Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Численные методы»

Вариант №3: Нахождение собственных значений и собственных векторов симметричных разреженных матриц большой размерности. Метод Ланцоша.

Студент: А. А. Литвина
Преподаватель: И. Э. Иванов
Группа: М8О-306Б
Дата:
Оценка:
Подпись:

Москва, 2020

1 Постановка задачи

Для разреженной симметричной матрицы большой размерности найти собственные значения и собственные векторы методом Ланцоша.

2 Описание

Метод Ланцоша сводит частичную проблему собственных значений симметричной вещественной матрицы к полной проблеме собственных значений для симметричной трехдиагональной матрицы меньшей размерности.

Алгоритм Ланцоша комбинирует метод Ланцоша построения крыловского подпространства и метод Рэлея-Ритца поиска приближённых собственных значений.

Метод Рэлея-Ритца является методом поиска k приближённых собственных значений симметричной вещественной матрицы A размера $n \times n$. Если $Q = [Q_k, Q_u]$ - ортонормированная матрица размера $n \times n$, Q_k имеет размер $n \times k$, Q_u имеет размер $n \times n - k$, то можно записать равенство:

$$T = Q^T A Q = [Q_k, Q_u]^T A [Q_k, Q_u] = \begin{bmatrix} Q_k^T A Q_k & Q_k^T A Q_u \\ Q_u^T A Q_k & Q_u^T A Q_u \end{bmatrix} = \begin{bmatrix} T_k & T_{ku}^T \\ T_{ku} & T_u \end{bmatrix}$$

Метод Рэлея-Ритца заключается в том, что собственные значения матрицы $T_k = Q_k^T A Q_k$ объявляются приближёнными собственными значениями матрицы A.

Метод Ланцоша - это метод построения матрицы Q, при использовании которого матрица T оказывается симметричной трёхдиагональной. Трёхдиагональность T приводит к тому, что матрица T_k является трёхдиагональной матрицей меньшей размерности, а для трёхдиагональных матриц существуют высокоэффективные методы поиска собственных значений.

В теории в методе Ланцоша для вычисления каждого следующего столбца q_{j+1} матрицы Q достаточно знать только q_{j-1} и q_j в силу трёхдиагональности матрицы T. На практике из-за ошибок округления, если не предпринимать специальных мер, набор векторов $q_1, ..., q_k$ перестаёт быть ортогональным. Для борьбы с этим явлением на каждом шаге метода Ланцоша приходится выполнять полную переортогонализацию - повторно запускать процесс ортогонализации Грамма-Шмидта.

Алгоритм Ланцоша

Заполняем начальные значения:
$q_1 = b/||b||$,
$\beta_1 = 0$,
$q_0 = 0$,
где b - прозвольный вектор.

Для всех $j = 1, .., k$:

1. $z = Aq_j$

2. Вычисляем элемент на позиции t_{jj} матрицы T_k:
$\alpha_j = q_j^T z$

3. Два раза проводим полную переортогонализацию Грамма-Шмидта:
$z = z - \sum_{i=1}^{j-1}(z^T q_i)q_i$
$z = z - \sum_{i=1}^{j-1}(z^T q_i)q_i$

4. $z = z - \alpha_j q_j - \beta_j q_{j-1}$

5. Вычисляем элементы на позициях $t_{j,j+1}$ и $t_{j+1,j}$:
$\beta_{j+1} = ||z||$

6. Если $\beta_{j+1} = 0$, то алгоритм завершается

7. $q_{j+1} = z/\beta_{j+1}$

В моей реализации я использовала k=3.

Для входных данных
10
0 5 0 0 7 0 0 3 0 1
5 0 0 4 0 0 0 0 2 0
0 0 3 0 0 0 1 0 0 0
0 4 0 0 6 0 0 0 0 0
7 0 0 6 0 0 0 0 4 0
0 0 0 0 0 9 0 0 0 0
0 0 1 0 0 0 0 5 0 0
3 0 0 0 0 0 5 0 0 0

0 2 0 0 4 0 0 0 0 0
1 0 0 0 0 0 0 0 0 8

3 6 2 7 9 1 5 8 2 4

искомая матрица T_3 выглядит следующим образом:

$$\left\{ \begin{array}{lll} 0.0308785 & 0.412675 & 0 \\ 0.412675 & 25.755 & 675.991 \\ 0 & 675.991 & 0.00694791 \end{array} \right\}$$

Как и ожидалось, она трехдиагональная и симметричная.

При решении полной проблемы собственных значений для симметричных трехдиагональных матриц эффективным является QR-алгоритм, позволяющий находить как вещественные, так и комплексные собственные значения.

В основе QR-алгоритма лежит представление матрицы в виде $T = QR$, где Q - ортогональная матрица ($Q^{-1} = Q^T$), а R - верхняя треугольная. Такое разложение существует для любой квадратной матрицы.

Одним из возможных подходов к построению QR - разложения является использование преобразования Хаусхолдера, позволяющего обратить в нуль группу поддиагональных элементов столбца матрицы. Матрица Хаусхолдера имеет вид:

$$H = E - \frac{2}{v^T v} v v^T.$$

Произведем разложение $T = QR$ $n-1$ раз, где $Q = H_1 H_2 ... H_{n-1}$. Далее произведем перемножение матриц в обратном порядке: $T^{(k+1)} = R^{(k)} Q^{(k)}$.

Для нахождения **собственных векторов** необходимо перемножить все матрицы $Q:\ Q = Q^{(1)} * Q^{(2)} * ... * Q^{(k)}$. Столбцы матрицы Q и будут собственными векторами матрицы T. (Собственные векторы исходной матрицы A - это с.в. матрицы T, дополненные нулями.)

Итерационный процесс продолжается, пока сумма квадратов поддиагольнальных элементов первого столбца достаточно велика, т.е.

$$\Big(\sum_{l=2}^{n} \big(t_{l1}^{(k)}\big)^2\Big)^{1/2} > \varepsilon.$$

Когда критерий окончания нарушается, диагональный элемент $t_{11}^{(k)}$ может быть принят в качестве собственного значения, а элементы $t_{22}^{(k)}$, $t_{23}^{(k)}$, $t_{32}^{(k)}$, $t_{33}^{(k)}$ составляют комплексно-сопряженные пары собственных значений, поэтому остальные с.з. определяются из решения квадратного уравнения $(t_{22}^{(k)} - \lambda^{(k)})(t_{33}^{(k)} - \lambda^{(k)}) = t_{23}^{(k)} t_{32}^{(k)}$.

Чтобы проверить, что найденные собственные значения являются правильными, нужно посчитать след матрицы T, он должен равняться сумме с.з. :

$$trT = \sum \lambda_i.$$

Для нашей матрицы: $0.0308785 + 25.755 + 0.00694791 = 688.995 + 0.0308785 - 663.233 = 25.79282641$ с точностью до ε.

3 Исходный код

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

using namespace std;

const double eps=0.0001;

double norm(vector <double> b) {
  double s=0;
  for (int i=0; i<b.size(); i++) {
    s+=pow(b[i],2);
  }
  return s;
}

int sign(double a) {
  if (a>0) return 1;
  if (a<0) return -1;
  return 0;
}

void print(vector <vector<double>> v) {
  for (int i=0; i<v.size(); i++) {
    for (int j=0; j<v.size(); j++) {
      cout << v[i][j] << "\t";
    }
    cout << endl;
  }
}

double max(vector <vector<double>> A) {
  double max=0;
  for (int k=1; k<A.size(); k++) {
    if (abs(A[k][0])>eps) {
      max+=pow(A[k][0],2);
    }
  }
  return pow(max,0.5);
}

void QR(vector <vector<double>> A) {
  int N=A.size();
  vector <vector <double>> newA (N, vector <double>(N,0));
  vector <vector <double>> H (N, vector <double>(N,0));
  vector <vector <double>> Q (N, vector <double>(N,0));
  vector <vector <double>> newQ (N, vector <double>(N,0));
```

```
48  vector <vector <double>> Qtemp (N, vector <double>(N,0));
49  vector <vector <double>> Qp (N, vector <double>(N,0));
50  vector <vector <double>> R (N, vector <double>(N,0));
51  vector <vector <double>> vvt (N, vector <double>(N,0));
52  vector <double> v (N,0);
53
54  for (int i=0; i<N; i++) {
55    Q[i][i]=1;
56  }
57
58  for (int i=0; i<N; i++) {
59    Qp[i][i]=1;
60  }
61
62  while (max(A)>eps) {
63    for (int i=0; i<N-1; i++) {
64      int j;
65      for (j=0; j<i; j++) {
66        v[j]=0;
67      }
68      double sum=0;
69      for (int k=i; k<N; k++) {
70        sum+=pow(A[k][i],2);
71      }
72
73      sum=pow(sum,0.5);
74      v[j]=A[j][i]+sign(A[j][i])*sum;
75      for (j=i+1; j<N; j++) {
76        v[j]=A[j][i];
77      }
78
79      for (int k=0; k<N; k++) {
80        for (int l=0; l<N; l++) {
81          vvt[k][l]=v[k]*v[l];
82        }
83      }
84
85      double vtv=0;
86      for (int k=0; k<N; k++) {
87        vtv+=v[k]*v[k];
88      }
89
90      for (int k=0; k<N; k++) {
91        for (int l=0; l<N; l++) {
92          if (k==l)
93            H[k][l]=1-2*vvt[k][l]/vtv;
94          else
95            H[k][l]=-2*vvt[k][l]/vtv;
96        }
```

```
97        }
98
99        for (int k=0; k<N; k++) {
100         for (int l=0; l<N; l++) {
101           for (int m=0; m<N; m++) {
102             newQ[k][l]+=Q[k][m]*H[m][l];
103           }
104         }
105       }
106
107       for (int k=0; k<N; k++) {
108         for (int l=0; l<N; l++) {
109           for (int m=0; m<N; m++) {
110             newA[k][l]+=H[k][m]*A[m][l];
111           }
112         }
113       }
114
115       A=newA;
116       Q=newQ;
117
118       newA.assign(N, vector <double>(N,0));
119       newQ.assign(N, vector <double>(N,0));
120     }
121
122     for (int i=0; i<N; i++) {
123       for (int j=0; j<N; j++) {
124         for (int k=0; k<N; k++) {
125           Qtemp[i][j]+=Qp[i][k]*Q[k][j];
126         }
127       }
128     }
129
130     Qp=Qtemp;
131     Qtemp.assign(N, vector <double>(N,0));
132
133     R=A;
134     A.assign(N, vector <double>(N,0));
135     for (int k=0; k<N; k++) {
136       for (int l=0; l<N; l++) {
137         for (int m=0; m<N; m++) {
138           A[k][l]+=R[k][m]*Q[m][l];
139         }
140       }
141     }
142
143     Q.assign(N, vector <double>(N,0));
144     for (int i=0; i<N; i++) {
145       Q[i][i]=1;
```

```
    }
  }

  cout << "lambda1 = " << A[0][0] << endl;
  double d;
  d=pow(A[1][1]+A[2][2],2)-4*(A[1][1]*A[2][2]-A[1][2]*A[2][1]);
  if (d>=0) {
    d=pow(d,0.5);
    cout << "lambda2 = " << (A[1][1]+A[2][2]+d)/2 << endl;
    cout << "lambda3 = " << (A[1][1]+A[2][2]-d)/2 << endl;
  }
  else if (d<0) {
    d=pow(-d,0.5);
    cout << "lambda2 = " << (A[1][1]+A[2][2])/2 << " + " << d/2 << "i\n";
    cout << "lambda3 = " << (A[1][1]+A[2][2])/2 << " - " << d/2 << "i\n";
  }

  for (int i=1; i<=N; i++) {
    cout << "X" << i << "\t\t";
  }
  cout << "\n-------------------------------------------------\n";
  print(Qp);
}

int main() {
  int N;
  int k=3;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <vector <double>> q (k+2, vector <double>(N,0));
  vector <vector <double>> T (k, vector <double>(k,0));
  vector <double> b (N,0);
  vector <double> z (N,0);
  vector <double> temp (N,0);
  double alpha;
  double beta=0;

  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
  }

  for (int i=0; i<N; i++) {
    cin >> b[i];
  }

  for (int i=0; i<N; i++) {
    q[1][i]=b[i]/norm(b);
```

```
195   }
196
197   for (int i=1; i<k+1; i++) {
198     for (int j=0; j<N; j++) {
199       double s=0;
200       for (int k=0; k<N; k++) {
201         s+=A[j][k]*q[i][k];
202       }
203       z[j]=s;
204     }
205
206     alpha=0;
207     for (int j=0; j<N; j++) {
208       alpha+=q[i][j]*z[j];
209     }
210
211     T[i-1][i-1]=alpha;
212     if (i!=1) {
213       T[i-1][i-2]=beta;
214       T[i-2][i-1]=beta;
215     }
216
217     for (int p=0; p<2; p++) { // 2 times
218       for (int j=0; j<N; j++) { // for z
219         for (int k=1; k<i; k++) { // for sum
220           double s=0;
221           for (int m=0; m<N; m++) { // for z*q
222             s+=z[m]*q[k][m];
223           }
224           for (int m=0; m<N; m++) { //for s*q
225             temp[m]+=q[k][m]*s;
226           }
227         }
228         z[j]-=temp[j];
229       }
230     }
231
232     for (int j=0; j<N; j++) {
233       z[j]=z[j]-alpha*q[i][j]-beta*q[i-1][j];
234     }
235
236     beta=norm(z);
237     if (beta==0)
238       break;
239
240     for (int j=0; j<N; j++) {
241       q[i+1][j]=z[j]/beta;
242     }
243   }
```

```
244 ||
245 ||   QR(T);
246 ||
247 ||   for (int i=0; i<N-k; i++) {
248 ||     for (int j=0; j<k; j++){
249 ||       cout << 0 << "\t\t";
250 ||     }
251 ||     cout << endl;
252 ||   }
253 ||   cout << "-------------------------------------------------\n";
254 || }
```

4 Консоль

Входные данные:

Введем симметричную разреженную матрицу размерности 10 и произвольный вектор той же размерности

```
10
0 5 0 0 7 0 0 3 0 1
5 0 0 4 0 0 0 0 2 0
0 0 3 0 0 0 1 0 0 0
0 4 0 0 6 0 0 0 0 0
7 0 0 6 0 0 0 0 4 0
0 0 0 0 0 9 0 0 0 0
0 0 1 0 0 0 0 5 0 0
3 0 0 0 0 0 5 0 0 0
0 2 0 0 4 0 0 0 0 0
1 0 0 0 0 0 0 0 0 8

3 6 2 7 9 1 5 8 2 4
```

Выходные данные:

```
Собственные значения:

lambda1 = 688.995
lambda2 = 0.0308785
lambda3 = -663.233

Собственные векторы:

X1                  X2                  X3
-----------------------------------------------
0.000427555      0.000435745       1
0.713807        -0.700342         -2.16111e-08
0.700342         0.713807         -0.000610473
0                0                  0
0                0                  0
0                0                  0
0                0                  0
0                0                  0
```

```
0                0                0
0                0                0
-----------------------------------------------
```

Входные данные:

```
15
0 0 0 0 2 0 5 0 0 0 7 0 0 0 1
0 4 0 0 0 0 3 0 0 0 0 0 6 0 0
0 0 1 0 0 0 0 0 8 0 0 0 0 0 0
0 0 0 0 0 0 7 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 9 0 0 0 1 0
0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
5 3 0 7 0 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 6 0 0
0 0 8 0 0 0 0 0 2 0 0 0 0 0 0
0 0 0 0 9 0 0 0 0 0 0 7 0 0 0
7 0 0 0 0 0 3 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 7 0 0 0 3 0
0 6 0 0 0 0 0 6 0 0 2 0 1 0 0
0 0 0 0 1 0 0 0 0 0 0 3 0 0 4
1 0 0 0 0 0 0 0 0 0 0 0 0 4 0

4 0 7 1 0 4 6 0 8 0 3 1 0 2 6
```

Выходные данные:

```
Собственные значения:

lambda1 = 856.138
lambda2 = 0.0354674
lambda3 = -829.661

Собственные векторы:

X1                  X2                 X3
-----------------------------------------------
-0.000314459   -0.000319414      1
-0.712634       0.701536        -1.37824e-08
-0.701536      -0.712634        -0.000448229
```

```
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
0                   0                   0
-------------------------------------------------
```

Выводы

Выполнив этот курсовой проект, я познакомилась с одним из методов решения частичной задачи собственных значений симметричной матрицы - методом Ланцоша, который сводит симметричную вещественную матрицу к симметричной матрице меньшей размерности. Я реализовала его в виде программы, а так же повторила и закрепила QR-алгоритм нахождения собственных значений, который был изучен мною ранее.

На моем пути встретилось много трудноностей во время выполнения проекта. Самой главной из них, пожалуй, стала нехватка качественной понятной литературы.

Список литературы

Алгоритм Ланцоша для арифметики с плавающей точкой с полной переортогонализацией
URL: http://algorithms.parallel.ru/ru/Алгоритм_Ланцоша_для_арифметики_с_плавающей_точкой_с_полной_переортогонализацией

Алгоритм Ланцоша вычисления собственных значений симметричной матрицы для точной арифметики (без переортогонализации)
URL: https://algowiki-project.org/ru/Участник:AleksLevin/Алгоритм_Ланцоша_вычисления_собственных_значений_симметричной_матрицы_для_точной_арифметики_(без_переортогонализации)

Методы Крыловского подпространства
James W. Demmel - Вычислительная линейная алгебра. Теория и приложения //Мир. - 2001 (с. 313, § 6.6 Методы Крыловского подпространства)