Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа 4 по курсу «Численные методы»

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

Москва, 2024

Лабораторная работа 4

Задача: Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров.

9.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + \sin\ x \sin\ y(\mu\cos\mu t + (a+b)\sin\ \mu t),$$

$$u(0, y, t) = 0,$$

$$u(\frac{\pi}{2}, y, t) = \sin y \sin(\mu t),$$

$$u(x, 0, t) = 0,$$

$$u_y(x, \pi, t) = -\sin x \sin(\mu t),$$

$$u(x, y, 0) = 0.$$

Аналитическое решение: $U(x, y, t) = \sin x \sin y \sin(\mu t)$.

1). $a = 1, b = 1, \mu = 1$.
2). $a = 2, b = 1, \mu = 1$.
3). $a = 1, b = 2, \mu = 1$.
4). $a = 1, b = 1, \mu = 2$.

1 Исходный код

Программа хранит двухмерную сетку с координатами X Y, и постепенно приближает начальное состояние к правильному решению. Цикл заканчиается, если расхождение решений достаточно мало, либо количество шагов превысило 100. Есть несколько функций расчёта, для каждого из методов. Условия задачи и параметры сетки вынесены как константы и отдельные функции по возможности.

```
#include <iostream>
#include <cmath>
#include <vector>
#include <stdexcept>

const double MAX_X = M_PI / 2;
const double MAX_Y = M_PI;
const double MAX_T = 1;

const double A = 1;
const double B = 1;
const double MU = 1;

// y = 0
double phi_1(double x, double t)
{
    return 0;
}

// y = max
double phi_d2(double x, double t)
{
    return -sin(x) * sin(MU * t);
}

// x = 0
double phi_3(double y, double t)
{
    return 0;
}

// x = max
double phi_4(double y, double t)
{
    return sin(y) * sin(MU * t);
}

// t = 0
double psi(double x, double y)
```

```
40 {
41     return 0;
42 }
43
44 double f(double x, double y, double t)
45 {
46     return sin(x) * sin(y) * (MU * cos(MU * t) + (A + B) * sin(MU * t));
47 }
48
49 double correct(double x, double y, double t)
50 {
51     return sin(x) * sin(y) * sin(MU * t);
52 }
53
54 std::vector<double> progon(std::vector<double> ar, std::vector<double> br, std::vector
       <double> cr, std::vector<double> dr)
55 {
56     int len = br.size();
57     if (dr.size() != len)
58     {
59         throw std::invalid_argument("bad array dimensions");
60     }
61     if (ar.size() + 1 == len)
62     {
63         ar.insert(ar.begin(), 0);
64     }
65     else if (ar.size() != len)
66     {
67         throw std::invalid_argument("bad array dimensions");
68     }
69     if (cr.size() + 1 == len)
70     {
71         cr.push_back(0); // Corrected
72     }
73     else if (cr.size() != len)
74     {
75         throw std::invalid_argument("bad array dimensions");
76     }
77     std::vector<double> P(len, 0);
78     std::vector<double> Q(len, 0);
79
80     //
81     P[0] = -cr[0] / br[0];
82     Q[0] = dr[0] / br[0];
83     for (int i = 1; i < len; ++i)
84     {
85         double denom = br[i] + ar[i] * P[i - 1];
86         P[i] = -cr[i] / denom;
87         Q[i] = (dr[i] - ar[i] * Q[i - 1]) / denom;
```

```
88      }
89
90      //
91      std::vector<double> result(len);
92      result[len - 1] = Q[len - 1];
93      for (int i = len - 2; i >= 0; --i)
94      {
95          result[i] = P[i] * result[i + 1] + Q[i];
96      }
97      return result;
98  }
99
100 class griddy
101 {
102     int max_x;
103     int max_y;
104     int max_ti;
105     double step_x;
106     double step_y;
107     double step_ti;
108     std::vector<double> vals;
109
110     void process_arguments(int x, int y, int t) const
111     {
112         if ((x < 0) || (x >= max_x))
113         {
114             throw std::invalid_argument(" x  ");
115         }
116         if ((y < 0) || (y >= max_y))
117         {
118             throw std::invalid_argument(" y  ");
119         }
120         if ((t < 0) || (t > max_ti))
121         {
122             throw std::invalid_argument(" t  ");
123         }
124     }
125     double get_x(int x) const
126     {
127         process_arguments(x, 0, 0);
128         return step_x * x;
129     }
130     double get_y(int y) const
131     {
132         process_arguments(0, y, 0);
133         return step_y * y;
134     }
135     double get_t(int ti) const
136     {
```

```
137         process_arguments(0, 0, ti);
138         return step_ti * ti;
139     }
140 
141     void halfstep_unclear_x(int j, int k)
142     {
143         std::vector<double> as(max_x - 2, 0);
144         std::vector<double> bs(max_x - 2, 0);
145         std::vector<double> cs(max_x - 2, 0);
146         std::vector<double> ds(max_x - 2, 0);
147 
148         for (int i = 1; i < max_x - 1; i++)
149         {
150             double sa = -A / pow(step_x, 2);
151             double sb = (1.0 / step_ti) + (2.0 * A / pow(step_x, 2));
152             double sc = -A / pow(step_x, 2);
153             double ff = f(get_x(i), get_y(j), get_t(k - 1));
154             double sd = (ff / 2) + (U(i, j, k - 1) / step_ti);
155             if (i == 1)
156             {
157                 sd -= U(0, j, k) * sa;
158                 sa = 0;
159             }
160             if (i == (max_x - 2))
161             {
162                 sd -= U(max_x - 1, j, k) * sc;
163                 sc = 0;
164             }
165             as[i - 1] = sa;
166             bs[i - 1] = sb;
167             cs[i - 1] = sc;
168             ds[i - 1] = sd;
169         }
170         std::vector<double> rez = progon(as, bs, cs, ds);
171         for (int i = 1; i < max_x - 1; i++)
172         {
173             U_mut(i, j, k) = rez[i - 1];
174         }
175     }
176 
177     void halfstep_unclear_y(int i, int k)
178     {
179         std::vector<double> as(max_y - 2, 0);
180         std::vector<double> bs(max_y - 2, 0);
181         std::vector<double> cs(max_y - 2, 0);
182         std::vector<double> ds(max_y - 2, 0);
183 
184         for (int j = 1; j < max_y - 1; j++)
185         {
```

```
186             double sa = -B / pow(step_y, 2);
187             double sb = (1.0 / step_ti) + (2.0 * B / pow(step_y, 2));
188             double sc = -B / pow(step_y, 2);
189             double ff = f(get_x(i), get_y(j), get_t(k - 1));
190             double sd = (ff / 2) + (U(i, j, k - 1) / step_ti);
191             if (j == 1)
192             {
193                 sd -= U(i, 0, k) * sa;
194                 sa = 0;
195             }
196             if (j == (max_y - 2))
197             {
198                 sd -= U(i, max_y - 1, k) * sc;
199                 sc = 0;
200             }
201             as[j - 1] = sa;
202             bs[j - 1] = sb;
203             cs[j - 1] = sc;
204             ds[j - 1] = sd;
205         }
206         std::vector<double> rez = progon(as, bs, cs, ds);
207         for (int j = 1; j < max_y - 1; j++)
208         {
209             U_mut(i, j, k) = rez[j - 1];
210         }
211     }
212
213 public:
214     griddy(int x_steps, int y_steps, int t_steps)
215     {
216         max_x = x_steps;
217         max_y = y_steps;
218         max_ti = t_steps + 1;
219         step_x = MAX_X / (max_x - 1);
220         step_y = MAX_Y / (max_y - 1);
221         step_ti = MAX_T / (max_ti - 1);
222         vals = std::vector<double>(max_x * max_y * max_ti);
223     }
224     griddy(const griddy &other)
225     {
226         max_x = other.max_x;
227         max_y = other.max_y;
228         max_ti = other.max_ti;
229         step_x = other.step_x;
230         step_y = other.step_y;
231         step_ti = other.step_ti;
232         vals = std::vector<double>(other.vals.size());
233         for (int i = 0; i < max_x * max_y * max_ti; i++)
234         {
```

```
235             vals[i] = other.vals[i];
236         }
237     }
238     void annul()
239     {
240         for (int i = 0; i < vals.size(); i++)
241         {
242             vals[i] = 0;
243         }
244     }
245     int idx(int i, int j, int k) const
246     {
247         return (k * max_ti + j) * max_y + i;
248         // return (i * max_x + j) * max_y + k;
249     }
250     double &U_mut(int i, int j, int k)
251     {
252         process_arguments(i, j, k);
253         return vals[idx(i, j, k)];
254     }
255     double U(int i, int j, int k) const
256     {
257         process_arguments(i, j, k);
258         return vals[idx(i, j, k)];
259     }
260     void cheat_set_correct()
261     {
262         for (int i = 0; i < max_x; i++)
263         {
264             for (int j = 0; j < max_y; j++)
265             {
266                 for (int k = 0; k < max_ti; k++)
267                 {
268                     U_mut(i, j, k) = correct(get_x(i), get_y(j), get_t(k));
269                 }
270             }
271         }
272     }
273     void set_start()
274     {
275         for (int i = 0; i < max_x; i++)
276         {
277             for (int j = 0; j < max_y; j++)
278             {
279                 U_mut(i, j, 0) = 0; // psi(get_x(i), get_y(j));
280             }
281         }
282         for (int k = 0; k < max_ti; k++)
283         {
```

```
284             for (int i = 0; i < max_x; i++)
285             {
286                 U_mut(i, 0, k) = phi_1(get_x(i), get_t(k));
287             }
288             for (int j = 0; j < max_y; j++)
289             {
290                 U_mut(0, j, k) = phi_3(get_y(j), get_t(k));
291                 U_mut(max_x - 1, j, k) = phi_4(get_y(j), get_t(k));
292             }
293         }
294     }
295     double square_error()
296     {
297         double result = 0;
298         for (int i = 0; i < vals.size(); i++)
299         {
300             result += pow(vals[i], 2);
301         }
302         return sqrt(result);
303     }
304     double n_row_error(int k)
305     {
306         double res = 0;
307         for (int i = 0; i < max_x; i++)
308         {
309             for (int j = 0; j < max_y; j++)
310             {
311                 res += pow(U(i, j, k), 2);
312             }
313         }
314         return sqrt(res);
315     }
316     griddy diff(const griddy &other) const
317     {
318         if ((other.max_x != max_x) || (other.max_y != max_y) || (other.max_ti != max_ti
                ))
319         {
320             throw std::invalid_argument("  ");
321         }
322         griddy rez(max_x, max_y, max_ti);
323         for (int i = 0; i < max_x; i++)
324         {
325             for (int j = 0; j < max_y; j++)
326             {
327                 for (int k = 0; k < max_ti; k++)
328                 {
329                     double diff_value = this->U(i, j, k) - other.U(i, j, k);
330                     *(rez.vals.begin() + (k * max_x * max_y + j * max_x + i)) =
                            diff_value;
```

```
            }
        }
    }
    return rez;
}

void print_layer(int k)
{
    for (int j = 0; j < max_y; j++)
    {
        for (int i = 0; i < max_x; i++)
        {
            printf("%5.2lf ", U(i, j, k));
        }
        printf("\n");
    }
}
void print_all()
{
    for (int k = 0; k < max_ti; k++)
    {
        print_layer(k);
        printf("\n");
    }
    printf("-------------------------\n");
}

void full_halfstep_unclear_x(int k)
{
    for (int j = 0; j < max_y; j++)
    {
        halfstep_unclear_x(j, k);
    }
}

void full_halfstep_unclear_y(int k)
{
    for (int i = 0; i < max_x; i++)
    {
        halfstep_unclear_y(i, k);
    }
}

void collapse_halfstep(int k)
{
    for (int i = 0; i < max_x; i++)
    {
        for (int j = 0; j < max_y; j++)
        {
```

```
380 
381                 U_mut(i, j, k) = U(i, j, k + 1);
382                 U_mut(i, j, k + 1) = 0;
383             }
384         }
385     }
386 };
387 
388 double shmain(int w, int h, int d)
389 {
390     griddy x = griddy(w, h, d);
391     x.annul();
392     x.set_start();
393     for (int k = 1; k < d; k++)
394     {
395         x.full_halfstep_unclear_x(k);
396         x.full_halfstep_unclear_y(k + 1);
397         x.collapse_halfstep(k);
398         x.set_start();
399     }
400     griddy y = griddy(x);
401     y.cheat_set_correct();
402     griddy z = x.diff(y);
403     return z.square_error();
404 }
405 
406 int main()
407 {
408     printf("\033[2J");
409     int w = 10;
410     int h = 10;
411     int d = 10;
412     double er = shmain(w, h, d);
413     printf("error is %lf\n", er);
414     return 0;
415 }
```

2 Результаты

Что-то пошло не так, и компилатор перестал работать. Представлена последняя рабочая версия кода, но результаты получить невозможно.