Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа 1 по курсу «Численные методы»

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

Москва, 2024

Лабораторная работа 1

Задача: Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. 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.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial u}{\partial x},\ a > 0,\ b > 0.$$

$$u_x(0,t) - u(0,t) = -\exp(-at)(\cos(bt) + \sin(bt)),$$

$$u_x(\pi,t) - u(\pi,t) = \exp(-at)(\cos(bt) + \sin(bt)),$$

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

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

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.

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

const double MAX_X = M_PI;
const double MAX_T = 1;
/// in methods its a^2
const double A = 0.1;
const double B = 0.1;
/// unused
const double C = 0;

const double ALPHA = 1;
const double BETA = -1;
const double GAMMA = 1;
const double DELTA = -1;

double initial(double x)
{
    return cos(x);
}

double phi0(double t)
{
    return -exp(-A * t) * (cos(B * t) + sin(B * t));
}

double phiL(double t)
{
    return exp(-A * t) * (cos(B * t) + sin(B * t));
}

// U(x,t)=exp(-at)cos(x+bt)

double correct(double x, double t)
{
    return exp(-A * t) * cos(x + B * t);
}

```

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

```
        }
        P.push_back(-cr[i] / help);
        double help2 = dr[i];
        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;
}

enum border_precision
{
    point_2_order_1,
    point_2_order_2,
    point_3_order_2
};

/// DECISION: i w IS SPACE AND j h IS TIME
class griddy
{
    int w;
    int h;
    border_precision prec;
    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");
```

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

```
186     void print()
187     {
188         // for (int i = 0; i < vals.size(); i++)
189         // {
190         // printf("%3.0lf ", vals[i]);
191         // }
192         // printf("\n");
193 
194         // for (int i = 0; i < w; i++)
195         // {
196         for (int j = 0; j < h; j++)
197         {
198             for (int i = 0; i < w; i++)
199             {
200                 printf("%8.4lf ", U(i, j));
201             }
202             printf("\n");
203         }
204     }
205     void cheat_set_correct()
206     {
207         for (int i = 0; i < w; i++)
208         {
209             for (int j = 1; j < h; j++)
210             {
211                 *U_mut(i, j) = correct(i * hsh, j * tau);
212             }
213         }
214     }
215     void set_start()
216     {
217         annul();
218         for (int i = 0; i < w; i++)
219         {
220             *U_mut(i, 0) = initial(i * hsh);
221             for (int j = 1; j < h; j++)
222             {
223                 *U_mut(i, j) = 0;
224             }
225         }
226     }
227 
228     void clear_2p_1o(int k)
229     {
230         *U_mut(0, k + 1) = -(ALPHA / hsh) * U(1, k + 1) + phi0(k + 1);
231         *U_mut(0, k + 1) /= (BETA - ALPHA / hsh);
232         *U_mut(w - 1, k + 1) = (GAMMA / hsh) / (DELTA + GAMMA / hsh) * U(w - 2, k + 1)
                + phiL(k + 1) / (DELTA + GAMMA / hsh);
233     }
```

```
234
235     void clear_3p_2o(int k)
236     {
237         *U_mut(0, k + 1) = phi0(k + 1) - ALPHA * (4 * U(1, k + 1) - U(2, k + 1)) / (2 *
                hsh);
238         *U_mut(0, k + 1) /= BETA - 3 * ALPHA / (2 * hsh);
239         *U_mut(w - 1, k + 1) = phiL(k + 1) - GAMMA * (U(w - 3, k + 1) - 4 * U(w - 2, k
                + 1)) / (2 * hsh);
240         *U_mut(w - 1, k + 1) /= DELTA + 3 * GAMMA / (2 * hsh);
241     }
242
243     void clear_2p_2o(int k)
244     {
245         double a0 = 0;
246         double b0 = (2 * A) / hsh + hsh / tau - C * hsh - (BETA / ALPHA) * (2 * A - B *
                hsh);
247         double c0 = -2 * A / hsh;
248         double d0 = hsh / tau * U(0, k) - phi0(k + 1) * (2 * A - B * hsh) / ALPHA;
249         *U_mut(0, k + 1) = (d0 - c0 * U(1, k + 1)) / b0;
250
251         double aN = -2 * A / hsh;
252         double bN = (2 * A) / hsh + hsh / tau - C * hsh - (DELTA / GAMMA) * (2 * A - B
                * hsh);
253         double cN = 0;
254         double dN = hsh / tau * U(w - 1, k) - phiL(k + 1) * (2 * A - B * hsh) / GAMMA;
255         *U_mut(w - 1, k + 1) = (dN - cN * U(w - 2, k + 1)) / bN;
256     }
257
258     void clear_scheme(int k)
259     {
260         for (int i = 1; i < w - 1; i++)
261         {
262             *U_mut(i, k + 1) = krant() * (U(i - 1, k) + U(i + 1, k)) + (1 - 2 * krant()
                    ) * U(i, k);
263             *U_mut(i, k + 1) += tau * B / (2 * hsh) * (U(i + 1, k) - U(i - 1, k));
264         }
265         switch (prec)
266         {
267         case point_2_order_1:
268             clear_2p_1o(k);
269             break;
270         case point_2_order_2:
271             clear_2p_2o(k);
272             break;
273         case point_3_order_2:
274             clear_3p_2o(k);
275             break;
276         }
277     }
```

```
278
279     void unclear_2p_1o(std::vector<double> &ar, std::vector<double> &br, std::vector<
            double> &cr, std::vector<double> &dr, int k)
280     {
281         for (int i = 0; i < w; i++)
282         {
283             ar[i] = krant();
284             br[i] = -(1 + 2 * krant());
285             cr[i] = krant();
286             dr[i] = -U(i, k);
287         }
288         ar[0] = 0;
289         br[0] = BETA - ALPHA / hsh;
290         cr[0] = ALPHA / hsh;
291         dr[0] = phi0(k + 1) / (BETA - ALPHA / hsh);
292         int l = w - 1;
293         ar[l] = -GAMMA / hsh;
294         br[l] = DELTA + GAMMA / hsh;
295         cr[l] = 0;
296         dr[l] = phiL(k + 1) / (DELTA + GAMMA / hsh);
297     }
298
299     void unclear_3p_2o(std::vector<double> &ar, std::vector<double> &br, std::vector<
            double> &cr, std::vector<double> &dr, int k)
300     {
301         for (int i = 0; i < w; i++)
302         {
303             ar[i] = krant();
304             br[i] = -(1 + 2 * krant());
305             cr[i] = krant();
306             dr[i] = -U(i, k);
307         }
308
309         ar[0] = 0;
310         br[0] = BETA - 3 * ALPHA / (2 * hsh);
311         cr[0] = -2 * ALPHA / hsh;
312         double e0 = ALPHA / (2 * hsh);
313         dr[0] = phi0(k + 1) / br[0];
314         double k0 = e0 / cr[1];
315         br[0] -= ar[1] * k0;
316         cr[0] -= br[1] * k0;
317         e0 -= cr[1] * k0;
318         if (e0 != 0)
319         {
320             printf("ERRORORORO 0 %lf\n", e0);
321         }
322         dr[0] -= dr[1] * k0;
323
324         ar[w - 1] = 2 * GAMMA / hsh;
```

```
        br[w - 1] = DELTA + 3 * GAMMA / (2 * hsh);
        cr[w - 1] = 0;
        double el = -GAMMA / (2 * hsh);
        dr[w - 1] = phiL(k + 1) / br[w - 1];
        double kl = el / cr[w - 2];
        br[w - 1] -= cr[w - 2] * kl;
        ar[w - 1] -= br[w - 2] * kl;
        el -= ar[w - 2] * kl;
        if (el != 0)
        {
            printf("ERRORORORL %lf\n", e0);
        }
        dr[w - 1] -= dr[w - 2] * kl;
    }

    void unclear_2p_2o(std::vector<double> &ar, std::vector<double> &br, std::vector<
        double> &cr, std::vector<double> &dr, int k)
    {
        for (int i = 0; i < w; i++)
        {
            ar[i] = -(A / pow(hsh, 2) - B / (2 * hsh));
            br[i] = 2 * A / pow(hsh, 2) + 1 / tau - C;
            cr[i] = -(A / pow(hsh, 2) + B / (2 * hsh));
            dr[i] = U(i, k) / tau;
        }
        ar[0] = 0;
        cr[w - 1] = 0;
    }

    void unclear_scheme(int k)
    {
        // this is 2-2
        std::vector<double> ar(w, 0);
        std::vector<double> br(w, 0);
        std::vector<double> cr(w, 0);
        std::vector<double> dr(w, 0);
        std::vector<double> xr(w, 0);

        switch (prec)
        {
        case point_2_order_1:
            unclear_2p_1o(ar, br, cr, dr, k);
            break;
        case point_2_order_2:
            unclear_2p_2o(ar, br, cr, dr, k);
            break;
        case point_3_order_2:
            unclear_3p_2o(ar, br, cr, dr, k);
            break;
```

```
373         }
374 
375         // for (int i = 0; i < w; i++)
376         // {
377         // printf("a=%2.3lf a=%2.3lf a=%2.3lf a=%2.3lf \n", ar[i], br[i], cr[i], dr[i])
                ;
378         // }
379 
380         xr = progon(ar, br, cr, dr);
381         for (int i = 0; i < w; i++)
382         {
383             *U_mut(i, k + 1) = xr[i];
384         }
385     }
386 
387     void krank_nickolson(int k)
388     {
389         double theta = 0.5;
390         std::vector<double> half(w, 0);
391         clear_scheme(k);
392         for (int i = 0; i < w; i++)
393         {
394             half[i] = U(i, k);
395         }
396         unclear_scheme(k);
397         for (int i = 0; i < w; i++)
398         {
399             *U_mut(i, k) = U(i, k) * (1 - theta) + half[i] * theta;
400         }
401     }
402 
403     double square_error()
404     {
405         double result = 0;
406         for (int i = 0; i < vals.size(); i++)
407         {
408             result += pow(vals[i], 2);
409         }
410         return sqrt(result);
411     }
412     double n_row_error(int j)
413     {
414         double res = 0;
415         for (int i = 0; i < w; i++)
416         {
417             res += pow(U(i, j), 2);
418         }
419         return sqrt(res);
420     }
```

```
421     griddy diff(griddy &other)
422     {
423         if ((other.w != w) || (other.h != h))
424         {
425             throw std::invalid_argument("dimensions do not allign");
426         }
427         griddy rez = griddy(w, h);
428         for (int i = 0; i < w; i++)
429         {
430             for (int j = 0; j < h; j++)
431             {
432                 *rez.U_mut(i, j) = U(i, j) - other.U(i, j);
433             }
434         }
435         return rez;
436     }
437 };
438 
439 double shmain(int w, int h)
440 {
441     griddy x = griddy(w, h);
442     x.set_prec(point_3_order_2);
443     x.set_start();
444     for (int j = 0; j < h - 1; j++)
445     {
446         // x.clear_scheme(j);
447         x.unclear_scheme(j);
448         // x.krank_nickolson(j);
449     }
450     griddy y = griddy(x);
451     y.cheat_set_correct();
452     griddy z = x.diff(y);
453     z.print();
454     // for (int j = 0; j < h; j++)
455     // {
456     // printf("%lf\n", z.n_row_error(j));
457     // }
458     return z.n_row_error(h - 1);
459 }
460 
461 int main()
462 {
463     int w = 10;
464     int h = 7;
465     double er = shmain(w, h);
466     printf("%lf\n", er);
467     return 0;
468 }
```

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
2.693208
2.455935
2.282627
2.154431
2.058058
1.984218
1.926437
1.880204
1.842361
1.842361
Последнее число - ошибка на последнем слое. Она будет приводится далее.
<20,10> - 1.961996
<50,10> - 2.643591
<20,20> - 1.963352
<50,20> - 2.661760
<20,50> - 1.965696
<50,50> - 2.671939