NA_3

P 1

a

$$\begin{aligned}||\mathbf{x}-\tilde{\mathbf{x}}||_\infty &= \max\{0.2,0.5,0.4\}=0.5\\ ||A\tilde{\mathbf{x}}-\mathbf{b}||_\infty &= \max\{0,0.3,0.2\}=0.3\end{aligned}$$

b

$$\begin{aligned}||\mathbf{x}-\tilde{\mathbf{x}}||_\infty &= \max\{0.33,0.9,0.8\}=0.9\\ ||A\tilde{\mathbf{x}}-\mathbf{b}||_\infty &= \max\{0.27,0.16,0.21\}=0.27\end{aligned}$$

P 2

设 λ 为矩阵 $\mathbf{A}$ 的特征值，$\lambda_{\max}=\max|\lambda|$

$\because\ \mathbf{Ax}=\lambda\mathbf{x}$

$\therefore\ \begin{aligned}\mathbf{A}^2\mathbf{x}&=\lambda\cdot\mathbf{Ax}\\ \mathbf{A}^2\mathbf{x}&=\lambda^2\mathbf{x}\end{aligned}$

即 λ^2 为矩阵 $\mathbf{A}^2$ 的特征值

$\therefore \mathbf{A}^2$ 的最大绝对值特征值为 $\lambda^2_{\max}$

即 $\rho(\mathbf{A}^2)=\lambda^2_{\max}$

又 $\because\ ||\mathbf{A}||_2=[\rho(\mathbf{A}^T\mathbf{A})]^{1/2}$ 且矩阵 $\mathbf{A}$ 为对称矩阵

$\therefore\ ||\mathbf{A}||_2=[\rho(\mathbf{A}\cdot\mathbf{A})]^{1/2}=\lambda_{\max}=\rho(\mathbf{A})$

P 3

a

$x_1=10.000000$
$x_2=1.000000$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//找到矩阵第i+1行中的绝对值最大值
double Find_Max(double** A, int i, int r){
    double max = fabs(A[i][0]);
    for(int j = 1;j < r;j++)
        if(fabs(A[i][j])>max)
            max = fabs(A[i][j]);
    return max;
}

//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;
    }
}

//
int Find_P(double **A,double* s,int i,int r){
    int p = i;
    double max = fabs(A[i][i]) / s[i];
    for(int k = i;k<r;k++)
        if(fabs(A[k][i]) / s[k] > max){
            p = k;
            max = fabs(A[k][i]) / s[k];
        }
    return p;
}

//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;
}

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


int main(){
    //初始化，读入矩阵数据
    int r = 2,c = 3,p;
    double m,*x,*temp;
    x = (double*)malloc(sizeof(double)*r);

    double A1[3] = {0.03, 58.9, 59.2};
    double A2[3] = {5.31, -6.10, 47.0};
    double* A[2] = {A1,A2};
    double s[2];
    for(int i = 0;i<r;i++){
        s[i] = Find_Max(A, i, r);
        if(s[i] == 0 && printf("No Unique Solution Exists!"))
            return 0;
    }

    //Gaussian elimination
    for(int i =0;i<r-1;i++){
        p = Find_P(A, s, i, r);
        //若找不到满足条件的p，方程无法求解返回NULL
        if(A[p][i] == 0 && printf("No Unique Solution Exists!"))
            return 0;
        //若p+1不是第i+1行，则需要将其与第i+1行互换
        if(p!=i){
            temp = A[p];
            A[p] = A[i];
            A[i] = temp;
        }
        //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 0;
    Backward_Substitution(r,A,x);
    Print(x, r);
    return 0;
}

/***************************
输出（gcc version 8.2.0）：
x1 = 10.000000
x2 = 1.000000
***************************/
```

b

$x_1 = 0.000000$
$x_2 = 10.000000$
$x_3 = 0.142857$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//找到矩阵第i+1行中的绝对值最大值
double Find_Max(double** A, int i, int r){
    double max = fabs(A[i][0]);
    for(int j = 1;j < r;j++)
        if(fabs(A[i][j])>max)
            max = fabs(A[i][j]);
    return max;
}

//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;
    }
}

//
int Find_P(double **A,double* s,int i,int r){
    int p = i;
    double max = fabs(A[i][i]) / s[i];
    for(int k = i;k<r;k++)
        if(fabs(A[k][i]) / s[k] > max){
            p = k;
            max = fabs(A[k][i]) / s[k];
        }
    return p;
}

//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;
}

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


int main(){
    //初始化，读入矩阵数据
    int r = 3,c = 4,p;
    double m,*x,*temp;
    x = (double*)malloc(sizeof(double)*r);

    double A1[4] = {3.03, -12.1, 14, -119};
    double A2[4] = {-3.03, 12.1, -7, 120};
    double A3[4] = {6.11, -14.20, 21,-139};
    double* A[3] = {A1,A2,A3};
    double s[3];

    for(int i = 0;i<r;i++){
        s[i] = Find_Max(A, i, r);
        if(s[i] == 0 && printf("No Unique Solution Exists!"))
            return 0;
    }

    //Gaussian elimination
    for(int i =0;i<r-1;i++){
        p = Find_P(A, s, i, r);
        //若找不到满足条件的p，方程无法求解返回NULL
        if(A[p][i] == 0 && printf("No Unique Solution Exists!"))
            return 0;
        //若p+1不是第i+1行，则需要将其与第i+1行互换
        if(p!=i){
            temp = A[p];
            A[p] = A[i];
            A[i] = temp;
        }
        //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 0;
    Backward_Substitution(r,A,x);
    Print(x, r);
    return 0;
}

/***************************
输出（gcc version 8.2.0）：
x1 = 0.000000
x2 = 10.000000
x3 = 0.142857
***************************/
```

P 4

a

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大迭代次数
#define TOL 0.001 //设定迭代终止误差限

//计算无穷范数
double Norm(double* x,int r){
    double n = fabs(x[0]);
    for(int i = 1;i<r;i++)
        if(fabs(x[i])>n)
            n = fabs(x[i]);
    return n;
}

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

int main(){
    //初始化，写入相关数据
    int k = 1,r = 3, c = 4;
    double xo[3] = {0,0,0};
    double x[3] = {0,0,0};
    double temp[3] = {0,0,0};
    double a[3][3] = {{4,1,-1},{-1,3,1},{2,2,5}};
    double b[3] = {5,-4,1};
    double sum  = 0;

    //the Jacobi method
    while(k<Max_Iterations){
        for(int i = 0;i<r;i++){
            sum = 0;
            for(int j = 0;j<r;j++){
                if(j==i)
                    continue;
                else
                    sum += a[i][j]*xo[j];
            }
            x[i] = 1.0/a[i][i]*(b[i]-sum);
        }
        //打印前三次迭代结果
        if(k<=3){
            printf("--- After %d Iterations ---\n",k);
            Print(x,r);
        }
        for(int i = 0;i<r;i++)
            temp[i] = x[i] - xo[i];
        if(Norm(temp, r)<TOL){
            printf("--- Finished After %d Iterations ---\n",k);
            Print(x,r);
            printf("The procedure was successful.");
            break;
        }
        k++;
        for(int i = 0;i<r;i++)
            xo[i] = x[i];
    }
    //如果迭代次数超过上限，则输出迭代失败
    if(k>=Max_Iterations)
        printf("Maximum number of iterations exceeded");
    return 0;
}

/**************************
输出（gcc version 8.2.0）：
--- After 1 Iterations ---
x1 = 1.25000
x2 = -1.33333
x3 = 0.20000
--- After 2 Iterations ---
x1 = 1.63333
x2 = -0.98333
x3 = 0.23333
--- After 3 Iterations ---
x1 = 1.55417
x2 = -0.86667
x3 = -0.06000
--- Finished After 10 Iterations ---
x1 = 1.44764
x2 = -0.83556
x3 = -0.04502
The procedure was successful.
**************************/
```

b

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大迭代次数
#define TOL 0.001 //设定迭代终止误差限

//计算无穷范数
double Norm(double* x,int r){
    double n = fabs(x[0]);
    for(int i = 1;i<r;i++)
        if(fabs(x[i])>n)
            n = fabs(x[i]);
    return n;
}

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

```
int main(){
    //初始化，写入相关数据
    int k = 1,r = 3, c = 4;
    double xo[3] = {0,0,0};
    double x[3] = {0,0,0};
    double temp[3] = {0,0,0};
    double a[3][3] = {{-2,1,0.5},{1,-2,-0.5},{0,1,2}};
    double b[3] = {4,-4,0};
    double sum  = 0;

    //the Jacobi method
    while(k<Max_Iterations){
        for(int i = 0;i<r;i++){
            sum = 0;
            for(int j = 0;j<r;j++){
                if(j==i)
                    continue;
                else
                    sum += a[i][j]*xo[j];
            }
            x[i] = 1.0/a[i][i]*(b[i]-sum);
        }
        //打印前三次迭代结果
        if(k<=3){
            printf("--- After %d Iterations ---\n",k);
            Print(x,r);
        }
        for(int i = 0;i<r;i++)
            temp[i] = x[i] - xo[i];
        if(Norm(temp, r)<TOL){
            printf("--- Finished After %d Iterations ---\n",k);
            Print(x,r);
            printf("The procedure was successful.");
            break;
        }
        k++;
        for(int i = 0;i<r;i++)
            xo[i] = x[i];
    }
    //如果迭代次数超过上限，则输出迭代失败
    if(k>=Max_Iterations)
        printf("Maximum number of iterations exceeded.");
    return 0;
}

/***************************
输出（gcc version 8.2.0）：
--- After 1 Iterations ---
x1 = -2.00000
x2 = 2.00000
x3 = 0.00000
--- After 2 Iterations ---
x1 = -1.00000
x2 = 1.00000
x3 = -1.00000
--- After 3 Iterations ---
x1 = -1.75000
x2 = 1.75000
```

```
x3 = -0.50000
--- Finished After 21 Iterations ---
x1 = -1.45486
x2 = 1.45486
x3 = -0.72704
The procedure was successful.
**************************/
```

P 5

a

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大迭代次数
#define TOL 0.001 //设定最大迭代次数

//计算无穷范数
double Norm(double* x,int r){
    double n = fabs(x[0]);
    for(int i = 1;i<r;i++)
        if(fabs(x[i])>n)
            n = fabs(x[i]);
    return n;
}

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

int main(){
    //初始化，写入相关数据
    int k = 1,r = 3, c = 4;
    double xo[3] = {0,0,0};
    double x[3] = {0,0,0};
    double temp[3] = {0,0,0};
    double a[3][3] = {{3,-1,1},{3,6,2},{3,3,7}};
    double b[3] = {1,0,4};
    double sum1  = 0, sum2 = 0, sum = 0;

    //分别用两种方法进行迭代
    for(int method = 0;method<2;method++){
        if(method == 0)
            printf("--- Using the Jacobi method ---\n");
        else
            printf("--- Using the Gauss-Seidel method ---\n");
        while(k<Max_Iterations){
            if(method == 0)
            //Jacobi method下的迭代计算
                for(int i = 0;i<r;i++){
                    sum = 0;
                    for(int j = 0;j<r;j++){
                        if(j==i)
                            continue;
                        else
                            sum += a[i][j]*xo[j];
                    }
                    x[i] = 1.0/a[i][i]*(b[i]-sum);
                }
            else
            //Gauss-Seidel method下的迭代计算
                for(int i = 0;i<r;i++){
                    sum1 = 0;
                    sum2 = 0;
                    for(int j = 0;j<i;j++)
                        sum1 += a[i][j]*x[j];
                    for(int j = i+1;j<r;j++)
                        sum2 += a[i][j]*xo[j];
                    x[i] = 1.0/a[i][i]*(b[i]-sum1-sum2);
                }
            //printf("--- After %d Iterations ---\n",k);
            //Print(x,r);
            for(int i = 0;i<r;i++)
                temp[i] = x[i] - xo[i];
            if(Norm(temp, r)<TOL){
                printf("--- Finished Successfully After %d Iterations ---\n",k);
                Print(x,r);
                break;
            }
            k++;
            for(int i = 0;i<r;i++)
                xo[i] = x[i];
        }
        //如果迭代次数超过上限，则输出迭代失败
        if(k>=Max_Iterations)
            printf("Maximum number of iterations exceeded");
    }
    return 0;
}

/***************************
输出（gcc version 8.2.0）：
--- Using the Jacobi method ---
--- Finished Successfully After 9 Iterations ---
x1 = 0.03510
x2 = -0.23664
x3 = 0.65813
--- Using the Gauss-Seidel method ---
--- Finished Successfully After 9 Iterations ---
x1 = 0.03510
x2 = -0.23668
x3 = 0.65782
***************************/
```

b

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Max_Iterations 1000 //设定最大迭代次数
#define TOL 0.001 //设定迭代终止误差限

//计算无穷范数
double Norm(double* x,int r){
    double n = fabs(x[0]);
    for(int i = 1;i<r;i++)
        if(fabs(x[i])>n)
            n = fabs(x[i]);
    return n;
}

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

int main(){
    //初始化，写入相关数据
    int k = 1,r = 3, c = 4;
    double xo[3] = {0,0,0};
    double x[3] = {0,0,0};
    double temp[3] = {0,0,0};
    double a[3][3] = {{10,-1,0},{-1,10,-2},{0,-2,10}};
    double b[3] = {9,7,6};
    double sum1  = 0, sum2 = 0, sum = 0;

    //分别用两种方法进行迭代
    for(int method = 0;method<2;method++){
        if(method == 0)
            printf("--- Using the Jacobi method ---\n");
        else
            printf("--- Using the Gauss-Seidel method ---\n");
        while(k<Max_Iterations){
            if(method == 0)
            //Jacobi method下的迭代计算
                for(int i = 0;i<r;i++){
                    sum = 0;
                    for(int j = 0;j<r;j++){
                        if(j==i)
                            continue;
                        else
                            sum += a[i][j]*xo[j];
                    }
                    x[i] = 1.0/a[i][i]*(b[i]-sum);
                }
            else
            //Gauss-Seidel method下的迭代计算
                for(int i = 0;i<r;i++){
                    sum1 = 0;
                    sum2 = 0;
                    for(int j = 0;j<i;j++)
                        sum1 += a[i][j]*x[j];
                    for(int j = i+1;j<r;j++)
                        sum2 += a[i][j]*xo[j];
                    x[i] = 1.0/a[i][i]*(b[i]-sum1-sum2);
                }
            //printf("--- After %d Iterations ---\n",k);
            //Print(x,r);
            for(int i = 0;i<r;i++)
                temp[i] = x[i] - xo[i];
            if(Norm(temp, r)<TOL){
                printf("--- Finished Successfully After %d Iterations ---\n",k);
                Print(x,r);
                break;
            }
            k++;
            for(int i = 0;i<r;i++)
                xo[i] = x[i];
        }
        //如果迭代次数超过上限，则输出迭代失败
        if(k>=Max_Iterations)
            printf("Maximum number of iterations exceeded");
    }
    return 0;
}

/**************************
输出（gcc version 8.2.0）：
--- Using the Jacobi method ---
--- Finished Successfully After 6 Iterations ---
x1 = 0.99572
x2 = 0.95778
x3 = 0.79145
--- Using the Gauss-Seidel method ---
--- Finished Successfully After 6 Iterations ---
x1 = 0.99572
x2 = 0.95779
x3 = 0.79156
**************************/
```