**第三章作业**

贺自怡 3140100524

**Navigator**

[作业要求](#作业要求)

程序

**直接法**

高斯直接

<[代码](#C1) [结果](#R1)>

高斯列主元

<[代码](#C2) [结果](#R2)>

LU分解

<[代码](#C3) [结果](#R3)>

**迭代法**

Jacobi迭代

<[代码](#C4) [结果](#R4)>

Gauss-Seidel迭代

<[代码](#C5) [结果](#R5)>

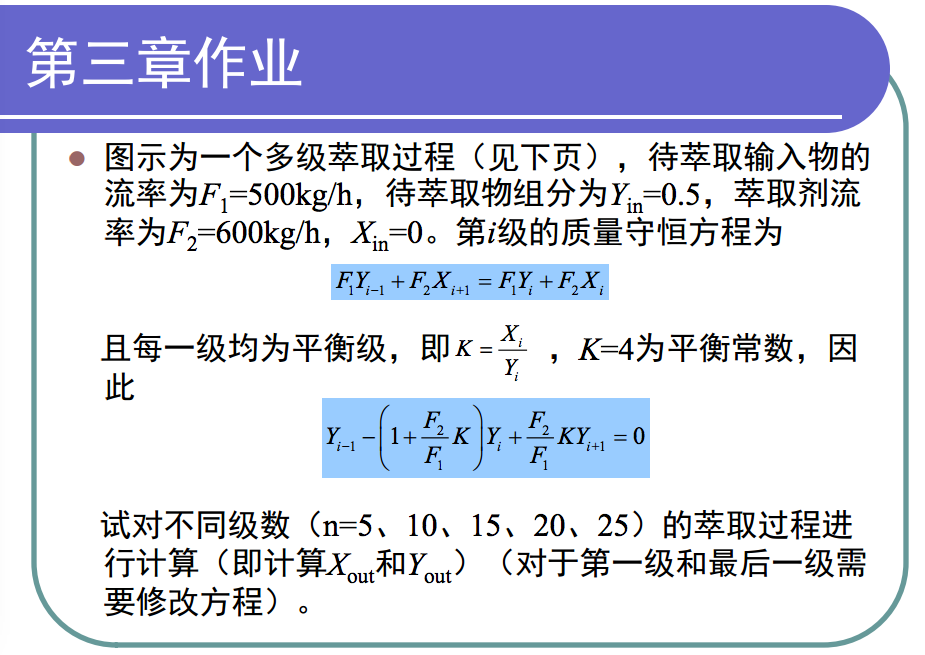
SOR迭代

<[代码](#C6) [结果](#R6)>

[总结](#Con)

**Content**

1．作业要求



第i级的质量守恒方程为：

5Yi-1 + 6Xi+1 = 5Yi + 6Xi

i=1时 Xout –X2 = 5/6 \*（Yin – Y1）

累加得到Xout – Xin = 5/6\*（Yin-Yout）

即 Xout = 5/6 \*（0.5-Yout）

Yi-1 – 5.8Yi + 4.8Yi+1 = 0;

列出要解的方程为

-5.8Y1 + 4.8Y2 = - 0.5

Y1 -5.8Y2 + 4.8Y3 = 0

Y2 -5.8Y3 + 4.8Y4 = 0

…

Yn-2 – 5.8Yn-1+4.8Yn = 0

Yn-1 -5.8Yn = 0

2．程序实现

主程序

|  |
| --- |
| % main    List = [5 10 15 20 25];  x = zeros(5,6);  y = zeros(5,6);  standard\_y = zeros(5,1);  standard\_x = zeros(5,1);  for i = 1:5        index = List(i);      [a,b] = createEquation(index);        temp = vpa(a\b,20);      standard\_y(i) = temp(index);      standard\_x(i) = 5/6\*(0.5-standard\_y(i));        %% 高斯直接      result1 = vpa(gauss\_direct(a,b),20);      y(i,1) = result1(index);      x(i,1) = 5/6\*(0.5-y(i,1));        %% 高斯列主元      result2 = vpa(gauss\_column(a,b),20);      y(i,2) = result2(index);      x(i,2) = 5/6\*(0.5-y(i,2));        %% LU分解法      result3 = vpa(lumethod(a,b),20);      y(i,3) = result3(index);      x(i,3) = 5/6\*(0.5-y(i,3));        %% Jacobi      [result4] = vpa(jacobi(a,b),20);      k(i,1) = size(result4,2);      y(i,4) = vpa(result4(index,k(i,1)),20);      x(i,4) = vpa(5/6\*(0.5-y(i,4)),20);        %% Gauss-Seidel      [result5] = vpa(gaussSeidel(a,b),20);      k(i,2) = size(result5,2);      y(i,5) = vpa(result5(index,k(i,2)),20);      x(i,5) = vpa(5/6\*(0.5-y(i,5)),20);        %% SOR      [result6] = vpa(sor(a,b,1),20);      k(i,3) = size(result6,2);      y(i,6) = vpa(result6(index,k(i,3)),20);      x(i,6) = vpa(5/6\*(0.5-y(i,6)),20);    end  figure;hold on;  for i = 1:5      plot(1:6,x(i,1:6));  end |

**直接法**

1. 高斯消去法

Code

|  |
| --- |
| function result = gauss\_direct(a,b)  %% Guass  equa = [a b];  origin  = equa;  num = size(origin,1);    %% elimination  for i=1:num-1                      % 每次选取一个主元      if (origin(i,i) == 0)           % 这一项是0的话可以不用消后面的了          continue;      end      for j= i+1:num                 % 对后几行消元          origin(j,:) = origin(j,:) - origin(j,i)/origin(i,i)\*origin(i,:);      end  end    %% inverse  origin(num,:) = origin(num,:)/origin(num,num);  for i = num-1:-1:1      for j = i+1:num                 % 倒着对后面几项都消元          origin(i,:) = origin(i,:)-origin(i,j)/origin(j,j)\*origin(j,:);      end      origin(i,:) = origin(i,:)/origin(i,i);  end  result = origin(:,size(origin,2)); |

Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 1.55360885e-4 | 0 | 0.416537199262342 | 0 |
| 10 | 6.09677270e-8 | 0 | 0.416666615860228 | 0 |
| 15 | 2.3927306e-11 | 0 | 0.416666666646727 | 0 |
| 20 | 9.3904762e-15 | -1.680130e-16 | 0.416666666666659 | 0 |
| 25 | 3.6853728e-18 | -2.090350e-16 | 0.416666666666667 | 0 |

1. 列主元消去法

Code

|  |
| --- |
| function result = gauss\_column(a,b)  %% Guass  equa = [a b];  origin  = equa;  num = size(origin,1);    %% elimination  for i=1:num-1                       % 每次选取一个主元      temp = abs(origin(i:num,i));      [~,line] = max(temp);      if (line~=1)                    % 这一行的i列数据不是最大的，则要交换          temp1 = origin(i,:);          origin(i,:) = origin(line+i-1,:);          origin(line+i-1,:) = temp1;      end      if (origin(i,i) == 0)           % 这一项是0的话可以不用消后面的了          continue;      end      for j= i+1:num                  % 对后几行消元          origin(j,:) = origin(j,:) - origin(j,i)/origin(i,i)\*origin(i,:);      end  end    %% inverse  origin(num,:) = origin(num,:)/origin(num,num);  for i = num-1:-1:1      for j = i+1:num                 % 倒着对后面几项都消元          origin(i,:) = origin(i,:)-origin(i,j)/origin(j,j)\*origin(j,:);      end      origin(i,:) = origin(i,:)/origin(i,i);  end    %% validate  result = origin(:,size(origin,2));  % ans(:,1) = equa(:,1:num)\*result;    % 代回去计算方程右侧  % ans(:,2) = ans(:,1) - equa(:,size(origin,2));  % 计算误差 |

Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 1.55360885e-4 | 0 | 0.416537199262342 | 0 |
| 10 | 6.09677270e-8 | 0 | 0.416666615860228 | 0 |
| 15 | 2.3927306e-11 | 0 | 0.416666666646727 | 0 |
| 20 | 9.3904762e-15 | -1.680130e-16 | 0.416666666666659 | 0 |
| 25 | 3.6853728e-18 | -2.090350e-16 | 0.416666666666667 | 0 |

结果和高斯直接消去一样。

1. LU分解法

Code

|  |
| --- |
| function [u]=lumethod(A,B)  n = size(B,1);  flag=0;  [A,h,flag]=decompose(A,n,eps,flag);     % h中放置行序号  if flag~=-1      [u]=back1(A,h,n,B);  end  end  %% decompose  function [A,h,flag]=decompose(A,n,tol,flag)  h=zeros(n,1);                           % 预分配内存  s=zeros(n,1);  for i=1:n      h(i)=i;      s(i)=abs(A(i,1));      for j=2:n          if abs(A(i,j))>s(i)              s(i)=abs(A(i,j));               % 选行最大元素          end      end  end  for k=1:n-1      [h]=change1(A,h,s,n,k);      if abs(A(h(k),k)/s(h(k)))<tol           % 判断是否有0          flag=-1;          break;      end      for i=k+1:n          factor=A(h(i),k)/A(h(k),k);          A(h(i),k)=factor;                   %L放入A          for j=k+1:n                         % U放入A              A(h(i),j)=A(h(i),j)-factor\*A(h(k),j);          end      end  end  if abs(A(h(k),k)/s(h(k)))<tol      flag=-1;  end  end  %% inverse  function [u]=back1(A,h,n,B)  for i=2:n      sum=B(h(i));      for j=1:i-1          sum=sum-A(h(i),j)\*B(h(j));      end      B(h(i))=sum;                                % 前向代入求U  end  u(n)=B(h(n))/A(h(n),n);  for i=n-1:-1:1      sum=0;      for j=i+1:n          sum=sum+A(h(i),j)\*u(j);      end      u(i)=(B(h(i))-sum)/A(h(i),i);               % 后向代入求U  end  u=u';  end  %% inverse  function [h]=change1(A,h,s,n,k)  p=k;                                            % 记录行序号  big=abs(A(h(k),k)/s(h(k)));  for ii=k+1:n      dummy=abs(A(h(ii),k)/s(h(ii)));      if dummy>big          big=dummy;          p=ii;      end  end  dummy=h(p);                                     % 主元行序号交换  h(p)=h(k);  h(k)=dummy;  end |

Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 1.55360885e-4 | 0 | 0.416537199262342 | 0 |
| 10 | 6.09677270e-8 | 0 | 0.416666615860228 | 0 |
| 15 | 2.3927306e-11 | 0 | 0.416666666646727 | 0 |
| 20 | 9.3904762e-15 | -1.680130e-16 | 0.416666666666659 | 0 |
| 25 | 3.6853728e-18 | -2.090350e-16 | 0.416666666666667 | 0 |

结果同上

迭代法

1. Jacobi迭代法

Code

|  |
| --- |
| function [x,i]=jacobi(a,b)  n = size(b,1);  D = diag(diag(a));                  % 计算对角线上元素组成的向量  N = D-a;                            % 计算运算所需各矩阵  G = D^-1\*N;  f = D^-1\*b;  max\_iter = 1e10;                       % 设置迭代最大次数，防止死循环  x(:,1) = zeros(n,1);                     % 生成初始向量    for i=2:max\_iter                       % 进行运算      x(:,i) = G\*x(:,i-1) + f;      if isnan(sum(x(:,i)))          break;      end      err(i) = norm(x(:,i)-x(:,i-1))/norm(x(:,i));      if err(i) < eps % 判断是否符合跳出循环条件          break;      end  end  plot(2:i-1,err(2:i-1)); |

Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 迭代次数 | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 83 | 1.55360885e-4 | -5.233953e-15 | 0.416537199262342 | 0 |
| 10 | 103 | 6.09677270e-8 | -2.987024e-13 | 0.416666615860228 | 0 |
| 15 | 106 | 2.3927306e-11 | -2.032465e-11 | 0.416666666646727 | 0 |
| 20 | 107 | 9.3904762e-15 | -7.549103e-10 | 0.416666666666667 | 1.8784973e-14 |
| 25 | 107 | 3.6853727e-18 | -2.984348e-08 | 0.416666666666667 | 0 |

1. Gauss-Seidel迭代法

Code

|  |
| --- |
| function [x,i] = gaussSeidel(a,b)  n = size(b,1);  D = diag(diag(a));                  % 运算中所需各矩阵  L = -tril(a, -1);  U = -triu(a,1);  G=(D-L)^-1\*U;  f=(D-L)^-1\*b;  max\_iter = 1e10;                       % 设置迭代最大次数，防止死循环  x(:,1) = zeros(n,1);                     % 生成向量  for i=2:max\_iter                       % 进行运算      x(:,i) = G\*x(:,i-1) + f;      err(i) = norm(x(:,i)-x(:,i-1))/norm(x(:,i));      if err(i) < eps         % 判断是否符合跳出循环条件          break;      end  end  plot(2:i-1,err(2:i-1)); |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 迭代次数 | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 43 | 1.55360885e-4 | 0 | 0.416537199262342 | 0 |
| 10 | 53 | 6.09677270e-8 | -1.107109e-14 | 0.416666615860228 | 0 |
| 15 | 54 | 2.3927306e-11 | -3.057334e-13 | 0.416666666646727 | 0 |
| 20 | 55 | 9.3904762e-15 | -2.322611e-12 | 0.416666666666667 | 1.8784973e-14 |
| 25 | 55 | 3.6853728e-18 | -2.055399e-11 | 0.416666666666667 | 0 |

Result

1. SOR迭代法

Code

|  |
| --- |
| function [result,i] = sor(a,b,w)  n = size(b,1);  D = diag(diag(a));                  % 运算所需各矩阵  L = -tril(a, -1);  U = -triu(a,1);  G = (D- w\*L)\((1-w)\*D + w\* U );  f = (D-L)\(b\*w);  i\_max = 1e10;                       % 设置迭代最大次数，防止死循环  err=eps;                            % 近似相对误差，作为迭代终止条件  y(:,1) = zeros(n,1);                     % 迭代初始向量    for i = 2:i\_max      y(:,i) = G\*y(:,i-1) + f;                       % 进行运算      if norm(y(:,i)-y(:,i-1))/norm(y(:,i-1)) < err   % 判断是否符合跳出循环条件          break;      end  end    result = y; |

Result

先测试不同的松弛因子下，yout相对误差和计算迭代的次数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 0.9 | | 1.0 | | 1.1 | |
| 5 | 0.3164235996725 | 55 | 0 | 43 | -0.31642359967 | 30 |
| 10 | 0.4904208775446 | 68 | -1.128817e-14 | 53 | -0.49042087754 | 39 |
| 15 | 0.5579701628570 | 70 | -3.060035e-13 | 54 | -0.55797016285 | 41 |
| 20 | 0.5841934410385 | 70 | -2.322947e-12 | 55 | -0.58419344105 | 41 |
| 25 | 0.5943735680378 | 70 | -1.168129e-11 | 56 | -0.59437356817 | 41 |

可以看出松弛因子为1时结果较精确。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | Yout | Yout相对误差 | Xout | Xout相对误差 |
| 5 | 1.5536088e-04 | 0 | 0.416537199262342 | 0 |
| 10 | 6.09677270e-8 | -1.128817e-14 | 0.416666615860228 | 0 |
| 15 | 2.3927306e-11 | -3.060035e-13 | 0.416666666646727 | 0 |
| 20 | 9.3904762e-15 | -2.322947e-12 | 0.416666666666667 | 1.8784973e-14 |
| 25 | 3.6853728e-18 | -1.168129e-11 | 0.416666666666667 | 0 |

3．总结

*所有方法Yout相对误差比较*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 5 | 10 | 15 | 20 | 25 |
| 高斯直接 | 0 | 0 | 0 | -1.68013e-16 | -2.09035e-16 |
| 高斯列主 | 0 | 0 | 0 | -1.68013e-16 | -2.09035e-16 |
| LU分解 | 0 | 0 | 0 | -1.68013e-16 | -2.09035e-16 |
| Jacobi | 1.5536088e-4 | 6.0967727e-8 | 2.392730e-11 | 9.390476e-15 | 3.685372e-18 |
| Gauss-S | 1.5536088e-4 | 6.0967727e-8 | 2.392731e-11 | 9.390476e-15 | 3.685373e-18 |
| SOR | 0 | -1.12882e-14 | -3.06004e-13 | -2.32295e-12 | -1.16813e-11 |

总体而言，直接法的精度高于迭代法，直接法各个方法结果近似，是因为题目的方程不是病态方程。

*迭代法各法比较*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 迭代次数 | 5 | 10 | 15 | 20 | 25 |
| Jacobi | 83 | 103 | 106 | 107 | 107 |
| Gauss-S | 43 | 53 | 54 | 55 | 55 |
| SOR | 43 | 53 | 54 | 55 | 56 |

各种迭代方法的迭代次数都随着级数增加而增加，Gauss-Seidel和松弛因子为1的SOR迭代次数差不多，都比Jacobi小，而Jacobi和Gauss-Seidel精度都随n增加而提高，SOR精度下降，因此Gauss-Seidel在这里表现较好。