

if y1<y2 % 对比函数值

b=x2;

x2=x1;

y2=y1;

x1=a+0.382\*(b-a);

y1=g(x1);

else

a=x1;

x1=x2;

y1=y2;

x2=a+0.618\*(b-a);

y2=g(x2);

end

k=b-a;

end

x0=(a+b)/2; % 求出最后的x

y0=g(x0); % 计算最后的y值

%y=g(x);

fplot(g,[0.0,3.0])

hold on;

plot(x0,y0,'r\*');

fprintf('%f %f\n',x0,y0);

fprintf('%d',i);

定义了目标函数g(x)，该函数描述了关于x的函数关系。

设置迭代计数器i为零，用于记录迭代次数。

初始化搜索区间[a, b]，并计算初始区间长度k为b-a。

基于黄金分割比例（0.382和0.618），计算首次迭代中的两个试探点x1和x2。

开始迭代循环，检查区间长度k是否小于预设的收敛阈值（如0.001）。若满足，则结束循环；否则，继续迭代。

在每次迭代中，计算x1和x2对应的函数值y1和y2。

根据函数值y1和y2的比较结果，更新搜索区间[a, b]。若y1较小，则更新b为x2，x2为新计算的x1，并重新计算x1；否则，更新a为x1，x1为新计算的x2，并重新计算x2。

更新区间长度k为b-a。

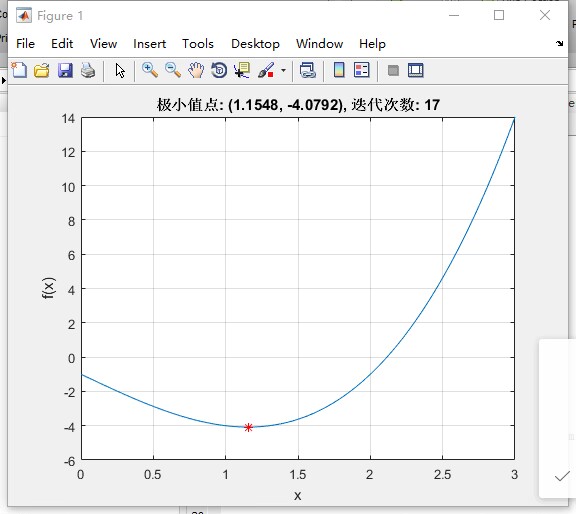
重复迭代步骤，直至满足收敛条件。

迭代结束后，得到近似最小值点x0，并计算其对应的函数值y0。

使用fplot函数绘制函数g(x)的图像，覆盖[0.0, 3.0]区间。

在图像上，使用plot函数标记出最小值点(x0, y0)。

通过fprintf函数，打印出最小值点的坐标(x0, y0)和迭代次数i。



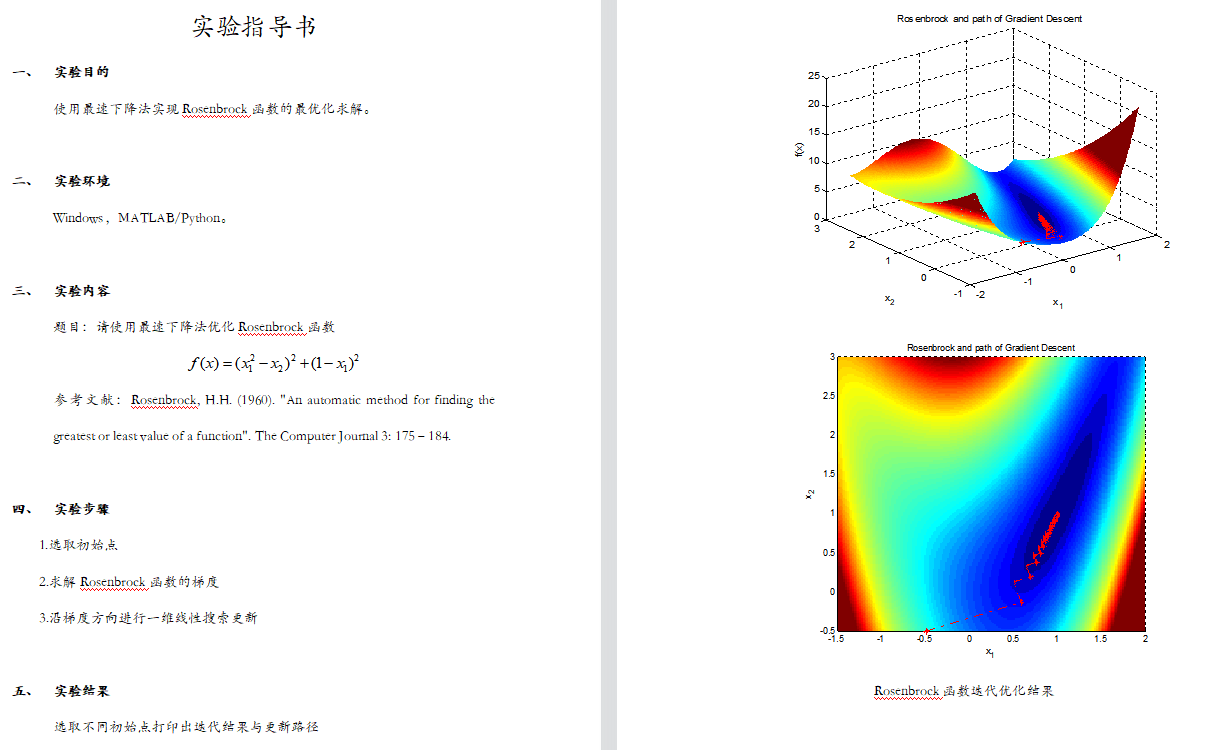
结果如图所示，红色对应的点极为最小值点

1. 讨论

利用黄金分割法，进行多次迭代运算，对计算出来的y（x1）和y（x2）进行比较，选取不同的区间进行迭代

**第三次实验**

1. 问题概述

****

1. 方法设计

% 定义梯度函数，@(x) 表示函数句柄，x 是输入参数

f = @(x) (x(1)^2 - x(2))^2 + (1 - x(1))^2;

% 初始化参数

x\_initial = [-0.5; -0.5]; % 初始点（可以更改为不同的初始点）

learning\_rate = 0.1; % 学习率，控制参数更新的步长

max\_iterations = 1000; % 最大迭代次数，避免无限循环

tolerance = 1e-8; % 收敛容差，判断算法是否达到收敛

% 迭代过程记录

x\_history = zeros(2, max\_iterations); % 用于记录每次迭代的参数值

x\_history(:, 1) = x\_initial; % 将初始点存入记录中

% 最速下降法优化

x = x\_initial; % 初始点作为起点

for iteration = 2:max\_iterations

% 计算梯度

grad = gradient(x);

% 梯度表示函数在当前点的变化率，用于指导参数的更新方向

% 更新参数

x\_new = x - learning\_rate \* grad;

% 根据梯度和学习率更新参数，控制参数更新的步长

% 判断是否达到收敛

if norm(x\_new - x) < tolerance

x\_history(:, iteration:end) = repmat(x\_new, 1, max\_iterations - iteration + 1);

% 若参数变化较小，判断算法已达到收敛，将剩余迭代次数内的参数设为最终收敛点

break;

end

% 更新参数并记录历史值

x = x\_new;

x\_history(:, iteration) = x;

% 更新参数为新的值，并将新的参数存入记录中

end

% 输出结果

fprintf('最优点：[%f, %f]\n', x(1), x(2));

fprintf('最优值：%f\n', f(x));

% 输出最终收敛点的坐标和最小值

% 生成网格点坐标

[X1, X2] = meshgrid(-1.5:0.01:2, -0.5:0.01:3);

% 计算函数值

Z = (X1.^2 - X2).^2 + (1 - X1).^2;

% 绘制三维视图

figure;

mesh(X1, X2, Z);

xlabel('x1');

ylabel('x2');

zlabel('f');

title('函数 f 的三维视图');

% 绘制更新路径和等高线图

figure;

hold on;

[X1, X2] = meshgrid(-2:0.01:2, -1:0.01:3);

Z = (X1.^2 - X2).^2 + (1 - X1).^2;

contourf(X1, X2, Z, 100, 'LineColor', 'none');

plot(x\_history(1, :), x\_history(2, :), 'ro-');

quiver(x\_history(1, 1:end-1), x\_history(2, 1:end-1), x\_history(1, 2:end) - x\_history(1, 1:end-1), x\_history(2, 2:end) - x\_history(2, 1:end-1), 0.5, 'k');

hold off;

xlabel('x1');

ylabel('x2');

title('更新路径和等高线图');

colorbar;

首先，定义了计算梯度的函数句柄，该句柄接收输入参数x并返回函数在该点的梯度值。

接着，设定了初始条件，包括起始点x\_initial、学习率learning\_rate、最大迭代次数max\_iterations和收敛阈值tolerance。

为了追踪迭代过程，初始化了一个数组x\_history来存储每次迭代中的参数值。

然后，应用最速下降法进行优化。从初始点开始，通过循环迭代进行参数更新，直到达到最大迭代次数或满足收敛条件。

在每次迭代中，计算当前参数点处的梯度grad，并根据梯度和学习率计算新的参数值x\_new。

检查新参数与当前参数的差异是否小于收敛阈值。如果满足条件，则提前结束迭代，将后续迭代中的参数值设为最终收敛点。

更新当前参数为新的值，并将新参数添加到x\_history中。

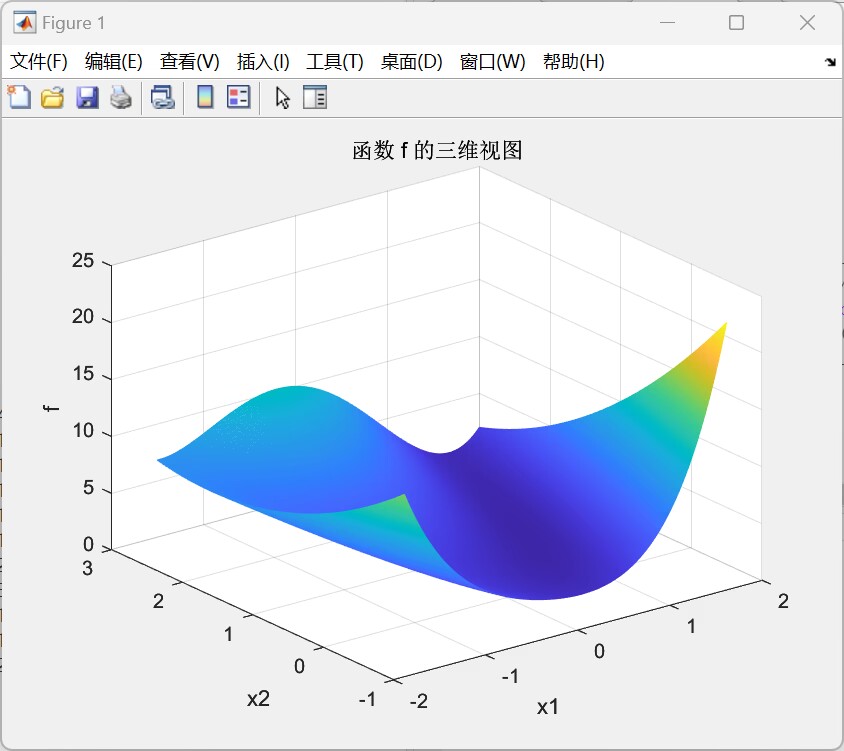
完成迭代后，输出最终收敛点的坐标和对应的函数值。

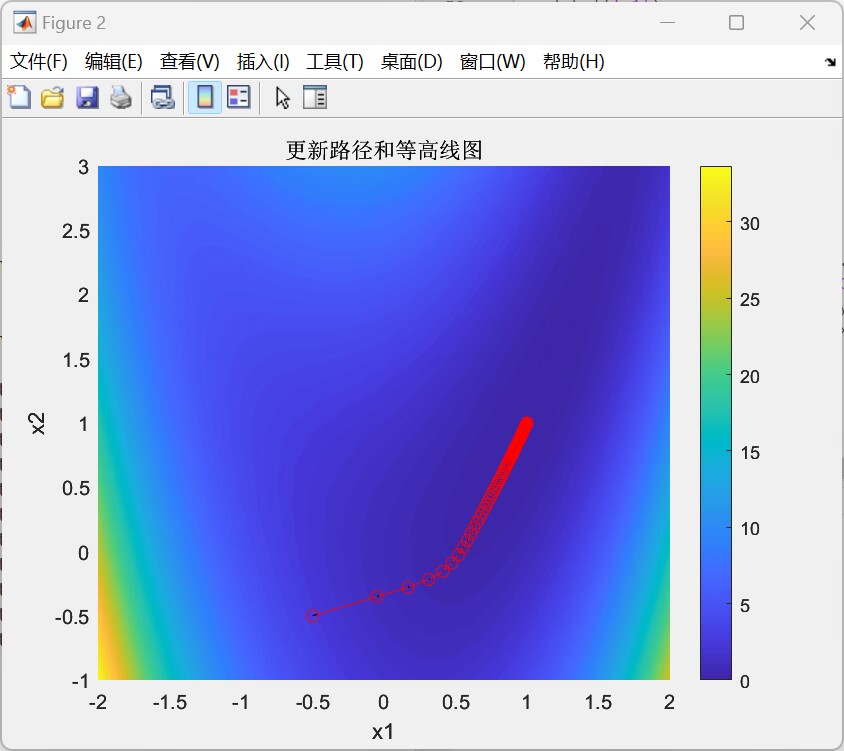
为了可视化结果，生成了网格点坐标并计算了相应的函数值，用于绘制函数的三维视图。

使用绘图函数创建了函数的三维视图，并添加了参数更新路径和等高线图。等高线图通过contourf函数绘制，而参数更新路径使用plot函数绘制，并使用quiver函数（或类似的箭头表示）展示了参数更新的方向。

通过这段代码，可以使用最速下降法来求解二维函数的最小值点，并绘制函数的三维视图、更新路径和等高线图，帮助理解算法的优化过程。

1. 结果





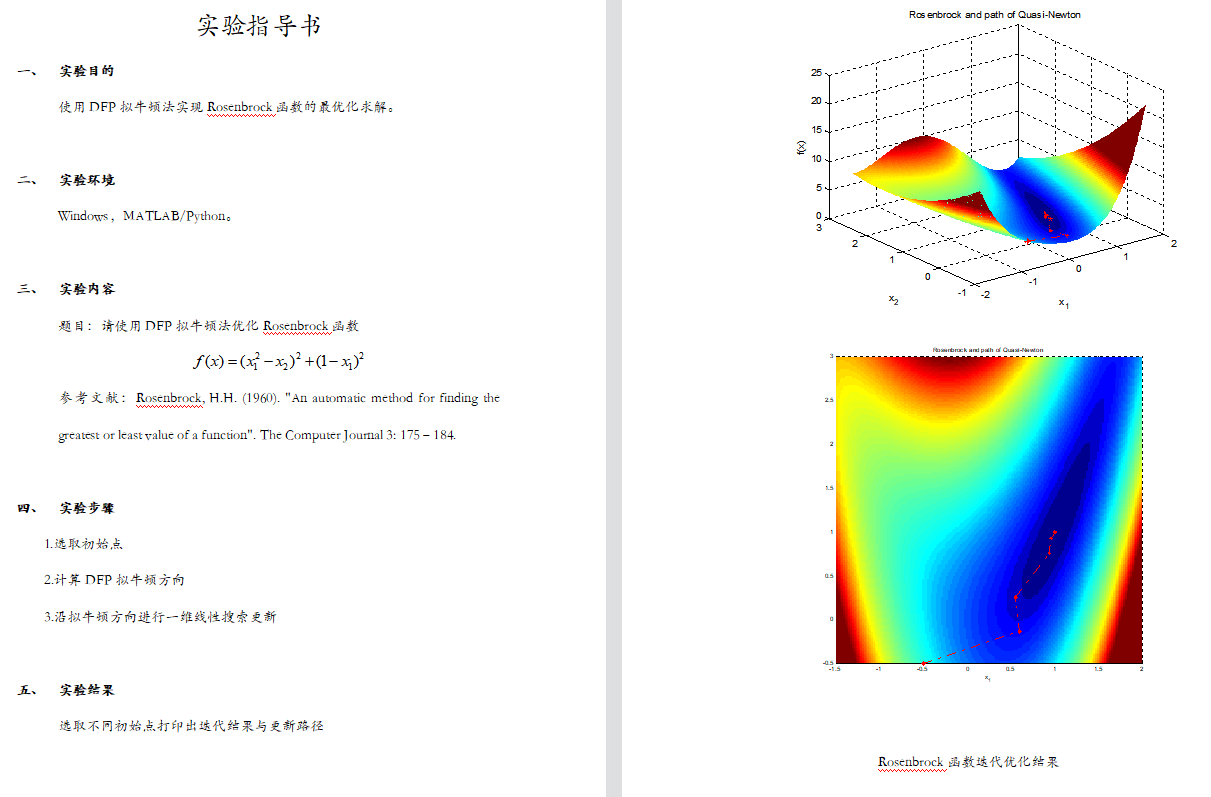
1. 讨论

按最优化算法的 基本思想是从一个给定的初始点出发，通过基本迭代格式，按照特定的算法产生一串点列如果点列收敛，则该点列的极限点为问题的最优解。

它不仅可以直接用来求解无约束优化问题，而且很多约束优化问题也常将其转化为无约束优化问题。

**第四次实验**

1. 问题概述

****

1. 方法设计

f = @(x) (x(1)^2 - x(2))^2 + (1 - x(1))^2;% 定义目标函数，@(x) 表示函数句柄，x 是输入参数

gradient = @(x) [4 \* x(1) \* (x(1)^2 - x(2)) + 2 \* (x(1) - 1); -2 \* (x(1)^2 - x(2))];% 定义梯度函数，@(x) 表示函数句柄，x 是输入参数

x\_initial = [-0.5; -0.5];% 初始点（可以更改为不同的初始点）

max\_iterations = 1000;% 最大迭代次数，避免无限循环

tolerance = 1e-8;% 收敛容差，判断算法是否达到收敛

H = eye(2);

x\_history = zeros(2, max\_iterations);% 用于记录每次迭代的参数值

x\_history(:, 1) = x\_initial;% 将初始点存入记录中

x = x\_initial;

grad = gradient(x);

for iteration = 2:max\_iterations

p = -H \* grad;

step\_size = 1;

while f(x + step\_size \* p) > f(x) + 0.5 \* step\_size \* (grad' \* p)

step\_size = step\_size / 2;

end

x\_new = x + step\_size \* p;

grad\_new = gradient(x\_new);

% 梯度表示函数在当前点的变化率，用于指导参数的更新方向

s = x\_new - x;

y = grad\_new - grad;

rho = 1 / (y' \* s);

H = (eye(2) - rho \* s \* y') \* H \* (eye(2) - rho \* y \* s') + rho \* s \* s';

% 判断是否达到收敛

if norm(x\_new - x) < tolerance

x\_history(:, iteration:end) = repmat(x\_new, 1, max\_iterations - iteration + 1);

% 若参数变化较小，判断算法已达到收敛，将剩余迭代次数内的参数设为最终收敛点

break;

end

x = x\_new;

grad = grad\_new;

x\_history(:, iteration) = x;

% 更新参数为新的值，并将新的参数存入记录中

end

fprintf('Optimal point: [%f, %f]\n', x(1), x(2));

fprintf('Optimal value: %f\n', f(x));

% 输出最终收敛点的坐标和最小值

% 生成网格点坐标

[X1, X2] = meshgrid(-1.5:0.01:2, -0.5:0.01:3);

% 计算函数值

Z = (X1.^2 - X2).^2 + (1 - X1).^2;

% 绘制三维视图

figure;

mesh(X1, X2, Z);

xlabel('x1');

ylabel('x2');

zlabel('f');

title('3D view of f');

% 绘制更新路径和等高线图

figure;

hold on;

colormap jet; % Set the colormap to jet (from red to purple)

contourf(X1, X2, Z, 80, 'LineColor', 'none');

plot(x\_history(1, :), x\_history(2, :), 'r.-');

quiver(x\_history(1, 1:end-1), x\_history(2, 1:end-1), x\_history(1, 2:end) - x\_history(1, 1:end-1), x\_history(2, 2:end) - x\_history(2, 1:end-1), 0.0001, 'k');

hold off;

xlabel('x1');

ylabel('x2');

title('Convergence path and contour plot');

colorbar;

定义梯度函数：首先，定义了一个梯度函数gradient，它接受输入参数x并返回函数在该点的梯度值。

初始化参数：设定了初始点x\_initial、最大迭代次数max\_iterations和收敛容差tolerance。

初始化Hessian矩阵：将Hessian矩阵H初始化为单位矩阵，作为拟牛顿法迭代的起点。

拟牛顿法迭代：从初始点x开始，进行迭代直至达到最大迭代次数或满足收敛条件。

计算当前点的梯度grad。

使用Hessian矩阵H计算搜索方向p。

通过线搜索确定合适的步长step\_size，确保满足Wolfe条件。更新参数值x，得到新的参数点x\_new。

计算新的梯度grad\_new和参数变化量s。

根据拟牛顿法的更新公式更新Hessian矩阵H。

检查收敛性，若满足条件则停止迭代。

记录迭代历史：在迭代过程中，记录每次更新的参数值到x\_history中。

结果输出：迭代完成后，输出最终收敛点的坐标和对应的函数值。

三维视图绘制：生成网格点坐标并计算函数值，用于绘制函数的三维视图。

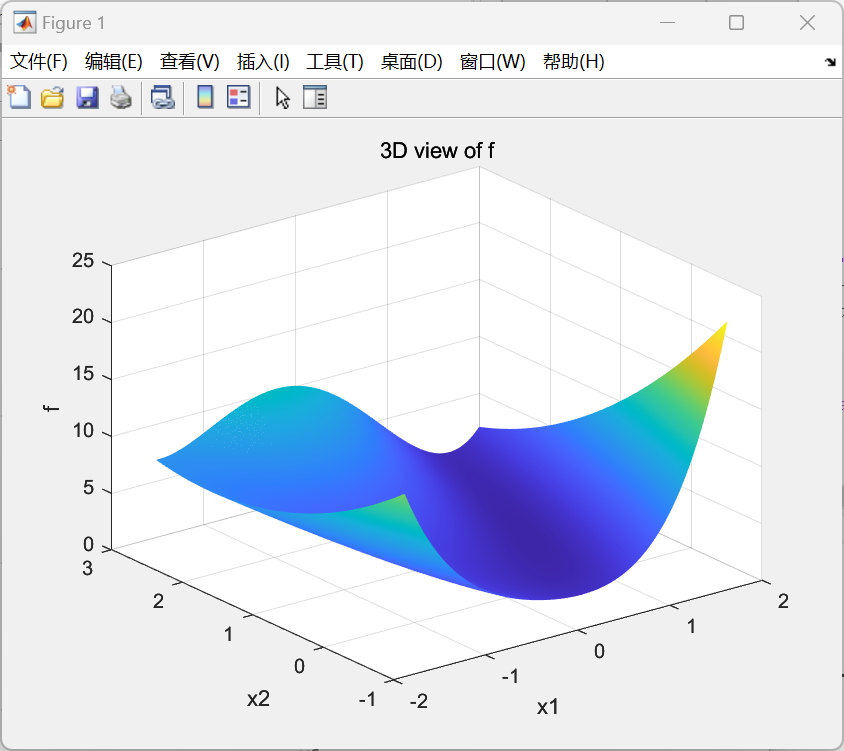
可视化优化过程：

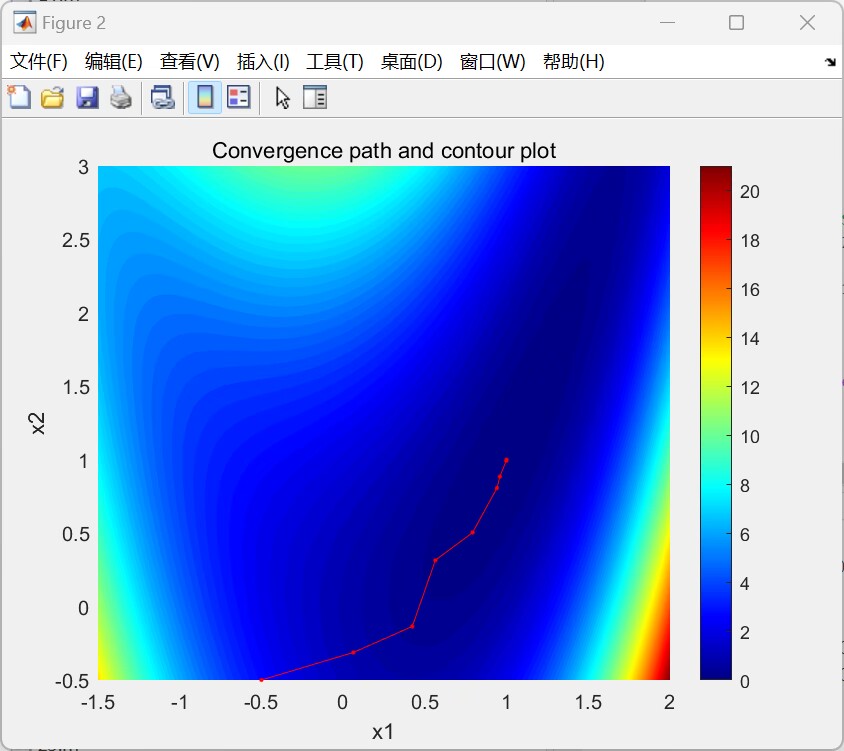
使用contourf函数绘制等高线图。

使用plot函数绘制参数更新路径。

使用quiver函数（或类似函数）绘制表示更新方向的箭头。

图像标签与标题：为生成的图像设置合适的标签和标题。





1. 讨论

已知目标函数，根据迭代方向进行求最优解运算首先构造目标函数在当前迭代$x\_k$的二次模型：m\_k(p)=f\_k+g\_k^T p+p^T B\_k p/2，这里f\_k=f(x\_k)，g\_k=▽f(x\_k)，B\_k是一个对称[正定矩阵](https://baike.so.com/doc/5437178-5675487.html)。于是我们取这个二次模型的最优解p\_k=-B\_k^{-1} g\_k作为搜索方向，并且得到新的迭代点x\_{k+1}=x\_k+a\_k p\_k，其中我们要求步长a\_k满足Wolfe条件。这样的迭代类似与[牛顿法](https://baike.so.com/doc/5960204-6173152.html)，区别就在于用近似的Hesse矩阵B\_k代替真实的Hesse矩阵。所以拟牛顿法最关键的地方就是每一步迭代中矩阵B\_k的更新。现在假设得到一个新的迭代x\_{k+1}，并得到一个新的二次模型：m\_{k+1}(p)=f\_{k+1}+g\_{k+1}^T p + p^T B\_{k+1} p/2。我们尽可能地利用上一步的信息来选取B\_{k+1}。具体地，我们要求g\_{k+1}-g\_k=a\_k B\_{k+1} p\_k，从而得到B\_{k+1}s\_k=y\_k，其中s\_k=x\_{k+1}-x\_k，y\_k=g\_{k+1}-g\_k。