Практическая работа 16.
Решение параболических уравнений в системе MatLab

Цель работы: приобрести навыки численного решения параболических уравнений с различными начальными условиями по методу Кранка-Николсона в системе MatLab.

Теоретические сведения

Параболические уравнения

Параболические уравнения — класс дифференциальных уравнений в частных производных. Описывают нестационарные процессы. Классическим примером уравнения параболического типа является уравнение теплопроводности (диффузии). В одномерном по пространству случае однородное (без источников энергии) уравнение теплопроводности имеет вид

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2},\ 0 < x < l,\ t > 0$$

Если на границах $x = 0$ и $x = l$ заданы значения искомой функции $u(x,t)$ в виде

$$u(0,t) = \varphi_1(t),\ x = 0,\ t > 0,$$
$$u(l,t) = \varphi_2(t),\ x = l,\ t > 0,$$

т.е. граничные условия первого рода, и , кроме того заданы начальные условия

$$u(x,0) = \psi(x),\ 0 \le x \le l,\ t = 0,$$

то задачу называют первой начально-краевой задачей для уравнения теплопроводности.

Основные определения, связанные с методом конечных разностей, рассмотрим на примере конечно-разностного решения первой начально-краевой задачи для уравнения теплопроводности.

Согласно методу сеток в плоской области D строится сеточная область Dh, состоящая из одинаковых ячеек. При этом область Dh должна как можно лучше приближать область D. Сеточная область (то есть сетка) Dh состоит из изолированных точек, которые называются узлами сетки. Число узлов будет характеризоваться основными размерами сетки h: чем меньше h, тем больше узлов содержит сетка. Узел сетки называется внутренним, если он принадлежит области D, а все соседние узлы принадлежат сетке Dh. В противном случае он называется граничным. Совокупность граничных узлов образует границу сеточной области Гh.

Сетка может состоять из клеток разной конфигурации: квадратных, прямоугольных, треугольных и других. После построения сетки исходное дифференциальное уравнение заменяется разностным уравнением во всех внутренних узлах сетки. Затем на основании граничных условий устанавливаются значения искомого решения в граничных узлах. Присоединяя граничные условия сеточной задачи к разностным уравнениям, записанных для внутренних узлов, получаем систему уравнений, из которой определяем значения искомого решения во всех узлах сетки.

На введенной сетке вводят сеточные функции , первая из которых известна, вторая подлежит определению. Для определения в задаче заменяют (аппроксимируют) дифференциальные операторы отношением конечных разностей.

В случае явных схем значения функции в узле очередного слоя можно найти, зная значения в узлах предыдущих слоев. В случае неявных схем для нахождения значений решения в узлах очередного слоя приходится решать систему уравнений. Для проведения вычислений самой простой схемой оказывается первая: достаточно на основании начального условия найти значения функции в узлах слоя 0 , чтобы в дальнейшем последовательно определять значения решения в узлах слоев 1, 2, и т.д. В случае второй схемы, которая является неявной, обязательно приходится решать систему уравнений для нахождения решения сеточной задачи. В любом случае согласно методу сеток будем иметь столько уравнений, сколько имеется неизвестных (значения искомой функции в узлах). Число неиз-

вестных равно числу всех узлов сетки. Решая систему уравнений, получаем решение поставленной задачи.

Разрешимость этой системы для явных схем вопросов не вызывает, так как все действия выполняются в явно определенной последовательности. В случае неявных схем разрешимость системы следует исследовать в каждом конкретном случае.

Последовательность действий

Для решения данной работы требуется составление и заполнение матрицы результатов числовых расчётов.

Сначала выполняется заполнение начальных значений
Border values, X, T, coeff, N, M, Result_matrix;

Например,

```
clear all;
m=10;
n=10;
x=10;
t=5;
k=1;
a=0;
b=0;
N=0;

starttime=clock;
```

И проведение подготовительных вычислений, таких как шаг матрицы:

```
Grid_x -> x/n;
Grid_t -> t/m;
r -> coeff^2*Grid_x/Grid_t^2;
s1 -> 2+2/r;
s2 -> 2/r-2;
Output -> zeros(n,m);
```

Например,

```
Grid_x=x/n;
Grid_t=t/m;
r=(k*k*Grid_x)/((Grid_t)^2);
s1=2+2/r;
s2=2/r-2;
Output=zeros(n,m);
```

После выполнения подготовительных вычислений можно переходить к выполнению расчётов. Заполняем уже известные граничные значения.

```
Output(1,1:m) -> border_value_1;
Output(n,1:m) -> border_value_2;
```

Например,

```
Output(1,1:m)=a;
Output(n,1:m)=b;
```

Далее переходим к заполнению самой матрицы. Заполняем первую строку. Функция f(x) — соответственно варианту.

```
Output(2:n-1, 1) -> transp(f(Grid_x:Grid_x:(n-2)*Grid_x));
```

Например, для f(x)= exp(x)

```
Output(2:n-1, 1)=(exp(Grid_x:Grid_x:(n-2)*Grid_x))';
```

Далее формируем диагональные и не лежащие на диагонали вспомогательные элементы матрицы A, вектора постоянных B, и решение трехдиагональной системы AX=B.

```
Vd(1 x 1:n) -> s1*ones(1 x n);
Vd(1) -> 1;
Vd(n) -> 1;
Va -> -E(1 x n-1);
```

В данном случае функция E означает формирование массива единиц (аналогично zeros). Формирование данного массива осуществляется функцией ones. Синтаксис данной функции выглядит следующим образом:

```
Y = ones(n)
Y = ones(m, n)
Y = ones(size(A)):
```

Y = ones(n) формирует массив единиц размера n x n.
Y = ones(m, n) формирует массив единиц размера m x n.
Y = ones(size(A)) формирует массив единиц соразмерный с массивом A.

```
Va(n-1) -> 0;
Vc -> -E(1 x n-1);
Vc(1) -> 0;
Vb(1) -> border_value_1;
Vb(n) -> border_value_2;
```

Например,

```
Vd(1, 1:n)=s1*ones(1, n);
Vd(1)=1;
Vd(n)=1;
Va=-ones(1, n-1);
Va(n-1)=0;
Vc=-ones(1,n-1);
Vc(1)=0;
Vb(1)=a;
Vb(n)=b;
```

Далее

```
For j = 2, m;
      For i = 2, n-1;
         Vb(i)->Output(i-1 x j-1)+Output(i+1 x j-1)+s2*Output(i x j-1);
      End;
      N -> length(Vb);
```

Здесь length(x) — функция определения длины массива в скобках.

Например,

```
for j=2:m;
for i=2:n-1;
Vb(i)=Output(i-1, j-1)+Output(i+1, j-1)+s2*Output(i, j-1);
end;

Vd(1, 1:n)=s1*ones(1, n);
Vd(1)=1;
Vd(n)=1;
Va=-ones(1, n-1);
Va(n-1)=0;
Vc=-ones(1,n-1);
Vc(1)=0;
Vb(1)=a;
Vb(n)=b;
```

Проводим расчёт метода прогонки:

```
for k = 2, N
mult -> Va(k-1)/Vd(k-1);
Vd (k) -> Vd (k)-mult*Vc(k-1);
Vb (k) -> Vb (k)-mult* Vb (k-1);
end
X(N) -> Vb (N)/ Vd (N);
for k -> N-1:-1:1
```

При инициализации данного цикла помимо начального и конечного значения итератора указывается так же его шаг. Шаг, при этом, расположен посередине выражения, в данном случае шаг имеет значение -1

```
X(k) -> (Vb(k)-Vc(k)*X(k+1))/Vd(k);
end
Output(1:n,j) -> transp(X);
End
Output -> transp(Output);
```

Например,

```
  N=length(Vb);
for k=2:N;
    mult=Va(k-1)/Vd(k-1);
    Vd(k)=Vd(k)-mult*Vc(k-1);
    Vb(k)=Vb(k)-mult*Vb(k-1);
end;
X(N)=Vb(N)/Vd(N);
for k=N-1:-1:1
    X(k)=(Vb(k)-Vc(k)*X(k+1))/Vd(k);
end;
Output(1:n,j)=(X)';
  end;
  Output=(Output)';
```

Расчёты завершены, после этого выполняется построение графика результатов:

```
endtime=clock;
surf(Output);
```

```
text(0,-10,num2str(etime(endtime,starttime)));
shading interp;
```

Функция Shading предназначена для определения цветов поверхности графика или объекта.

shading flat — каждый сегмент и линия сетки имеют расцветку определяемую по крайней точке сегмента или угловой точке поверхности с наименьшим индексом.

shading faceted — аналогично flat, но линии сетки не раскрашиваются; данный метод применяется по умолчанию.

shading interp — цвет каждого сегмента интерполируется, согласно значениям в крайних точках.

shading(axes_handle,...) — применяется когда необходимо указать конкретный график.

```
xlabel('X');
ylabel('t');
zlabel('U(x,t)');
colorbar;
axis([0 n 0 m 0 max(max(Output))]);
```