Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа 2 по курсу «Численные методы»

Студент: В. С. Нелюбин
Группа: М8О-408Б

Москва, 2024

Лабораторная работа 2

Задача: Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением. Исследовать зависимость погрешности от сеточных параметров.

9.

$$\frac{\partial^2 u}{\partial t^2} + 3\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} - u + \sin x \exp(-t),$$

$$u(0,t) = \exp(-t),$$

$$u(\pi,t) = -\exp(-t),$$

$$u(x,0) = \cos x,$$

$$u_t(x,0) = -\cos x.$$

Аналитическое решение: $U(x,t) = \exp(-t)\cos x$.

1 Исходный код

Программа хранит двухмерную сетку с координатами X T, и вычисляет следующий слой времени основываясь на предыдущем слое. Есть несколько функций расчёта, для каждого из методов. Так же есть вспомогательные функции, как функция решения методом прогона, испольщуемая в неявной схеме. Условия задачи и параметры сетки вынесены как константы и отдельные функции.

```
1  #include <iostream>
2  #include <cmath>
3  #include <vector>
4
5  const double MAX_X = M_PI;
6  const double MAX_T = 1;
7
8  double f(double x, double t)
9  {
10     return sin(x) * exp(-t);
11 }
12
13 double psi1(double x)
14 {
15     return cos(x);
16 }
17
18 double psi2(double x)
19 {
20     return -cos(x);
21 }
22
23 double phi0(double t)
24 {
25     return exp(-t);
26 }
27
28 double phiL(double t)
29 {
30     return -exp(-t);
31 }
32
33 double correct(double x, double t)
34 {
35     return exp(-t) * cos(x);
36 }
37
38 std::vector<double> progon(std::vector<double> ar, std::vector<double> br, std::vector
       <double> cr, std::vector<double> dr)
```

```
39 {
40     // printf("%d %d %d %d\n", ar.size(), br.size(), cr.size(), dr.size());
41     int len = br.size();
42     if (dr.size() != len)
43     {
44         throw std::invalid_argument("bad array dimensions");
45     }
46     if (ar.size() == len)
47     {
48         if (ar[0] != 0)
49         {
50             throw std::invalid_argument("a[0] must be 0");
51         }
52     }
53     else if (ar.size() + 1 == len)
54     {
55         ar.insert(ar.begin(), 0);
56     }
57     else
58     {
59         throw std::invalid_argument("bad array dimensions");
60     }
61     if (cr.size() == len)
62     {
63         if (cr[len - 1] != 0)
64         {
65             throw std::invalid_argument("c[LAST] must be 0");
66         }
67     }
68     else if (cr.size() + 1 == len)
69     {
70         cr.insert(ar.end(), 0);
71     }
72     else
73     {
74         throw std::invalid_argument("bad array dimensions");
75     }
76     std::vector<double> P;
77     std::vector<double> Q;
78     for (int i = 0; i < len; i++)
79     {
80         double help = br[i];
81         if (i > 0)
82         {
83             help += ar[i] * P[i - 1];
84         }
85         P.push_back(-cr[i] / help);
86         double help2 = dr[i];
87         if (i > 0)
```

```
        {
            help2 -= ar[i] * Q[i - 1];
        }
        Q.push_back(help2 / help);
    }

    // for (int i = 0; i < len; i++)
    // {
    // printf("P[%d] = %lf\tQ[%d] = %lf\n", i, P[i], i, Q[i]);
    // }

    std::vector<double> result(len, Q[len - 1]);
    for (int i = len - 2; i >= 0; i--)
    {
        result[i] = P[i] * result[i + 1] + Q[i];
    }
    return result;
}

/// DECISION: i w IS SPACE AND j h IS TIME
class griddy
{
    int w;
    int h;
    double hsh;
    double tau;
    std::vector<double> vals;

    void process_arguments(int i, int j)
    {
        if ((i < 0) || (i >= w))
        {
            throw std::invalid_argument("index i outside of range");
        }
        if ((j < 0) || (j >= h))
        {
            throw std::invalid_argument("index j outside of range");
        }
    }

public:
    griddy(int width, int height)
    {
        w = width;
        h = height;
        hsh = MAX_X / (w - 1);
        tau = MAX_T / (h - 1);
        vals = std::vector<double>(w * h);
    }
```

```
137     griddy(griddy &other)
138     {
139         w = other.w;
140         h = other.h;
141         hsh = other.hsh;
142         tau = other.tau;
143         vals = std::vector<double>(w * h);
144         for (int i = 0; i < w * h; i++)
145         {
146             vals[i] = other.vals[i];
147         }
148     }
149     void annul()
150     {
151         for (int i = 0; i < vals.size(); i++)
152         {
153             vals[i] = 0;
154         }
155     }
156     double krant()
157     {
158         return pow(1 / hsh, 2) * tau;
159     }
160     double f_proc(int i, int j)
161     {
162         return f(i * hsh, j * tau);
163     }
164     double *U_mut(int i, int j)
165     {
166         process_arguments(i, j);
167         int idx = i * h + j;
168         return &vals[idx];
169     }
170     double U(int i, int j)
171     {
172         return *U_mut(i, j);
173     }
174     void print()
175     {
176         // for (int i = 0; i < vals.size(); i++)
177         // {
178         // printf("%3.0lf ", vals[i]);
179         // }
180         // printf("\n");
181 
182         // for (int i = 0; i < w; i++)
183         // {
184         for (int j = 0; j < h; j++)
185         {
```

```
            for (int i = 0; i < w; i++)
            {
                printf("%8.4lf ", U(i, j));
            }
            printf("\n");
        }
    }
    void cheat_set_correct()
    {
        for (int i = 0; i < w; i++)
        {
            for (int j = 0; j < h; j++)
            {
                *U_mut(i, j) = correct(i * hsh, j * tau);
            }
        }
    }
    void set_start()
    {
        for (int i = 0; i < w; i++)
        {
            *U_mut(i, 0) = psi1(i * hsh);
        }
        for (int j = 1; j < h; j++)
        {
            *U_mut(0, j) = phi0(j * tau);
            *U_mut(w - 1, j) = phiL(j * tau);
        }
    }

    void second_level_1o()
    {
        for (int i = 0; i < w; i++)
        {
            *U_mut(i, 1) = U(i, 0) + tau * psi2(i * hsh);
        }
    }

    void second_level_2o()
    {
        for (int i = 0; i < w; i++)
        {
            *U_mut(i, 1) = psi1(i * hsh) + tau * psi2(i * hsh) + psi1(i * hsh) * tau *
                tau / 2;
        }
    }

    void cross(int j)
    {
```

```
234         for (int i = 1; i < w - 2; i++)
235         {
236             double p = (1 + hsh) * U(i + 1, j) - (2 + hsh) * U(i, j) + U(i - 1, j);
237             p /= hsh * hsh;
238             p += U(i, j) + f_proc(i, j);
239             p *= tau * tau;
240             p += (2 + 3 * tau) * U(i, j) - U(i, j - 1);
241             p /= 3 * tau + 1;
242             *U_mut(i, j + 1) = p;
243         }
244     }
245
246     void kris_cross(int j)
247     {
248         for (int i = 1; i < w - 2; i++)
249         {
250             double p = (2 + hsh) * U(i + 1, j) - 4 * U(i, j) + (2 - hsh) * U(i - 1, j);
251             p /= 2 * hsh * hsh;
252             p += U(i, j) + f_proc(i, j);
253             p *= 2 * tau * tau;
254             p -= -4 * U(i, j) + (2 - 3 * tau) * U(i, j - 1);
255             p /= 2 + 3 * tau;
256             *U_mut(i, j + 1) = p;
257         }
258     }
259
260     void big_ol_t(int j)
261     {
262         std::vector<double> as(0);
263         std::vector<double> bs(0);
264         std::vector<double> cs(0);
265         std::vector<double> ds(0);
266         for (int i = 1; i < w - 2; i++)
267         {
268             double sa = -(2 - tau) / (2 * hsh * hsh);
269             double sb = -(-4) / (2 * hsh * hsh);
270             double sc = -(2 + tau) / (2 * hsh * hsh);
271             sb += (2 + 3 * tau) / (2 * tau * tau);
272             double sd = U(i, j) + f_proc(i, j);
273             sd -= (-4 * U(i, j) + (2 - 3 * tau) * U(i, j - 1)) / (2 * tau * tau);
274             as.push_back(sa);
275             bs.push_back(sb);
276             cs.push_back(sc);
277             ds.push_back(sd);
278         }
279         ds[0] -= as[0] * U(0, j + 1);
280         as[0] = 0;
281         int l = cs.size() - 1;
282         ds[l] -= cs[l] * U(w - 1, j + 1);
```

```
283         cs[l] = 0;
284         // for (int i = 0; i < as.size(); i++)
285         // {
286         // for (int j = 0; j < i; j++)
287         // {
288         // printf(" ");
289         // }
290         // printf("%7.3lf a + %7.3lf b + %7.3lf c", as[i], bs[i], cs[i]);
291         // for (int j = i + 1; j < as.size(); j++)
292         // {
293         // printf(" ");
294         // }
295         // printf(" = %7.3lf d\n", ds[i]);
296         // }
297         std::vector<double> res = progon(as, bs, cs, ds);
298         for (int i = 1; i < w - 2; i++)
299         {
300             *U_mut(i, j + 1) = res[i];
301         }
302     }
303
304     double square_error()
305     {
306         double result = 0;
307         for (int i = 0; i < vals.size(); i++)
308         {
309             result += pow(vals[i], 2);
310         }
311         return sqrt(result);
312     }
313     double n_row_error(int j)
314     {
315         double res = 0;
316         for (int i = 0; i < w; i++)
317         {
318             res += pow(U(i, j), 2);
319         }
320         return sqrt(res);
321     }
322     griddy diff(griddy &other)
323     {
324         if ((other.w != w) || (other.h != h))
325         {
326             throw std::invalid_argument("dimensions do not allign");
327         }
328         griddy rez = griddy(w, h);
329         for (int i = 0; i < w; i++)
330         {
331             for (int j = 0; j < h; j++)
```

```
            {
                *rez.U_mut(i, j) = U(i, j) - other.U(i, j);
            }
        }
        return rez;
    }
};

double shmain(int w, int h)
{
    griddy x = griddy(w, h);
    printf("kurant %lf\n", x.krant());
    x.annul();
    x.cheat_set_correct();
    x.set_start();
    x.second_level_2o();
    for (int j = 1; j < h - 1; j++)
    {
        // x.kris_cross(j);
        x.big_ol_t(j);
        // break;
    }
    griddy y = griddy(x);
    y.cheat_set_correct();
    griddy z = x.diff(y);
    // z.print();
    for (int j = 0; j < h; j++)
    {
        printf("%lf\n", z.n_row_error(j));
    }
    return z.n_row_error(h - 1);
}

int main()
{
    int w = 10;
    int h = 7;
    double er = shmain(w, h);
    printf("%lf\n", er);
    return 0;
}
```

2 Результаты

На каждом шаге вычисляется среднеквадратическая значение ошибок в каждой ячейке пространственного уровня.
Для параметроа <10,10> (первое - шаги x, второй число - t)
0.000000
0.000522
0.830780
1.603687
2.360446
2.907696
3.216573
4.182426
6.846790
6.804757
6.804757
Последнее число - ошибка на последнем слое. Она будет приводится далее.
<20,10> - 4.092357
<50,10> - 3.851022
<20,10> - 0.597741
<20,50> - 1.010545
<50,50> - 1.006261