NA_2

P 1

当 $p_0 = -1$

$\because$ 牛顿法下产生的序列 $\{p_n\}_{n=1}^{\infty}$ 的定义为

$$\begin{aligned} p_n &= p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \\ &= \frac{2p_{n-1}^3 - p_{n-1}\sin p_{n-1} - \cos p_{n-1}}{3p_{n-1}^2 - \sin p_{n-1}} \end{aligned}$$

设 $g(x) = \frac{2x^3 - x\sin x - \cos x}{3x^2 - \sin x}$

$\therefore p_2 = g(p_1) = g(g(p_0)) \approx -0.86568$

当 $p_0 = 0$

$$f'(x) = -3x^2 + \sin x \Rightarrow f'(p_0) = f'(0) = 0$$

$\because$ 牛顿法下产生的序列 $\{p_n\}_{n=1}^{\infty}$ 的定义为

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$$

$\therefore p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$

又 $\because$ $f'(p_0) = 0$

$\therefore$ 牛顿法在 $p_0 = 0$ 下不能用

P 2

(i)

由题知：

$$\begin{aligned} \epsilon_k &= \frac{\frac{1}{b} - x_k}{\frac{1}{b}} \\ &= 1 - bx_k \end{aligned}$$

且

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = 2x_k - bx_k^2$$

$\therefore$

$$
\begin{aligned}
|\epsilon_{k+1}| &= |\frac{\frac{1}{b}-x_{k+1}}{\frac{1}{b}}| \\
&= |1-b(2x_k-bx_k^2)| \\
&= |b^2x_k^2-2bx_k+1| \\
&= (1-bx_k)^2 \\
&= \epsilon_k^2
\end{aligned}
$$

(ii)

设 $g(x)=x-\frac{f(x)}{f'(x)}$

$\because\ f'(x)=\frac{1}{x^2}\neq 0$ 且 $f'(x)$ 在 $0<x_0<\frac{2}{b}$ 上连续

$\therefore$ 在 $(0,\frac{2}{b})$ 上，$f'(x)$ 不为0，g 连续，且 $g'(x)=\frac{f(x)f''(x)}{[f'(x)]^2}$ 也连续

又 $\because\ f(\frac{1}{b})=0\Rightarrow \lim_{x\to\frac{1}{b}} g'(x)=0$

由极限定义：对于 $\forall\, k$，存在 $0<\delta<\frac{1}{b}$，当 $0<|x-\frac{1}{b}|<\delta$ 时有

$$|g'(x)-0|<k$$

$\therefore$ 对于 $0<k<1$，存在 $0<\delta<\frac{1}{b}$，当 $0<|x-\frac{1}{b}|<\delta$ 时有

$$|g'(x)|<k$$

由于 $x=\frac{1}{b}$ 时，$|g'(x)|=0<k$，也满足条件

$\therefore$ 对于 $0<k<1$，存在 $0<\delta<\frac{1}{b}$，对于 $x\in(\frac{1}{b}-\delta,\frac{1}{b}+\delta)$ 有：

$$|g'(x)|\leq k$$

由中值定理：在 x 和 $\frac{1}{b}$ 之间存在 ξ，使得 $|g(x)-g(\frac{1}{b})|=|g'(\xi)|\ |x-\frac{1}{b}|$

$$|g(x)-\frac{1}{b}|=|g(x)-g(\frac{1}{b})|=|g'(\xi)|\ |x-\frac{1}{b}|\leq k|x-\frac{1}{b}|<|x-\frac{1}{b}|<\frac{1}{b}$$

$\therefore\ g(x)\in(0,\frac{2}{b})$

根据不动点定理，由于 $g(x)\in(0,\frac{2}{b})$，$x\in(0,\frac{2}{b})$，在 $x_n=g(x_{n-1})$，其中 $n\geq 1$，定义下的序列 $\{p_n\}_{n=1}^{\infty}$ 对于任意 $x\in(0,\frac{2}{b})$ 会收敛到 $\frac{1}{b}$

P 3

a

$x_1^{(2)}=0.500167$
$x_2^{(2)}=0.250804$
$x_3^{(2)}=-0.517387$

计算由c语言实现:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.141592653589793 //定义pi值
#define UNK 3 //设定未知量的数量
#define Iterations 2 //设定迭代次数

//函数声明
void Subtract_Row(double *Aj,double m,double *Ai,int c);
void Swap_Row(double**A,int p,int ip,int c);
int Find_Min(int r,int i,double **A);
double* Backward_Substitution(int r,double** A,double* ans);
double* Gauss(double** x,double* y,double* ans);
void Print(double* x);
double* F(double* x,double* y);
double** J(double* x,double** y);

int main(){
    //相关变量初始化和动态空间开辟
    double *x , *y, *neg_f,**j_matrix;
    j_matrix = (double**)malloc(sizeof(double*)*UNK);
    for(int i = 0;i<UNK;i++)
        j_matrix[i] = (double*)malloc(sizeof(double)*UNK);
    x = (double*)malloc(sizeof(double)*UNK);
    y = (double*)malloc(sizeof(double)*UNK);
    neg_f = (double*)malloc(sizeof(double)*UNK);
    int k = 1;

    //设定初始x = {0,0,0}
    for(int i = 0;i<UNK;i++)
        x[i] = 0;

    while(1){
        //计算F(x)和Jacobian矩阵
        F(x, neg_f);
        J(x,j_matrix);

        //求解方程 Jy = -F 中的y
        for(int i = 0;i<UNK;i++)
            neg_f[i] = -neg_f[i];
        Gauss(j_matrix,neg_f,y);

        for(int i = 0;i<UNK;i++)
            x[i] = x[i] + y[i];

        //打印迭代结果
        printf("---After %d Iterations---\n",k);
        Print(x);
        if(k==Iterations){
            break;
        }
        k++;
    }
```

```
    free(j_matrix);
    free(y);
    free(x);
    free(neg_f);
    return 0;
}

//设定F函数
double* F(double* x,double* y){
    y[0] = 3*x[0]-cos(x[1]*x[2])-1.0/2.0;
    y[1] = 4*x[0]*x[0]-625*x[1]*x[1]+2*x[1]-1;
    y[2] = exp(-x[0]*x[1])+20*x[2]+10.0*PI/3.0-1;
    return y;
}

//设定Jacobian矩阵
double** J(double* x,double** y){
    y[0][0] = 3;
    y[0][1] = x[2]*sin(x[1]*x[2]);
    y[0][2] = x[1]*sin(x[1]*x[2]);
    y[1][0] = 8*x[0];
    y[1][1] = -1250*x[1]+2;
    y[1][2] = 0;
    y[2][0] = -x[1]*exp(-x[0]*x[1]);
    y[2][1] = -x[0]*exp(-x[0]*x[1]);
    y[2][2] = 20;
    return y;
}

//打印x的结果
void Print(double* x){
    for(int i = 0;i<UNK;i++)
        printf("x%d = %.6lf\n",i+1,x[i]);
}


/*--- 以下为 Gaussian elimination 相关函数 ---*/

//第j+1行减去乘上m后的i+1行，目的在于使第j+1行且第i+1列的元素为0
void Subtract_Row(double *Aj,double m,double *Ai,int c){
    for(int i = 0;i<c;i++)
        Aj[i] -= m*Ai[i];
}

//交换第p+1行和第ip行
void Swap_Row(double**A,int p,int ip,int c){
    double t;
    for(int j =0;j<c;j++){
        t = A[p][j];
        A[p][j] = A[ip][j];
        A[ip][j] = t;
    }
}

//在第i+1列找到最小的j(i<=j<r)且满足该列第j+1行的元素非0
//如果没有返回-1
int Find_Min(int r,int i,double **A){
    int min = -1;
```

```
    for(int j = i;j<r;j++)
        if(A[j][i]!=0){
            min = j;
            break;
        }
    return min;
}

//通过反向替换求解x
double* Backward_Substitution(int r,double** A,double* ans){
    double sum_temp;
    ans[r-1] = A[r-1][r]/A[r-1][r-1];
    for(int i = r-2;i>=0;i--){
        sum_temp = 0;
        for(int j = i+1;j<r;j++)
            sum_temp+=A[i][j]*ans[j];
        ans[i]=(A[i][r] - sum_temp)/A[i][i];
        }
    return ans;
}

//Gaussian elimination主体函数
double* Gauss(double** x,double* y,double* ans){
    //初始化，读入矩阵数据，r为行数，c为列数
    int r = UNK,c = UNK + 1,p;
    double m;

    //构建增广矩阵
    double **A=(double**)malloc(sizeof(double*)*r);
    for(int i = 0;i<r;i++)
        A[i] = (double*)malloc(sizeof(double)*c);
    for(int i = 0;i<r;i++)
        for(int j = 0;j<c;j++){
            if(j != c-1)
                A[i][j] = x[i][j];
            else
                A[i][j] = y[i];
        }

    //Gaussian elimination
    for(int i =0;i<r-1;i++){
        p = Find_Min(r,i,A);
        //若找不到满足条件的p，方程无法求解返回NULL
        if(p==-1 && printf("No Unique Solution Exists!"))
            return NULL;
        //若p不在第i+1行，则需要将其换到i+1行
        if(p!=i)
            Swap_Row(A,p,i,c);
        //将矩阵主对角线以下部分消为0
        for(int j = i+1;j<r;j++){
            m = A[j][i]/A[i][i];
            Subtract_Row(A[j],m,A[i],c);
        }
    }

    //开始反向替换，通过变换后的矩阵求解x
    if(A[r-1][r-1]==0&& printf("No Unique Solution Exists!"))
        return NULL;
```

```
    Backward_Substitution(r,A,ans);
    for(int i = 0;i<r;i++)
        free(A[i]);
    return ans;
}

/*
输出（gcc version 8.2.0）：
---After 1 Iterations---
x1 = 0.500000
x2 = 0.500000
x3 = -0.523599
---After 2 Iterations---
x1 = 0.500167
x2 = 0.250804
x3 = -0.517387
*/
```

b

$x_1^{(2)} = 4.350877$
$x_2^{(2)} = 18.491228$
$x_3^{(2)} = -19.842105$

计算由c语言实现：

```
/* - - - - coding: GB 2312 - - - - */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.141592653589793 //定义pi值
#define UNK 3 //设定未知量的数量
#define Iterations 2 //设定迭代次数

//函数声明
void Subtract_Row(double *Aj,double m,double *Ai,int c);
void Swap_Row(double**A,int p,int ip,int c);
int Find_Min(int r,int i,double **A);
double* Backward_Substitution(int r,double** A,double* ans);
double* Gauss(double** x,double* y,double* ans);
void Print(double* x);
double* F(double* x,double* y);
double** J(double* x,double** y);

int main(){
    //相关变量初始化和动态空间开辟
    double *x , *y, *neg_f,**j_matrix;
    j_matrix = (double**)malloc(sizeof(double*)*UNK);
    for(int i = 0;i<UNK;i++)
        j_matrix[i] = (double*)malloc(sizeof(double)*UNK);
    x = (double*)malloc(sizeof(double)*UNK);
    y = (double*)malloc(sizeof(double)*UNK);
```

```
    neg_f = (double*)malloc(sizeof(double)*UNK);
    int k = 1;

    //设定初始x = {0,0,0}
    for(int i = 0;i<UNK;i++)
        x[i] = 0;

    while(1){
        //计算F(x)和Jacobian矩阵
        F(x, neg_f);
        J(x,j_matrix);

        //求解方程 Jy = -F 中的y
        for(int i = 0;i<UNK;i++)
            neg_f[i] = -neg_f[i];
        Gauss(j_matrix,neg_f,y);

        for(int i = 0;i<UNK;i++)
            x[i] = x[i] + y[i];

        //打印迭代结果
        printf("---After %d Iterations---\n",k);
        Print(x);
        if(k==Iterations){
            break;
        }
        k++;
    }
    free(j_matrix);
    free(y);
    free(x);
    free(neg_f);
    return 0;
}

//设定F函数
double* F(double* x, double* y){
    y[0] = x[0]*x[0]+x[1]-37;
    y[1] = x[0]-x[1]*x[1]-5;
    y[2] = x[0]+x[1]+x[2]-3;
    return y;
}

//设定Jacobian矩阵
double** J(double* x, double** y){
    y[0][0] = 2*x[0];
    y[0][1] = 1;
    y[0][2] = 0;
    y[1][0] = 1;
    y[1][1] = -2*x[1];
    y[1][2] = 0;
    y[2][0] = 1;
    y[2][1] = 1;
    y[2][2] = 1;
    return y;
}

//打印x的结果
```

```
void Print(double* x){
    for(int i = 0;i<UNK;i++)
        printf("x%d = %.6lf\n",i+1,x[i]);
}


/*--- 以下为 Gaussian elimination 相关函数 ---*/

//第j+1行减去乘上m后的i+1行，目的在于使第j+1行且第i+1列的元素为0
void Subtract_Row(double *Aj,double m,double *Ai,int c){
    for(int i = 0;i<c;i++)
        Aj[i] -= m*Ai[i];
}

//交换第p+1行和第ip行
void Swap_Row(double**A,int p,int ip,int c){
    double t;
    for(int j =0;j<c;j++){
        t = A[p][j];
        A[p][j] = A[ip][j];
        A[ip][j] = t;
    }
}

//在第i+1列找到最小的j(i<=j<r)且满足该列第j+1行的元素非0
//如果没有返回-1
int Find_Min(int r,int i,double **A){
    int min = -1;
    for(int j = i;j<r;j++)
        if(A[j][i]!=0){
            min = j;
            break;
        }
    return min;
}

//通过反向替换求解x
double* Backward_Substitution(int r,double** A,double* ans){
    double sum_temp;
    ans[r-1] = A[r-1][r]/A[r-1][r-1];
    for(int i = r-2;i>=0;i--){
        sum_temp = 0;
        for(int j = i+1;j<r;j++)
            sum_temp+=A[i][j]*ans[j];
        ans[i]=(A[i][r] - sum_temp)/A[i][i];
        }
    return ans;
}

//Gaussian elimination主体函数
double* Gauss(double** x,double* y,double* ans){
    //初始化，读入矩阵数据，r为行数，c为列数
    int r = UNK,c = UNK + 1,p;
    double m;

    //构建增广矩阵
    double **A=(double**)malloc(sizeof(double*)*r);
    for(int i = 0;i<r;i++)
```

```c
        A[i] = (double*)malloc(sizeof(double)*c);
    for(int i = 0;i<r;i++)
        for(int j = 0;j<c;j++){
            if(j != c-1)
                A[i][j] = x[i][j];
            else
                A[i][j] = y[i];
        }

    //Gaussian elimination
    for(int i =0;i<r-1;i++){
        p = Find_Min(r,i,A);
        //若找不到满足条件的p，方程无法求解返回NULL
        if(p==-1 && printf("No Unique Solution Exists!"))
            return NULL;
        //若p不在第i+1行，则需要将其换到i+1行
        if(p!=i)
            Swap_Row(A,p,i,c);
        //将矩阵主对角线以下部分消为0
        for(int j = i+1;j<r;j++){
            m = A[j][i]/A[i][i];
            Subtract_Row(A[j],m,A[i],c);
        }
    }

    //开始反向替换，通过变换后的矩阵求解x
    if(A[r-1][r-1]==0&& printf("No Unique Solution Exists!"))
        return NULL;
    Backward_Substitution(r,A,ans);
    for(int i = 0;i<r;i++)
        free(A[i]);
    return ans;
}

/*
输出（gcc version 8.2.0）：
---After 1 Iterations---
x1 = 5.000000
x2 = 37.000000
x3 = -39.000000
---After 2 Iterations---
x1 = 4.350877
x2 = 18.491228
x3 = -19.842105
*/
```

P 4

a

$x_1 = 1.04879$
$x_2 = 1.04829$
$x_3 = 0.92018$
$g = 0.15806$
The procedure completed, might have a minimum.

计算由c语言实现：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大可迭代次数
#define PI 3.1415926535898 //定义pi值
#define UNK 3 //设定未知量的数量
#define TOL 0.05 //设定误差

//计算向量二范数
double Norm(double* x){
    double n = 0;
    for(int i = 1;i<UNK;i++)
        n += x[i]*x[i];
    return sqrt(n);
}

//打印结果
void Print(double* x, double g1){
    for(int i = 0;i<UNK;i++)
        printf("x%d = %.5lf\n",i+1,x[i]);
    printf("g = %.5lf\n",g1);
}

//设定g函数
double g(double* x){
    double *y = (double*)malloc(sizeof(double)*UNK),z = 0;
    y[0] = 15*x[0]+x[1]*x[1]-4*x[2]-13;
    y[1] = x[0]*x[0]+10*x[1]-x[2]-11;
    y[2] = x[1]*x[1]*x[1]-25*x[2]+22;
    for(int i = 0;i<UNK;i++)
        z += y[i]*y[i];
    return z;
}

//设定g函数的梯度
double* nabla_g(double* x,double* y){
    y[0] = 30*(15*x[0]+x[1]*x[1]-4*x[2]-13)+4*x[0]*(x[0]*x[0]+10*x[1]-x[2]-11);
    y[1] = 4*x[1]*(15*x[0]+x[1]*x[1]-4*x[2]-13)+20*(x[0]*x[0]+10*x[1]-
x[2]-11)+6*x[1]*x[1]*(x[1]*x[1]*x[1]-25*x[2]+22);
    y[2] = -8*(15*x[0]+x[1]*x[1]-4*x[2]-13)-2*(x[0]*x[0]+10*x[1]-x[2]-11)-50*
(x[1]*x[1]*x[1]-25*x[2]+22);
    return y;
}

int main(){
    //相关变量初始化和动态空间开辟
    double *x , *z, *temp, gn,g0,g1,g2,g3,z0,an,a0,a1,a2,a3,h1,h2,h3;
    x = (double*)malloc(sizeof(double)*UNK);
    z = (double*)malloc(sizeof(double)*UNK);
    temp = (double*)malloc(sizeof(double)*UNK);
    int k = 1,f = 0;

    //设定初始x = {0,0,0}
```

```
    for(int i = 0;i<UNK;i++)
        x[i] = 0;

    while(k<Max_Iterations){
        g1 = g(x);
        nabla_g(x,z);
        z0=Norm(z);
        if(z0==0){
            printf("Zero Gradient\n");
            Print(x,g1);
            printf("The procedure completed, might have a minimum.");
            break;
        }

        //将z转化为单位向量
        for(int i = 0;i<UNK;i++)
            z[i] = z[i] / z0;
        a1 = 0;
        a3 = 1;
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a3*z[i];
        g3 = g(temp);

        while(g3>=g1){
            a3 = a3/2;
            for(int i = 0;i<UNK;i++)
                temp[i] = x[i]-a3*z[i];
            g3 = g(temp);
            if(a3<TOL/2.0){
                printf("No Likely Improvement\n");
                Print(x,g1);
                printf("The procedure completed, might have a minimum.");
                f = 1;
                break;
            }
        }

        //如果在上面的循环中已输出结果，则退出迭代
        if(f == 1)
            break;

        a2 = a3/2;
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a2*z[i];
        g2 = g(temp);

        h1 = (g2-g1)/a2;
        h2 = (g3-g2)/(a3-a2);
        h3 = (h2-h1)/a3;

        a0 = 0.5*(a2-h1/h3);
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a0*z[i];
        g0 = g(temp);

        gn = g0<g3?g0:g3;
        an = gn==g0?a0:a3;
        for(int i = 0;i<UNK;i++)
```

```
            x[i] = x[i]-an*z[i];
        if(fabs(gn-g1)<TOL){
            printf("Finished\n");
            Print(x,gn);
            printf("The procedure was successful.");
            break;
        }
        k++;
    }
    //超出迭代最高次数，迭代失败
    if(k>=Max_Iterations){
        printf("Maximum iterations exceeded\n");
        printf("The procedure was unsuccessful.");
    }
    free(temp);
    free(x);
    free(z);
    return 0;
}
/*
输出（gcc version 8.2.0）：
No Likely Improvement
x1 = 1.04879
x2 = 1.04829
x3 = 0.92018
g = 0.15806
The procedure completed, might have a minimum.
*/
```

b

$x_1 = 0.21601$
$x_2 = 0.36714$
$x_3 = -1.38362$
$g = 0.07441$
The procedure completed, might have a minimum.

计算由c语言实现：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大可迭代次数
#define PI 3.1415926535898 //定义pi值
#define UNK 3 //设定未知量的数量
#define TOL 0.05 //设定误差

//计算向量二范数
double Norm(double* x){
    double n = 0;
    for(int i = 1;i<UNK;i++)
        n += x[i]*x[i];
    return sqrt(n);
}
```

```
//打印结果
void Print(double* x, double g1){
    for(int i = 0;i<UNK;i++)
        printf("x%d = %.5lf\n",i+1,x[i]);
    printf("g = %.5lf\n",g1);
}

//设定g函数
double g(double* x){
    double *y = (double*)malloc(sizeof(double)*UNK),z = 0;
    y[0] = 10*x[0]-2*x[1]*x[1]+x[1]-2*x[2]-5;
    y[1] = 8*x[1]*x[1]+4*x[2]*x[2]-9;
    y[2] = 8*x[1]*x[2]+4;
    for(int i = 0;i<UNK;i++)
        z += y[i]*y[i];
    return z;
}

//设定g函数的梯度
double* nabla_g(double* x,double* y){
    y[0] = 20*(10*x[0]-2*x[1]*x[1]+x[1]-2*x[2]-5);
    y[1] = -8*x[1]*(10*x[0]-2*x[1]*x[1]+x[1]-2*x[2]-5)+32*x[1]*
(8*x[1]*x[1]+4*x[2]*x[2]-9)+16*x[2]*(8*x[1]*x[2]+4);
    y[2] = -4*(10*x[0]-2*x[1]*x[1]+x[1]-2*x[2]-5)+16*x[2]*
(8*x[1]*x[1]+4*x[2]*x[2]-9)+16*x[1]*(8*x[1]*x[2]+4);
    return y;
}

int main(){
    //相关变量初始化和动态空间开辟
    double *x , *z, *temp, gn,g0,g1,g2,g3,z0,an,a0,a1,a2,a3,h1,h2,h3;
    x = (double*)malloc(sizeof(double)*UNK);
    z = (double*)malloc(sizeof(double)*UNK);
    temp = (double*)malloc(sizeof(double)*UNK);
    int k = 1, f = 0;

    //设定初始x = {0,0,0}
    for(int i = 0;i<UNK;i++)
        x[i] = 0;

    while(k<Max_Iterations){
        g1 = g(x);
        nabla_g(x,z);
        z0=Norm(z);
        if(z0==0){
            printf("Zero Gradient\n");
            Print(x,g1);
            printf("The procedure completed, might have a minimum.");
            break;
        }

        //将z转化为单位向量
        for(int i = 0;i<UNK;i++)
            z[i] = z[i] / z0;
        a1 = 0;
        a3 = 1;
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a3*z[i];
```

```
        g3 = g(temp);

        while(g3>=g1){
            a3 = a3/2;
            for(int i = 0;i<UNK;i++)
                temp[i] = x[i]-a3*z[i];
            g3 = g(temp);
            if(a3<TOL/2.0){
                printf("No Likely Improvement\n");
                Print(x,g1);
                printf("The procedure completed, might have a minimum.");
                f = 1;
                break;
            }
        }

        //如果在上面的循环中已输出结果，则退出迭代
        if(f == 1)
            break;

        a2 = a3/2;
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a2*z[i];
        g2 = g(temp);

        h1 = (g2-g1)/a2;
        h2 = (g3-g2)/(a3-a2);
        h3 = (h2-h1)/a3;

        a0 = 0.5*(a2-h1/h3);
        for(int i = 0;i<UNK;i++)
            temp[i] = x[i]-a0*z[i];
        g0 = g(temp);

        gn = g0<g3?g0:g3;
        an = gn==g0?a0:a3;
        for(int i = 0;i<UNK;i++)
            x[i] = x[i]-an*z[i];
        if(fabs(gn-g1)<TOL){
            printf("Finished\n");
            Print(x,gn);
            printf("The procedure was successful.");
            break;
        }
        k++;
    }
    //超出迭代最高次数，迭代失败
    if(k>=Max_Iterations){
        printf("Maximum iterations exceeded\n");
        printf("The procedure was unsuccessful.");
    }
    free(temp);
    free(x);
    free(z);
    return 0;
}

/*
```

```
输出（gcc version 8.2.0）：
No Likely Improvement
x1 = 0.21601
x2 = 0.36714
x3 = -1.38362
g = 0.07441
The procedure completed, might have a minimum.
*/
```