重庆大学
CHONGQINGDAXUE

先进制造技术课程报告

基于数字孪生的先进制造技术

姓名 闵进伍

学号 202407021385T

学院 机械与运载工程学院

任课教师 唐先智

2024 年 12 月 22 日

从一维搜索到无约束优化方法

摘 要

作为一种系统的、基于科学和数学的设计方法，优化设计是现代设计理论与方法的一个重要领域，广泛应用于各工业部门。通过合理的优化方法，可以在满足各种约束条件的前提下，找到最优的设计方案，从而提高产品或系统的性能、降低成本、节省资源并提升整体效率。本报告介绍了常见的一维搜索与无约束优化方法并编写了对应的 matlab 程序。最后，结合实践过程简要说明了自身的感想与体会。

关键词：优化设计；优化方法；一维搜索；无约束优化方法；

目录

1 概述

1.1 一维搜索与无约束优化方法概述

一维搜索方法是一类在一个固定方向或区间内寻找目标函数最优解的优化方法。其基本思想是在目标函数的一个维度上进行搜索，逐步逼近最优解。通常，目标函数是多维的，但在一维搜索中，我们只关心其中的一个维度或方向。这个方法的核心任务是通过计算和评估目标函数值来寻找该方向上的最优点。目前常见的精确一维搜索方法包括对分搜索法、等间隔搜索法、斐波那契法与黄金分割法，常见的非精确一维搜索方法包括 Armijo 搜索方法，wolfe 搜索方法以及强 wolfe 搜索方法。

无约束优化方法是指在没有任何约束条件下，寻找一个目标函数最小值或最大值的方法。在无约束优化问题中，优化的目标是找到函数的最优解，而不需要考虑任何限制或约束条件。通常，这类优化问题的目标函数是一个实值函数，定义在一个连续的变量空间内。无约束优化问题常见于机器学习、数据拟合、图像处理等领域。目前常见的无约束优化方法包括梯度下降法、powell 法、共轭梯度法与拟牛顿法。

一维搜索方法是无约束优化算法中的重要工具，尤其在确定合适步长时发挥关键作用。无约束优化方法通过计算梯度、海塞矩阵等信息来确定搜索方向，而一维搜索可以确定在某一方向上的精确步长或可接受步长，从而确保每步都向最优解靠近。因此，一维搜索在无约束优化中通常是不可或缺的组成部分，尤其在处理大规模、高维度的优化问题时，能够有效提高求解效率和精度。

常见一维搜索方法	
精确一维搜索方法	对分搜索法、等间隔搜索法、黄金分割法、斐波那契法、牛顿法
非精确一维搜索方法	Armijo 法、goldenstein 法、wolfe 法、强 wolfe 法
常见无约束优化方法	
坐标轮换法、powell 法、梯度下降法、共轭梯度法、拟牛顿法	

表 1.1 常见一维搜索与无约束优化方法

2 一维搜索方法

2.1 对分搜索法

对分搜索法是一维搜索法中较为简单的方法，它的实现原理主要利用了给定区间的中点，有点类似于二分法。对分搜索法实现的基本原理如下：

1、 给定目标函数、初始上下界、终止误差、当前的 x 值与当前方向

2、 计算中点结合终止误差得到新的上下界，得到上下界对应的函数的值

3、 进行上下界对应函数值判断来更新区间，最后通过终止误差来结束循环

4、 迭代次数加一

值得注意的是，对分搜索法每次区间的缩减率在 0.5 左右。

2.2 等间隔法

等间隔搜索方法包括多个等间隔搜索，本报告将介绍三点四区间等分搜索方法，三点等分搜索方法缩减率相比二点等分缩减率更低相比于四等分搜索方法地迭代次数更少，其中缩减率为上一区间与更新后区间比值，三等分搜索方法的缩减率为 0.5。三等分搜索方法原理如下：

1、 给定目标函数、初始上下界、终止误差、当前的 x 值与当前方向

2、 将函数区间等分为四份，计算三个等分点

3、 通过上下界与等分点函数值来更新区间，继续迭代直至小于等于终止误差

4、 迭代次数加一

2.3 斐波那契法

斐波那契方法是精确一维搜索方法中收敛速率相对较快的方法，基于斐波那契数列的方法，可以在初始时确定迭代次数与每一次迭代的缩减率，斐波那契方法的原理如下：

1、 给定目标函数、初始上下界、终止误差、当前的 x 值与当前方向

2、 通过终止误差与初始区间确定迭代次数与每次迭代缩减率

3、 通过区间与缩减率可以得到新的点，通过区间与点对应函数值更新区间直至倒数第二次迭代

4、 倒数第二次两个迭代点重合，此时采用一次对分搜索方法

5、 迭代次数加一

斐波那契方法的缩减率每次迭代是不相同的，并且区间更新是基于对称区间的思想，在第一次计算时需要计算两个迭代点而后续计算只需要计算一个新的迭代点，另一个迭代点只需要利用上一个迭代点的信息即可，在迭代到倒数第二个点时，此时两个迭代点重合，需要采用一次对分搜索法后得到精确步长。

2.4 黄金分割法

黄金分割法的思想是确保每次迭代的缩减率为黄金比例，取缩减率约等于 0.618，黄金分割法的原理如下：

1、 给定目标函数、初始上下界、终止误差、当前的 x 值与当前方向

2、 通过缩减率得到迭代点，通过迭代点的函数值更新区间直至终止法则

3、 迭代次数加一

2.5 Armijo 方法

Armijo 方法是获取非精确步长的一种方法，Armijo 方法的基本思想是利用了初始点的函数值与梯度信息来确定可接受步长范围，通过可接受步长范围结合函数插值的方法可以获取可接受步长，其具体实现原理如下：

1、 给定目标函数、目标函数梯度、当前的 x 值与当前方向，Armijo 条件参数 rho

2、 通过初始的区间，确定 Armijo 条件并进行判断是否达到条件直至满足

3、 迭代次数加一

2.6 wolfe 与强 wolfe 法

wolfe 是进一步利用了得到的新的迭代点的梯度信息并于初始点的梯度信息进行比较，当满足条件时即可确定可接受步长。强 wolfe 条件是在 wolfe 条件的基础上进一步缩减了区间范围，强 wolfe 条件确定的可接受步长与精确步长较为接近，强 wolfe 条件实现基本原理如下：

1、 给定目标函数、目标函数梯度、当前的 x 值与当前方向，Armijo 条件参数 rho 与 wolfe 条件参数 sigma

2、 通过初始的区间，确定 Armijo 条件与 wolfe 条件并进行判断是否达到条件直至满足

3、 迭代次数加一

”对分搜索方法代码”

```
function [alpha_star]=Dichotomous_search(f_test,alpha_lower,alpha_upper,tolerance,
    x_current,d_current)
    if(tolerance>=10^-8)
        disturbance_quantity = 10^-9;
    else
        disturbance_quantity = tolerance*0.1;
    end
    %   k = 0 时的情况
    k = 0;
    alpha_lower_k = alpha_lower;
    alpha_upper_k = alpha_upper;
    alpha_left_k = (alpha_lower_k+alpha_upper_k)/2 - disturbance_quantity;
    alpha_right_k = (alpha_lower_k+alpha_upper_k)/2 + disturbance_quantity;
    x_current_left_k = x_current+alpha_left_k*d_current;
    x_current_right_k = x_current+alpha_right_k*d_current;
    f_alpha_left_k = f_test(x_current_left_k);
    f_alpha_right_k = f_test(x_current_right_k);
    %   k >= 1 时的情况
    while(abs(alpha_upper_k - alpha_lower_k)>tolerance)
        if(f_alpha_left_k<f_alpha_right_k)
            alpha_upper_k = alpha_right_k;
        elseif(f_alpha_left_k>f_alpha_right_k)
            alpha_lower_k = alpha_left_k;
        else
            alpha_upper_k = alpha_right_k;
            alpha_lower_k = alpha_left_k;
        end
        alpha_left_k = (alpha_lower_k+alpha_upper_k)/2 - disturbance_quantity;
        alpha_right_k = (alpha_lower_k+alpha_upper_k)/2 + disturbance_quantity;
        x_current_left_k = x_current+alpha_left_k*d_current;
        x_current_right_k = x_current+alpha_right_k*d_current;
        f_alpha_left_k = f_test(x_current_left_k);
        f_alpha_right_k = f_test(x_current_right_k);
        k = k+1;
    end
    alpha_star = (alpha_upper_k+alpha_lower_k)/2;
end
```

”斐波那契方法代码”

```
function [alpha_star] = Fibonacci_search(f_test,alpha_lower,alpha_upper,tolerance,
    x_current,d_current)
    Fabonacci_series_upper = (alpha_upper - alpha_lower)/tolerance;
    Fabonacci_series = [1,2];
    n = 2;
    while(Fabonacci_series(n)<=Fabonacci_series_upper)
        n = n+1;
        Fabonacci_series(n) = Fabonacci_series(n-1)+Fabonacci_series(n-2);
    end
    %   k = 0时的左右点以及
    k = 0;
    reduction_rate_k = Fabonacci_series(n-1)/Fabonacci_series(n);
    alpha_upper_k = alpha_upper;
    alpha_lower_k = alpha_lower;
    L_k = (alpha_upper_k - alpha_lower_k);
    alpha_left_k = alpha_upper_k - L_k*reduction_rate_k;
    alpha_right_k = alpha_lower_k + L_k*reduction_rate_k;
    x_current_left_k = x_current + alpha_left_k*d_current;
    x_current_right_k = x_current + alpha_right_k*d_current;
    f_x_left_k = f_test(x_current_left_k);
    f_x_right_k = f_test(x_current_right_k);
    %   k>=1时开始迭代
    while(abs(alpha_right_k - alpha_left_k)>tolerance&&n>=4)
        n = n - 1;
        reduction_rate_k = Fabonacci_series(n-1)/Fabonacci_series(n);
        if(f_x_left_k >f_x_right_k)
            alpha_lower_k = alpha_left_k;
            alpha_left_k = alpha_right_k;
            L_k = abs(alpha_upper_k - alpha_lower_k);
            alpha_right_k = alpha_lower_k + L_k*reduction_rate_k;
        elseif(f_x_left_k <f_x_right_k)
            alpha_upper_k = alpha_right_k;
            alpha_right_k = alpha_left_k;
            L_k = abs(alpha_upper_k - alpha_lower_k);
            alpha_left_k = alpha_upper_k - L_k*reduction_rate_k;
        else
            alpha_lower_k = alpha_left_k;
            alpha_upper_k = alpha_right_k;
            L_k = abs(alpha_upper_k - alpha_lower_k);
            alpha_right_k = alpha_lower_k + L_k*reduction_rate_k;
            alpha_left_k = alpha_upper_k - L_k*reduction_rate_k;
        end
        x_current_left_k = x_current + alpha_left_k*d_current;
        x_current_right_k = x_current + alpha_right_k*d_current;
        f_x_left_k = f_test(x_current_left_k);
        f_x_right_k = f_test(x_current_right_k);
        k = k+1;
    end
```

```
    %一次对分搜索法
    if(tolerance >10^8)
        disturbance_quantity = 10^-9;
    else
        disturbance_quantity = tolerance*0.1;
    end
    alpha_middle = (alpha_upper_k+alpha_lower_k)/2;
    alpha_right_k = alpha_middle + disturbance_quantity;
    alpha_left_k = alpha_middle - disturbance_quantity;
    x_current_left_k = x_current + alpha_left_k*d_current;
    x_current_right_k = x_current + alpha_right_k*d_current;
    f_x_right_k = f_test(x_current_right_k);
    f_x_left_k = f_test(x_current_left_k);
    if(f_x_left_k <f_x_right_k)
        alpha_upper_k = alpha_right_k;
    elseif(f_x_left_k >f_x_right_k)
        alpha_lower_k = alpha_left_k;
    else
        alpha_upper_k = alpha_rght_k;
        alpha_lower_k = alpha_left_k;
    end
    alpha_star = (alpha_upper_k + alpha_lower_k)/2;
end
```

”强 wolfe 方法代码”

```
function [alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current
    ,rho,sigma)
    k_max = 1000;
    k = 0;
    alpha_lower_k = 0;
    alpha_upper_k = 10^8;
    x_alpha_lower_k = x_current + alpha_lower_k*d_current;
    f_x_alpha_lower_k = f_test(x_alpha_lower_k);
    df_x_alpha_lower_k = d_current'*g_test(x_alpha_lower_k);
    f_x_alpha_lower_0 = f_x_alpha_lower_k;
    df_x_alpha_lower_0 = f_x_alpha_lower_k;
    tolerance = 10^-15;
%     if(df_x_alpha_lower_0> 0)
%         df_x_alpha_lower_0 = -df_x_alpha_lower_0;
%     end
    if(abs(df_x_alpha_lower_k) >tolerance)
      alpha_k = -2*f_x_alpha_lower_k/df_x_alpha_lower_k;
    else
        alpha_k = 1;
    end
    if(alpha_k - alpha_lower_k <tolerance)
        alpha_k = 1;
    end
    for k = 1:k_max
        x_alpha_k = x_current + alpha_k*d_current;
        f_x_alpha_k = f_test(x_alpha_k);
        df_x_alpha_k = d_current'*g_test(x_alpha_k);
        Armijo_condition = f_x_alpha_k - f_x_alpha_lower_0 - rho*df_x_alpha_lower_0
            *alpha_k;
        wolfe_condition =  abs(df_x_alpha_k) - sigma*abs(df_x_alpha_lower_0);
        if(Armijo_condition <=0)
            if(wolfe_condition <=0)
                alpha_acceptable = alpha_k;
                break;
            else
                if(df_x_alpha_k <0)
                    delta_alpha_k = (alpha_k - alpha_lower_k)*df_x_alpha_k/(
                        df_x_alpha_lower_k - df_x_alpha_k);
                    if(delta_alpha_k <=0)
                        alpha_k_temp = alpha_k - delta_alpha_k;
                    else
                        alpha_k_temp = alpha_k + delta_alpha_k;
                    end
                    alpha_lower_k = alpha_k;
                    f_x_alpha_lower_k = f_x_alpha_k;
                    df_x_alpha_lower_k = df_x_alpha_k;
                    alpha_k = alpha_k_temp;
                else
```

```
46                      if(alpha_k<alpha_upper_k)
47                          alpha_upper_k = alpha_k;
48                      end
49                      alpha_k_temp = alpha_lower_k - (1/2)*((alpha_k - alpha_lower_k)
                            ^2*df_x_alpha_lower_k)/(f_x_alpha_k - ...
50                      f_x_alpha_lower_k - df_x_alpha_lower_k*(alpha_k - alpha_lower_k
                            ));
51                      alpha_k = alpha_k_temp;
52                  end
53              end
54          else
55              if(alpha_k <alpha_upper_k)
56                  alpha_upper_k = alpha_k;
57              end
58              alpha_k_temp = alpha_lower_k - (1/2)*((alpha_k - alpha_lower_k)^2*
                    df_x_alpha_lower_k)/(f_x_alpha_k - ...
59                  f_x_alpha_lower_k - df_x_alpha_lower_k*(alpha_k - alpha_lower_k));
60              alpha_k = alpha_k_temp;
61          end
62          if(alpha_upper_k - alpha_lower_k <tolerance)
63              alpha_acceptable = alpha_k;
64              break;
65          end
66      end
67      if((Armijo_condition >0)||(wolfe_condition>0))
68          alpha_acceptable = NaN;
69      end
70  end
```

3 无约束优化方法

3.1 梯度下降法

梯度下降方法是无约束优化方法中一种较为简单的方法，其实现原理主要利用了目标函数的梯度信息，通过确定迭代方向为负梯度方向结合泰勒展开可以知道迭代方向一定是下降的，梯度下降法实现原理如下：

1、 给定目标函数、目标函数梯度、初始点 x 与终止误差

2、 通过初始点与目标函数梯度，结合一维搜索可以得到下一个迭代点直至满足终止误差

3、 迭代次数加一

3.2 共轭梯度法

共轭梯度法是基于共轭方向方法与梯度信息的方法，对于一个二次型函数，它的初始方向为初始点处的负梯度方向，而后续方向是与上一迭代点方向 Q 共轭的。利用下一个迭代方向确定下一个迭代点。对于一个非二次型函数，本报告采用的是 Dai-Yuan 法来进行方向的更新，梯度下降法实现基本原理如下：

1、 给定目标函数、目标函数梯度、初始点 x 与终止误差

2、 通过初始点与目标函数梯度，结合一维搜索并且利用 DY 法得到下一个迭代方向直至满足终止误差

3、 迭代次数加一

3.3 拟牛顿法

拟牛顿法是基于牛顿法的改进方法。拟牛顿方法的思想是由于牛顿方向需要计算海塞矩阵的逆的信息，这可能会有较大的计算量以及误差并且如果矩阵非正定修正难度较大，所以考虑一个新的矩阵 Q，该矩阵能够替代海塞矩阵的逆 H^{-1}, 这样就可以得到 Q 的跟新公式即 DFP 的方法并且保正矩阵是正定的，而 DFP 方法在使用的过程中对于大规模非二次型目标函数计算可能较慢，所以考虑其对偶问题即 BFGS 方法（变尺度法），他们的基本原理如下：

1、 给定目标函数、目标函数梯度、初始点 x 与终止误差

2、 设定初始 Q 为单位矩阵 I，利用 DFP 或 BFGS 公式跟新拟牛顿方向，得到新的迭代点直至满足终止条件

3、 迭代次数加一

”梯度下降方法代码”

```
function [x_optimal,f_optimal,k] = negative_gradient(f_test,g_test,x_initial,
    tolerance)
k = 1;
rho = 0.1;
sigma = 0.11;
x_current = x_initial;
g_current = g_test(x_current);
d_current = -g_current;
[alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,
    sigma);  %强wolfe条件
% [alpha_star] = Fabonacci(f_test,0,2,10^-9,x_current,d_current); %斐波那契
x_next = x_current + alpha_acceptable*d_current;
% x_next = x_current + alpha_star*d_current;
while(norm(x_next - x_current)> tolerance)
    k = k+1;
    x_current = x_next;
    g_current = g_test(x_current);
    d_current = -g_current;
    x_next = x_current + alpha_acceptable*d_current;
%     x_next = x_current + alpha_star*d_current;
end
x_optimal = x_next;
f_optimal = f_test(x_optimal);
end
```

”共轭梯度方法代码”

```
function  [x_optimal,f_optimal,k] = conjugate_gradient(f_test,g_test,x_initial,
    tolerance)
k  = 1;
rho = 0.1; sigma = 0.11;
x_current = x_initial;
n = length(x_current);
g_current = g_test(x_current);
d_current = -g_current;
[alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,
    sigma);
x_next = x_current + alpha_acceptable*d_current;
while(norm(x_next - x_current)> tolerance)
    k  = k+1;
    g_previous = g_current;
    d_previous = d_current;
    x_current = x_next;
    g_current = g_test(x_current);
    if(mod(k,n) == 0)
        d_current = -g_current;
    else
        beta_current = (g_current'*g_current)/(d_previous'*(g_current - g_previous)
            );
        d_current = -g_current + beta_current*d_previous;
    end
    [alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,
        sigma);
    x_next = x_current + alpha_acceptable*d_current;
end
x_optimal = x_next;
f_optimal = f_test(x_optimal);
end
```

”拟牛顿方法代码”

```
function [x_optimal,f_optimal,k] = DFP_METHOD(f_test,g_test,x_initial,tolerance)
%%  结合一下一维搜索方法
k = 1;
sigma = 0.11;
rho = 0.1;
x_current = x_initial;
n = length(x_current);
Q_current = eye(n);
g_current = g_test(x_current);
d_current = -Q_current*g_current;
[alpha_acceptable] = Fibonacci(f_test,0,2,10^-9,x_current,d_current);   %斐波那契额精确搜索
% [alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,sigma);% 强wolfe非精确搜索
x_next = x_current + alpha_acceptable*d_current;
f_next = f_test(x_next);
while(norm(x_next - x_current)> tolerance)
    k = k+1;
    x_previous = x_current;
    x_current = x_next;
    g_previous = g_test(x_previous);
    g_current = g_test(x_current);
    Q_previous = Q_current;
    s_current = x_current - x_previous;
    y_current = g_current - g_previous;
    if(s_current'*y_current<= 0)    %拟牛顿方向正定条件
        Q_current = eye(n);
    else
        Q_current = Q_previous + ((s_current)*(s_current)')/((s_current)'*(
            y_current)) - ((Q_current*y_current)*(Q_current*y_current)')...
        /(y_current'*Q_current*y_current);  %DFP更新公式
    end
    d_current = -Q_current*g_current;
%     [alpha_acceptable] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,sigma);
    [alpha_acceptable] = Fibonacci(f_test,0,2 ,10^-9,x_current,d_current);
    if(isnan(alpha_acceptable)) %放宽Armijo_wolfe条件
        rho = rho + 0.1;
        sigma = sigma + 0.1;
        continue;
    else
        x_next = x_current + alpha_acceptable*d_current;
        f_next = f_test(x_next);
    end
end
x_optimal = x_next;
f_optimal = f_next;
end
```

```
45
46
47 %% function BFGS algorithm
48 function [x_optimal,f_optimal,k] = BFGS_METHOD(f_test,g_test,x_initial,tolerance)
49 %% 将一维搜索方法添加
50 k = 1;
51 rho = 0.1;
52 sigma = 0.11;
53 x_current = x_initial;
54 n = length(x_current);
55 g_current = g_test(x_current);
56 Q_current = eye(n);
57 d_current =  -Q_current*g_current;
58 [alpha] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,sigma);
59 x_next = x_current + alpha*d_current;
60 while(norm(x_next - x_current)> tolerance)
61     k = k+1;
62     x_previous = x_current;
63     x_current = x_next;
64     Q_previous = Q_current;
65     g_previous = g_test(x_previous);
66     g_current = g_test(x_current);
67     s_current = x_current - x_previous;
68     y_current = g_current - g_previous;
69     if(Q_previous< 0)
70         Q_current = eye(n);
71     else
72         formula1 = 1 + (y_current'*Q_previous*y_current)/(y_current'*s_current);
73         formula2 = (s_current*s_current')/(s_current'*y_current);
74         formula3 = ((Q_previous*y_current*s_current')+(Q_previous*y_current*
               s_current')')/(y_current'*s_current);
75         Q_current = Q_previous + formula1*formula2 - formula3;
76     end
77     d_current = Q_current*g_current;
78     [alpha] = Armijo_wolfe_search(f_test,g_test,x_current,d_current,rho,sigma);
79     if(isnan(alpha))
80         rho = rho+1;
81         sigma = sigma+1;
82         continue;
83     else
84         x_next = x_current + alpha*d_current;
85     end
86 end
87 x_optimal = x_next;
88 f_optimal = f_test(x_optimal);
89 end
```

4 我的学习感悟与体会

从一维搜索到无约束约束优化方法，这部分是优化方法中较为基础的部分。本报告列举了常见的一维搜索与无约束优化方法。

其中，在编写无约束优化方法代码的过程中，我认为对于一维搜索方法确定其上下界是一个非常重要的过程，但是对于精确一维搜索方法目前感觉没有一个较好的方法来确定其上下界，对于无已知条件的函数来说，利用划线法或者尝试修改区间可以确定初始的上下区间，但是可能对于有多个极小点的函数来说，可能会陷入局部的最优步长，这个问题目前比较困难。

对于非精确步长，其本身就是确定一个可接受步长，所以其也可能在有多个极值的条件下可能确定的也为局部的最优步长，并且对于不同步长的选择可能影响其收敛速率，如果区间给的不合理会导致收敛困难。

对于无约束优化方法，方向的确定是十分重要的，而目前方向的确定大都需要利用到梯度信息，在没有梯度信息的情况下坐标轮换法、powell 法以及改进 powell 法对于初始点的坐标是严格的，这就会导致最终求解较为困难。

参考文献

[1] 刘兴高，胡云卿. 应用最优化方法及 *matlab* 实现. 科学出版社, 2014-1.

[2] Stanislaw H.Zak Edwin K.P.Chong. 最优化导论. 电子工业出版社, 2021-1.

[3] 王开荣. 最优化方法. 科学出版社, 2012-8.