

**Московский авиационный институт
(национальный исследовательский университет)**

**Институт №8 «Информационные технологии и прикладная
математика»**

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент: С. П. Сабурова
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

1.1 LU - разложение матриц

1 Постановка задачи

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

Вариант: 18

$$\begin{cases} x_1 - x_2 + x_3 - 2x_4 = -20 \\ -9x_1 - x_2 + x_3 + 8x_4 = 60 \\ -7x_1 + 8x_3 - 6x_4 = -60 \\ 3x_1 - 5x_2 + x_3 - 6x_4 = -44 \end{cases}$$

2 Результаты работы

```
L =
1 0 0 0
-0.333333 1 0 0
0.777778 -0.145833 1 0
-0.111111 0.208333 0.11236 1

U =
-9 -1 1 8
0 -5.33333 1.33333 -3.33333
0 0 7.41667 -12.7083
0 0 0 1.01124

Matrix's determinant: 360
Equations solving:
-4
6.66134e-16
-8
4

Inversed matrix:
6.06667 -5.24444 0 -7.68889
0.166667 1.22222 8.32667e-17 -0.0555556
8.83333 -8.22222 1 -12.9444
5.86667 -4.84444 -1.11022e-16 -7.28889
```

Рис. 1: Вывод программы в консоли

1.2 Метод прогонки

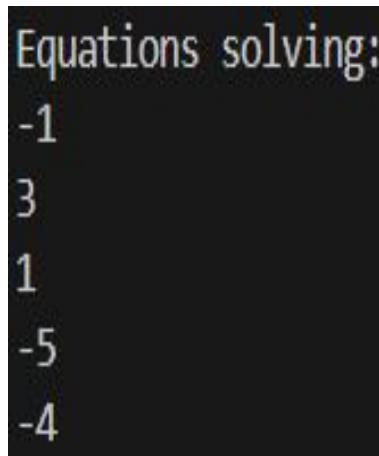
3 Постановка задачи

Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

Вариант: 18

$$\begin{cases} 8x_1 - 2x_2 = -14 \\ 7x_1 - 19x_2 + 9x_3 = -55 \\ -4x_2 + 21x_3 - 8x_4 = 49 \\ 7x_3 - 23x_4 + 9x_5 = 86 \\ 4x_4 - 7x_5 = 8 \end{cases}$$

4 Результаты работы



```
Equations solving:
-1
3
1
-5
-4
```

Рис. 2: Вывод программы в консоли

1.3 Метод простых итераций. Метод Зейделя

5 Постановка задачи

Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

Вариант: 18

$$\begin{cases} 18x_1 + 8x_2 - 3x_3 + 4x_4 = -84 \\ -7x_1 + 15x_2 - 5x_3 - 2x_4 = -5 \\ -4x_1 + 13x_3 + 4x_4 = -38 \\ -8x_1 - 8x_2 - 6x_3 + 31x_4 = 263 \end{cases}$$

6 Результаты работы

```
Simple iterations method:
-4.99626
-4.00237
-5.99874
4.99709
Number of iterations: 14

Seidel's method:
-5.00291
-4.00316
-6.00067
4.99831
Number of iterations: 8
```

Рис. 3: Вывод программы в консоли

1.4 Метод вращений

7 Постановка задачи

Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

Вариант: 18

$$\begin{pmatrix} 2 & 8 & 7 \\ 8 & 2 & 7 \\ 7 & 7 & -8 \end{pmatrix}$$

8 Результаты работы

```
Eigenvalues:
-6
14.3791
-12.3791

Eigenvectors:
0.707388 0.646405 -0.28594
-0.706825 0.64692 -0.286168
0 0.404542 0.914519
```

Рис. 4: Вывод программы в консоли

1.5 QR – разложение матриц

9 Постановка задачи

Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

Вариант: 26

$$\begin{pmatrix} -2 & 7 & -6 \\ -1 & 9 & -4 \\ -1 & 8 & -3 \end{pmatrix}$$

10 Результаты работы

```
QR decomposition:
Q:
-0.816497 0.573068 -0.0701862
-0.408248 -0.659028 -0.631676
-0.408248 -0.487108 0.772049

R:
2.44949 -12.6557 7.75672
0 -5.81664 0.659028
0 8.88178e-16 0.631676

Eigenvalues:
(-1.85022,-0) (4.85411,0) (0.996109,0)
```

Рис. 5: Вывод программы в консоли

11 Исходный код

Реализация матрицы и операции над ней:

```
1  #include <vector>
2  #include <ccomplex>
3  #include <fstream>
4
5  using namespace std;
6
7  struct matrix
8  {
9      int n = 0, m = 0;
10     vector <vector <double>> data;
11
12     matrix() {}
13     matrix(int _n, int _m)
14     {
15         n = _n;
16         m = _m;
17         data = vector<vector<double>>(n, vector <double>(m));
18     }
19
20     vector <double>& operator[] (int row)
21     {
22         return data[row];
23     }
24
25     operator double()
26     {
27         return data[0][0];
28     }
29 };
30
31 //
32 matrix operator*(matrix lhs, matrix rhs)
33 {
34     matrix res(lhs.n, rhs.m);
35     for (int i = 0; i < res.n; i++)
36         for (int j = 0; j < res.m; j++)
37         {
38             res[i][j] = 0;
39             for (int k = 0; k < lhs.m; k++)
40                 res[i][j] += lhs[i][k] * rhs[k][j];
41         }
42     return res;
43 }
44
45 //
46 matrix operator*(double lhs, matrix rhs)
```

```

47 {
48     for (int i = 0; i < rhs.n; i++)
49         for (int j = 0; j < rhs.m; j++)
50             rhs[i][j] *= lhs;
51     return rhs;
52 }
53
54 //
55 matrix operator+(matrix lhs, matrix rhs)
56 {
57     matrix res(lhs.n, lhs.m);
58     for (int i = 0; i < rhs.n; i++)
59         for (int j = 0; j < res.m; j++)
60             res[i][j] = lhs[i][j] + rhs[i][j];
61     return res;
62 }
63
64 //
65 matrix operator-(matrix lhs, matrix rhs)
66 {
67     matrix res(lhs.n, lhs.m);
68     for (int i = 0; i < rhs.n; i++)
69         for (int j = 0; j < res.m; j++)
70             res[i][j] = lhs[i][j] - rhs[i][j];
71     return res;
72 }
73
74 //
75 ostream& operator<<(ostream& stream, matrix _matrix)
76 {
77     for (int i = 0; i < _matrix.n; i++)
78     {
79         for (int j = 0; j < _matrix.m; j++)
80             stream << _matrix[i][j] << " ";
81         stream << endl;
82     }
83     return stream;
84 }
85
86 //
87 istream& operator>>(istream& stream, matrix& _matrix)
88 {
89     for (int i = 0; i < _matrix.n; i++)
90         for (int j = 0; j < _matrix.m; j++)
91             stream >> _matrix[i][j];
92     return stream;
93 }
94
95 //

```



```

96 matrix transposition(matrix _matrix)
97 {
98     matrix res(_matrix.m, _matrix.n);
99     for (int i = 0; i < _matrix.n; i++)
100         for (int j = 0; j < _matrix.m; j++)
101             res[j][i] = _matrix[i][j];
102     return res;
103 }

```

Код выполнения лабораторных работ (все подзадачи):

```

1  #include <iostream>
2  #include <vector>
3  #include <ccomplex>
4  #include <fstream>
5  #include "matrix.cpp"
6
7  using namespace std;
8
9
10 pair<matrix, matrix> lu(matrix u, matrix& roots, bool root_flag)
11 {
12     int n = u.n;
13     matrix l(n, n);
14     for (int k = 0; k < n; k++)
15     {
16         matrix prev = u;
17
18         int swap_index = k;
19         for (int i = k + 1; i < n; i++)
20         {
21             if (abs(prev[swap_index][k]) < abs(prev[i][k]))
22                 swap_index = i;
23         }
24
25         swap(u[k], u[swap_index]);
26         swap(l[k], l[swap_index]);
27         swap(prev[k], prev[swap_index]);
28         if (root_flag)
29             swap(roots[k], roots[swap_index]);
30
31         for (int i = k + 1; i < n; i++)
32         {
33             double h = prev[i][k] / prev[k][k];
34             l[i][k] = h;
35             for (int j = k; j < n; j++)
36                 u[i][j] = prev[i][j] - h*prev[k][j];
37         }
38     }
39     for (int i = 0; i < n; i++)

```

```

40     l[i][i] = 1;
41     return make_pair(l, u);
42 }
43
44 matrix direct_passage(matrix x_matr, matrix y_vect, bool flag)
45 {
46     int n = x_matr.n;
47     matrix res(n, 1);
48     int d, first;
49     if (flag)
50     {
51         first = n-1;
52         d = -1;
53     }
54     else
55     {
56         first = 0;
57         d = 1;
58     }
59     for (int i = first; i < n && i >= 0; i += d)
60     {
61         res[i][0] = y_vect[i][0];
62         for (int j = 0; j < n; j++)
63         {
64             if (i != j)
65                 res[i][0] -= x_matr[i][j] * res[j][0];
66         }
67         res[i][0] = res[i][0] / x_matr[i][i];
68     }
69     return res;
70 }
71
72 matrix solve(pair <matrix, matrix> lu, matrix y_vect)
73 {
74     matrix z = direct_passage(lu.first, y_vect, false);
75     matrix x = direct_passage(lu.second, z, true);
76     return x;
77 }
78
79 matrix inverse(matrix u, matrix roots)
80 {
81     int n = u.n;
82     matrix y_vect(n, 1);
83     pair <matrix, matrix> lu_pair = lu(u, roots, true);
84     matrix res(n, n);
85     for (int i = 0; i < n; i++)
86     {
87         y_vect[max(i - 1, 0)][0] = 0;
88         y_vect[i][0] = 1;

```

```

89     matrix col = solve(lu_pair, y_vect);
90     for (int j = 0; j < n; j++)
91         res[j][i] = col[j][0];
92 }
93 return res;
94 }
95
96 double det(matrix u, matrix roots)
97 {
98     int n = u.n;
99     pair <matrix, matrix> lu_pair = lu(u, roots, false);
100    double res = 1;
101    for (int i = 0; i < n; i++)
102        res *= lu_pair.second[i][i];
103    return res;
104 }
105
106 matrix solve_tridiagonal(matrix& x_matr, matrix& y_vect)
107 {
108     int n = x_matr.n;
109     vector <double> p(n), q(n);
110     p[0] = -x_matr[0][1] / x_matr[0][0];
111     q[0] = y_vect[0][0] / x_matr[0][0];
112     for (int i = 1; i < n; i++)
113     {
114         if (i != n - 1)
115             p[i] = -x_matr[i][i + 1] / (x_matr[i][i] + x_matr[i][i - 1] * p[i - 1]);
116         else
117             p[i] = 0;
118         q[i] = (y_vect[i][0] - x_matr[i][i - 1] * q[i - 1]) / (x_matr[i][i] + x_matr[i][i - 1] * p[i - 1]);
119     }
120     matrix res(n, 1);
121     res[n - 1][0] = q[n - 1];
122     for (int i = n - 2; i >= 0; i--)
123         res[i][0] = p[i] * res[i + 1][0] + q[i];
124     return res;
125 }
126
127 double max_el(matrix x_matr)
128 {
129     double m = 0;
130     for (int i = 0; i < x_matr.n; i++)
131     {
132         double s = 0;
133         for (int j = 0; j < x_matr.m; j++)
134             s += abs(x_matr[i][j]);
135         if (s > m)
136             m = s;

```

```

137     }
138     return m;
139 }
140
141 matrix iterations(matrix x_matr, matrix y_vect, double EPS, int& iters_count)
142 {
143     int n = x_matr.n;
144     matrix x1(n, n), y1(n, 1);
145     for (int i = 0; i < n; i++)
146     {
147         for (int j = 0; j < n; j++)
148             x1[i][j] = -x_matr[i][j] / x_matr[i][i];
149         x1[i][i] = 0;
150     }
151     for (int i = 0; i < n; i++)
152         y1[i][0] = y_vect[i][0] / x_matr[i][i];
153     matrix x = y1;
154     double m = max_el(x_matr);
155     double cur = m, eps = EPS + 1;
156     iters_count = 0;
157     while (eps > EPS)
158     {
159         matrix prev = x;
160         x = y1 + x1 * x;
161         if (m < 1)
162             eps = cur / (1 - m) * max_el(x - prev);
163         else
164             eps = max_el(x - prev);
165         cur = cur * m;
166         iters_count++;
167     }
168     return x;
169 }
170
171 matrix seidel(matrix x_matr, matrix y_vect, double EPS, int& iters_count)
172 {
173     int n = x_matr.n;
174     matrix x1(n, n), y1(n, 1);
175     for (int i = 0; i < n; i++)
176     {
177         for (int j = 0; j < n; j++)
178             x1[i][j] = -x_matr[i][j] / x_matr[i][i];
179         x1[i][i] = 0;
180     }
181     for (int i = 0; i < n; i++)
182         y1[i][0] = y_vect[i][0] / x_matr[i][i];
183     matrix x = y1;
184     double m = abs(x_matr);
185     double cur = m, eps = EPS + 1;

```

```

186     iters_count = 0;
187     while (eps > EPS)
188     {
189         matrix prev = x;
190         for (int i = 0; i < n; i++)
191         {
192             double cur = y1[i][0];
193             for (int j = 0; j < n; j++)
194                 cur += x1[i][j] * x[j][0];
195             x[i][0] = cur;
196         }
197         if (m < 1)
198             eps = cur / (1 - m) * abs(x - prev);
199         else
200             eps = abs(x - prev);
201         cur = cur * m;
202         iters_count++;
203     }
204     return x;
205 }
206
207
208 pair <matrix, matrix> jacobi(matrix a, double EPS)
209 {
210     int n = a.n;
211     double eps = EPS + 1;
212
213     matrix eigenvector(n, n);
214     for (int i = 0; i < n; i++)
215         eigenvector[i][i] = 1;
216
217     while (eps > EPS)
218     {
219         int cur_i = 1, cur_j = 0;
220         for (int i = 0; i < n; i++)
221         {
222             for (int j = 0; j < i; j++)
223                 if (abs(a[cur_i][cur_j]) < abs(a[i][j]))
224                 {
225                     cur_i = i;
226                     cur_j = j;
227                 }
228         }
229         matrix u(n, n);
230         double phi = 3.14 / 4;
231         if (abs(a[cur_i][cur_i] - a[cur_j][cur_j]) > 1e-9)
232             phi = 0.5 * atan((2 * a[cur_i][cur_j]) / (a[cur_i][cur_i] - a[cur_j][cur_j]));
233         for (int i = 0; i < n; i++)

```

```

234         u[i][i] = 1;
235
236         u[cur_j][cur_i] = sin(phi);
237         u[cur_i][cur_j] = -sin(phi);
238         u[cur_i][cur_i] = cos(phi);
239         u[cur_j][cur_j] = cos(phi);
240
241         eigenvector = eigenvector * u;
242         a = transposition(u) * a * u;
243         eps = 0;
244
245         for (int i = 0; i < n; i++)
246             for (int j = 0; j < i; j++)
247                 eps += a[i][j] * a[i][j];
248
249         eps = sqrt(eps);
250     }
251
252     matrix eigenvalue(n, 1);
253     for (int i = 0; i < n; i++)
254         eigenvalue[i][0] = a[i][i];
255
256     return make_pair(eigenvalue, eigenvector);
257 }
258
259 double sign(double x)
260 {
261     return x > 0 ? 1 : -1;
262 }
263
264 pair <matrix, matrix> QR(matrix x_matr)
265 {
266     int n = x_matr.n;
267     matrix E(n, n);
268
269     for (int i = 0; i < n; i++)
270         E[i][i] = 1;
271
272     matrix q = E;
273     for (int i = 0; i < n - 1; i++)
274     {
275         matrix v(n, 1);
276         double s = 0;
277
278         for (int j = i; j < n; j++)
279             s += x_matr[j][i] * x_matr[j][i];
280
281         v[i][0] = x_matr[i][i] + sign(x_matr[i][i]) * sqrt(s);
282

```

```

283     for (int j = i + 1; j < n; j++)
284         v[j][0] = x_matr[j][i];
285
286     matrix h = E - (2.0 / double(transposition(v) * v)) * (v * transposition(v));
287     q = q * h;
288     x_matr = h * x_matr;
289 }
290 return make_pair(q, x_matr);
291 }
292
293 vector<complex<double>> get_eigenvalues(matrix x_matr, double eps)
294 {
295     int n = x_matr.n;
296     vector<complex<double>> prev(n);
297     while (true)
298     {
299         pair <matrix, matrix> p = QR(x_matr);
300
301         x_matr = p.second * p.first;
302
303         vector<complex<double>> current;
304         for (int i = 0; i < n; i++)
305         {
306             if (i < n - 1 && abs(x_matr[i + 1][i]) > 1e-9)
307             {
308                 double b = -(x_matr[i][i] + x_matr[i + 1][i + 1]);
309                 double c = x_matr[i][i] * x_matr[i + 1][i + 1] - x_matr[i][i + 1] *
310                     x_matr[i + 1][i];
311                 double d = b * b - 4 * c;
312
313                 complex<double> isMinus;
314                 if (d > 0)
315                     isMinus = complex<double>(1, 0);
316                 else
317                     isMinus = complex<double>(0, 1);
318
319                 d = sqrt(abs(d));
320                 current.push_back(0.5 * (-b - isMinus * d));
321                 current.push_back(0.5 * (-b + isMinus * d));
322                 i++;
323             }
324             else
325                 current.push_back(x_matr[i][i]);
326         }
327         bool flag = true;
328         for (int i = 0; i < n; i++)
329             flag = flag && abs(current[i] - prev[i]) < eps;
330         if (flag)
331             break;

```

```

331     prev = current;
332 }
333 return prev;
334 }
335
336
337 int main()
338 {
339     cout << "If you want to change input data, open \"res\" directory and and change
        the data in the file of the corresponding task" << endl;
340     cout << "Enter the code of task you want to check:" << endl;
341     cout << "1) LU decomposition" << endl;
342     cout << "2) Tridiagonal algorithm" << endl;
343     cout << "3) Seidel and simple iterations methods" << endl;
344     cout << "4) Method Jacobi" << endl;
345     cout << "5) QR decomposition" << endl << endl;
346     cout << "enter \"exit\" if you want to stop the programm" << endl << endl;
347
348     int command = 8;
349     while (command){
350         cin >> command;
351         if (command == 0)
352         {
353             cout << "Thank you for using this program. Come back again!";
354             return 0;
355         }
356         else if (command == 1)
357         {
358             ifstream file_input("res/input_1.1.txt");
359             int n;
360             file_input >> n;
361             matrix coeffs(n, n), roots(n, 1);
362             file_input >> coeffs >> roots;
363             pair <matrix, matrix> lu_pair = lu(coeffs, roots, true);
364             cout << "L =\n" << lu_pair.first << "\nU =\n" << lu_pair.second << endl;
365             cout << "\nMatrix's determinant: " << det(coeffs, roots);
366             cout << "\nEquations solving:\n" << solve(lu_pair, roots);
367             cout << "\nInversed matrix:\n" << inverse(coeffs, roots) * coeffs;
368             cout << endl << endl;
369         }
370         else if (command == 2)
371         {
372             ifstream file_input("res/input_1.2.txt");
373             int n;
374             file_input >> n;
375
376             matrix coeffs(n, n), roots(n, 1);
377             file_input >> coeffs >> roots;
378

```



```

379         cout << "Equations solving:\n" << solve_tridiagonal(coeffs, roots);
380         cout << endl << endl;
381     }
382     else if (command == 3)
383     {
384         ifstream file_input("res/input_1.3.txt");
385         int n;
386         file_input >> n;
387         matrix coeffs(n, n), roots(n, 1);
388         file_input >> coeffs >> roots;
389
390         int iters_count = 0;
391         cout << "Simple iterations method:\n" << iterations(coeffs, roots, 0.01,
392             iters_count) << "Number of iterations: " << iters_count << endl;
393         cout << "\nSeidel's method:\n" << seidel(coeffs, roots, 0.01, iters_count)
394             << "Number of iterations: " << iters_count << endl << endl << endl;
395     }
396     else if (command == 4)
397     {
398         ifstream file_input("res/input_1.4.txt");
399         int n;
400         file_input >> n;
401         matrix coeffs(n, n);
402         file_input >> coeffs;
403
404         pair<matrix, matrix> p = jacobi(coeffs, 0.01);
405         cout << "Eigenvalues:\n" << p.first << "\nEigenvectors:\n" << p.second <<
406             endl << endl;
407     }
408     else if (command == 5)
409     {
410         ifstream file_input("res/input_1.5.txt");
411         int n;
412         file_input >> n;
413         matrix coeffs(n, n);
414         file_input >> coeffs;
415
416         pair<matrix, matrix> p = QR(coeffs);
417         cout << "QR decomposition:\nQ:\n" << p.first << "\nR:\n" << p.second;
418
419         vector<complex<double>> v = get_eigenvalues(coeffs, 0.01);
420         cout << "\nEigenvalues:\n";
421         for (int i = 0; i < n; i++)
422             cout << v[i] << ' ';
423         cout << endl << endl;
424     } else
425         cout << "Sorry, but this command is invalid" << endl << endl;
426 }

```