Белорусский государственный университет
Факультет прикладной математики и информатики

Методы исследования проблемы собственных значений

Отчет по лабораторной работе
студента 2 курса 3 группы
Аквуха Джеймса

Преподаватель:
Будник А. М.

Минск 2015

Постановка задачи

Дана положительно определенная симметрическая матрица:

$$A=\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{nn} \end{pmatrix}$$

Требуется найти характеристический многочлен матрицы, ее собственные значения, а также собственный вектор, соответствующий максимальному собственному значению.

1. Метод Крылова

Метод Крылова основан на свойстве квадратной матрицы обращать в нуль свой характеристический многочлен. Согласно теореме Гамильтона-Кали, всякая квадратная матрица является корнем своего характеристического многочлена и, следовательно, обращает его в нуль. Пусть

$$D(\lambda)=\det(A-\lambda E)=(-1)^n\left(\lambda^n+p_1\lambda^{n-1}+p_2\lambda^{n-2}+\dots+p_n\right)$$

- характеристический многочлен.

Заменим величину λ на A, получим:

$$A^n+p_1A^{n-1}+p_2A^{n-2}+\dots+p_nE=0$$

Возьмем произвольный ненулевой вектор:

$$y^{(0)}=\left[\,y_1^{(0)}\;\;y_2^{(0)}\;\dots\;y_n^{(0)}\right]^T$$

Умножим обе части $A^n+p_1A^{n-1}+p_2A^{n-2}+\dots+p_nE=0$ на $y^{(0)}$:

$$A^n y^{(0)}+p_1A^{n-1}y^{(0)}+p_2A^{n-2}y^{(0)}+\dots+p_ny^{(0)}=0$$

Положим $y^{(k)}=Ay^{(k-1)}\;\;k=\overline{1,n}$, т.е. $y^{(1)}=Ay^{(0)},\ y^{(2)}=Ay^{(1)}=A^2y^{(0)},\ \dots,\ y^{(n)}=Ay^{(n-1)}=A^ny^{(0)}$.
Получим:

$$p_1y^{(n-1)}+p_2y^{(n-2)}+\dots+p_ny^{(0)}=-y^{(n)}$$

Иначе:

$$\begin{cases} y_1^{(n-1)}p_1+y_1^{(n-2)}p_2+\dots+y_1^{(0)}p_n=-y_1^{(n)}; \\ y_2^{(n-1)}p_1+y_2^{(n-2)}p_2+\dots+y_2^{(0)}p_n=-y_2^{(n)}; \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ y_n^{(n-1)}p_1+y_n^{(n-2)}p_2+\dots+y_n^{(0)}p_n=-y_n^{(n)}; \end{cases}$$

Если эта система имеет единственное решение, то ее корни являются коэффициентами характеристического многочлена.

Поскольку $y_{0,}y_{1,}\dots,y_{n-1}$, $y_k=y_0^k$ - базис пространства собственных векторов, непосредственно собственный вектор может быть найден по формуле

$$x=\beta_1y_{n-1}+\beta_2y_{n-2}+\dots+\beta_ny_0$$
$$\beta_1=1,\beta_j=\lambda\beta_{j-1}-p_{j-1},\;\;j=\overline{2,n}$$

2. Метод Данилевского

Сущность метода Данилевского заключается в преобразовании исходной матрицы в подобную ей матрицу Фробениуса.
Это делается домножением исходной матрицы на элементарные матрицы особого вида.
На $k-ом$ шаге метода элементарная матрица принимает вид

$$R_{k+1}=\begin{pmatrix} 1 & \cdots & 0 & 0 & 0\cdots & 0 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 1 & 0 & 0\cdots & 0 & \\ -\frac{a_{k1}}{a_{k,k+1}} & \cdots & -\frac{a_{kk}}{a_{k,k+1}} & \frac{1}{a_{k,k+1}} & -\frac{a_{k,k+2}}{a_{k,k+1}}\cdots & -\frac{a_{kn}}{a_{k,k+1}} & \\ 0 & \cdots & 0 & 0 & 1\cdots & 0 & \\ 0 & \cdots & 0 & 0 & 0\cdots & 1 & \end{pmatrix}$$

$$R_{k+1}^{-1}=\begin{pmatrix} 1 & \cdots & 0 & 0 & 0\cdots & 0 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 1 & 0 & 0\cdots & 0 & \\ a_{k1} & \cdots & a_{kk} & a_{k,k+1} & a_{k,k+2}\cdots & a_{kn} & \\ 0 & \cdots & 0 & 0 & 1\cdots & 0 & \\ 0 & \cdots & 0 & 0 & 0\cdots & 1 & \end{pmatrix}$$

$$\Phi=R_n^{-1}R_{n-1}^{-1}\dots R_2^{-1}A\,R_2\dots R_{n-1}R_n$$

Элементы последней строки матрицы Φ есть коэффициенты характеристического уравнения p_i .
Метод может вырождаться. Если возможно переставить строки и столбцы матрицы так, чтобы избавиться от вырожденности, то следует сделать это, домножив матрицу на соответствующие матрицы переноса строк и столбцов (преобразование подобия). Если такая перестановка невозможна, то это означает, что матрица приведена к диагональному виду, и ее характеристический многочлен равен произведению характеристических многочленов диагональных блоков.

Собственный вектор можно найти по формуле:

$$x=R_1R_2\dots R_n\left(\lambda^{n-1},\lambda^{n-2},\dots,1\right)^T$$

3. Степенной метод

Итерационный процесс степенного метода строится следующим образом:

$$y_{k+1}=A\,y_k,\ \ y_0=(1,1,\dots,1)^T$$

Если максимальное по модулю собственное значение матрицы является действительным числом, то метод сходится и $\lambda_{max}=\frac{y_i^{k+1}}{y_i^k}$, а сам вектор y_k сходится к собственному вектору, соответствующему λ_{max} .

4. Метод Якоби

Метод Якоби используется для решения полной проблемы собственных значений и основан на ортогональном преобразовании подобия исходной матрицы с помощью ортогональной матрицы.
При реализации метода вращений преобразование подобия применяется к исходной матрицы A многократно:

$$A^{(k+1)} = \left(H^{(k)}\right)^{-1} \cdot A^{(k)} \cdot H^{(k)} = \left(H^{(k)}\right)^{T} \cdot A^{(k)} \cdot H^{(k)}, \quad k = 0, 1, 2, \ldots$$

Пусть максимальный недиагональный элемент - $a_{ij}^{(k)}$. Тогда:

$$H^{(k)} = \begin{pmatrix} 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cos\varphi^{(k)} & 0 & \cdots & 0 & -\sin\varphi^{(k)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \sin\varphi^{(k)} & 0 & \cdots & 0 & \cos\varphi^{(k)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

$$\operatorname{tg} 2\varphi^{(k)} = \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}$$

Получаемая в процессе итераций матрица $A^{(k)}$ сходится к $diag(\lambda_1, \ldots, \lambda_n)$.

Результат выполнения

```
Krilov method for eigenvalues:
Initial vector:
    1.00000       1.00000     1.00000     1.00000     1.00000
Given matrix:
    2.6338e-01   -1.3841e-02    3.2883e-02   -3.6496e-02    2.3403e-01
   -1.3841e-02    6.2236e-01    1.7919e-02    9.0351e-02    6.0530e-02
    3.2883e-02    1.7919e-02    4.4166e-01   -1.1968e-01    1.6245e-01
   -3.6496e-02    9.0351e-02   -1.1968e-01    4.1642e-01   -3.7151e-02
    2.3403e-01    6.0530e-02    1.6245e-01   -3.7151e-02    6.6459e-01
Polynom coeffs:
   -1.0000e+00    2.4084e+00   -2.1552e+00    8.8429e-01   -1.6375e-01    1.0704e-02
lambda_max:
8.6247e-01
Eigenvector:
    3.4811e-01    1.5058e-01    3.9822e-01   -1.7288e-01    8.1712e-01
Matrix residual:
    1.9950e-06
    1.9950e-06
    1.9950e-06
    1.9950e-06
    1.9950e-06
Polynom residual:
1.0430e-07
Matrix residual norm:
1.9950e-06

Danilevski method for eigenvalues:
Given matrix:
    2.6338e-01   -1.3841e-02    3.2883e-02   -3.6496e-02    2.3403e-01
   -1.3841e-02    6.2236e-01    1.7919e-02    9.0351e-02    6.0530e-02
    3.2883e-02    1.7919e-02    4.4166e-01   -1.1968e-01    1.6245e-01
   -3.6496e-02    9.0351e-02   -1.1968e-01    4.1642e-01   -3.7151e-02
    2.3403e-01    6.0530e-02    1.6245e-01   -3.7151e-02    6.6459e-01
Frobenius matrix:
    2.4084e+00   -2.1552e+00    8.8429e-01   -1.6375e-01    1.0704e-02
    1.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00
    0.0000e+00    1.0000e+00   -2.2204e-16    0.0000e+00    0.0000e+00
    4.5017e-14   -1.0130e-13    1.0000e+00   -2.3595e-14    2.1085e-15
    0.0000e+00    0.0000e+00    0.0000e+00    1.0000e+00    0.0000e+00
Polynom coeffs:
   -1.0000e+00    2.4084e+00   -2.1552e+00    8.8429e-01   -1.6375e-01    1.0704e-02
lambda_max:
8.6247e-01
Eigenvector:
    3.4822e-01    1.5333e-01    3.9708e-01   -1.7276e-01    8.1714e-01
Matrix residual:
   -1.3824e-04
   -6.6772e-04
    5.2239e-04
    3.2554e-04
   -2.7012e-13
Polynom residual:
1.0430e-07
Matrix residual norm:
6.6772e-04

Power iteration method for max eigenvalue & eigenvector:
Given matrix:
    2.6338e-01   -1.3841e-02    3.2883e-02   -3.6496e-02    2.3403e-01
   -1.3841e-02    6.2236e-01    1.7919e-02    9.0351e-02    6.0530e-02
    3.2883e-02    1.7919e-02    4.4166e-01   -1.1968e-01    1.6245e-01
   -3.6496e-02    9.0351e-02   -1.1968e-01    4.1642e-01   -3.7151e-02
    2.3403e-01    6.0530e-02    1.6245e-01   -3.7151e-02    6.6459e-01
Initial vector:
```

```
    1.0000e+00    1.0000e+00    1.0000e+00    1.0000e+00    1.0000e+00
Precision:
9.2622e-07 (40 iterations)
lambda_max:
8.6247e-01
Eigenvector:
    3.4811e-01    1.5060e-01    3.9821e-01   -1.7287e-01    8.1712e-01
Matrix residual:
    1.6560e-06
   -2.8412e-06
    2.1417e-06
   -2.2491e-06
    3.1947e-06
Matrix residual norm:
3.1947e-06

Jacobi method for eigenvalues:
Given matrix:
    2.6338e-01   -1.3841e-02    3.2883e-02   -3.6496e-02    2.3403e-01
   -1.3841e-02    6.2236e-01    1.7919e-02    9.0351e-02    6.0530e-02
    3.2883e-02    1.7919e-02    4.4166e-01   -1.1968e-01    1.6245e-01
   -3.6496e-02    9.0351e-02   -1.1968e-01    4.1642e-01   -3.7151e-02
    2.3403e-01    6.0530e-02    1.6245e-01   -3.7151e-02    6.6459e-01
Transformed matrix (25 iterations):
    1.4390e-01   -8.8753e-09    1.6941e-21   -3.0374e-14   -8.9615e-12
   -8.8753e-09    6.6274e-01   -2.7371e-08    8.8074e-07    2.7105e-20
   -1.6964e-17   -2.7371e-08    2.8898e-01   -5.5516e-10    2.6285e-07
   -3.0355e-14    8.8074e-07   -5.5516e-10    4.5032e-01    3.6881e-07
   -8.9615e-12   -2.0762e-17    2.6285e-07    3.6881e-07    8.6248e-01
Precision:
8.8074e-07
Eigenvalues:
    1.4390e-01    6.6274e-01    2.8898e-01    4.5032e-01    8.6248e-01
```

Вывод

1. Для нахождения максимального собственного значения степенным методом с погрешностью 10^{-6} потребовалось 40 итераций. Это подтверждает теоретические результаты о медленной сходимости степенного метода. Тем не менее тот факт, что метод является сходящимся позволяет утверждать, что метод сходится за конечное число итераций, при условии что максимальное по модулю собственное значение матрицы является действительным числом.

2. Метод вращений обеспечил нахождение спектра матрицы с погрешностью 10^{-6} за 25 итераций. Это подтверждает теоретические результаты о лучшей сходимости метода вращений, по сравнению со степенным методом. Поскольку метод является сходящимся, он обеспечивает нахождение спектра матрицы с любой наперед заданной точностью. Однако собственный вектора, соответствующие спектру матрицы уже не могут быть найдены с произвольной точностью, поскольку их нахождение требует нахождение произведения всех матриц подобных преобразований, что ведет за собой накопление погрешности.

3. Погрешность собственного вектора, найденного методом Данилевского на 3 порядка превышает погрешность собственного вектора, найденного методом Крылова. Это объясняется различным числом вычислений в методах. При нахождени собственного вектора методом Крылова, необходимо произвести операцию умножения вектора на матрицу n раз, затем решить полученную систему методом Гаусса, и рекурсивным методом найти собственный вектор. Причем погрешность зависит как от точности вычислений, так и от погрешности собственного значения, для которого находится собственный вектор. При нахождени собственного вектора методом Данилевского, необходимо также произвести умножение матрицы и вектора n раз, однако при этом сам начальный вектор $(\lambda^{n-1}, \lambda^{n-2}, \dots, 1)$ имеет большую погрешность, чем в методе крылова, и на каждом этапе производится домножение этого вектора на матрицу подобного преобразования, которая найдена с погрешностью.
Таким образом метод Данилевского обоснованно применим лишь для матриц небольшой размерности.

Приложение А. Листинг программы

```
 /**
  * @author James Akwuh <@jakwuh>
  * @description The purpose of the code is only to get calculation results.
  * Optimization, memory handling, etc. are ignored for simplicity.
  * (However there should be no memory leaks because of using vectors)
  */
<... the previous lab's code ...>
using namespace helpers;
namespace io {

	void print(Vector v) {
		if (DEBUG)cout << "{";
		bool first = true;
		for (auto it : v) {
			if (DEBUG)if (!first) cout << ",";
			cout << (DEBUG ? std::fixed : std::scientific) << setprecision(4) << setw(DEBUG && first ? 13 : 14) << it;
			first = false;
		}
		cout << (DEBUG ? "}" : "") << endl;
	}

	void print(Matrix a) {
		int c = a.size();
		if (DEBUG) cout << "{";
		for (auto it : a) {
			print(it);
			if (DEBUG) cout << (--c > 0 ? "," : "}");
		}
		if (DEBUG) cout << endl;
	}

	void check(Matrix a, Vector x, Vector p, double lambda) {
		p.insert(p.begin(), -1);
		Matrix R1 = a * transpose(x) - transpose(x) * lambda;
		double R2 = 0, cc = 1;
		for (int i = 0; i < p.size(); ++i) {
			R2 += p[p.size() - i - 1] * cc;
			cc *= lambda;
		}

		cout << "Polynom coeffs:" << endl;
		print(p);
		cout << "lambda_max:" << endl;
		cout << lambda << endl;
		cout << "Eigenvector:" << endl;
		print(x);
		cout << "Matrix residual norm:" << endl;
		print(R1);
		cout << "Polynom residual:" << endl;
		cout << R2 << endl;
		cout << "Matrix residual norm:" << endl;
		cout << norm(R1) << endl << endl;
	}

	void check(Matrix a, Vector x, double lambda) {
		Matrix R1 = a * transpose(x) - transpose(x) * lambda;

		cout << "lambda_max:" << endl;
		cout << lambda << endl;
		cout << "Eigenvector:" << endl;
		print(x);
		cout << "Matrix residual:" << endl;
		print(R1);
		cout << "Matrix residual norm:" << endl;
		cout << norm(R1) << endl << endl;
	}
}

using namespace io;
namespace methods {
	Vector gauss(Matrix a) {
		Matrix ac(a);
		Vector xs(a.size());
		for (int i = 0; i < xs.size(); ++i) xs[i] = i;
		for (int i = 0; i < ac.size(); ++i) {
			double max = fabs(a[i][i]), maxI = i, maxJ = i;
			for (int l = i; l < ac.size(); ++l) {
				for (int m = i; m < ac.size(); ++m) {
					if (fabs(ac[l][m]) > max) {
						max = fabs(ac[l][m]);
						maxI = l;
						maxJ = m;
					}
				}
			}
			if (maxI != i) ac = swap_rows(ac, i, maxI);
			if (maxJ != i) ac = swap_cols(ac, i, maxJ);
			swap(xs[i], xs[maxJ]);

			for (int k = i + 1; k < ac.size(); ++k) {
				double c = - ac[k][i] / ac[i][i];
				for (int j = i; j < ac[0].size(); ++j) {
					ac[k][j] += c * ac[i][j];
				}
			}
			for (int j = ac[0].size() - 1; j >= i; --j) {
				ac[i][j] /= ac[i][i];
			}
		}
		Vector v(ac.size());
		for (int i = ac.size() - 1; i >= 0; --i) {
			v[xs[i]] = ac[i][ac[0].size()-1];
			for (int j = i + 1; j < ac.size(); ++j) {
				v[xs[i]] -= v[xs[j]] * ac[i][j];
			}
		}
		return v;
	}

	double power_iteration(Matrix a, double precision, bool doPrint = false) {
```

```
	int n = a.size();
	Matrix x, xk = transpose(Vector(n, 1));
	double eps = precision + 1, lambda, lambdak;
	bool first = true;
	while (eps > precision) {
		x = xk;
		xk = a * x;
		if (first) {
			first = false;
			continue;
		}
		lambda = lambdak;
		lambdak = 0;
		for (int i = 0; i < n; ++i) {
			lambdak += xk[i][0] / x[i][0] / n;
		}
		eps = abs(lambdak - lambda);
	}
	if (doPrint) {
		xk = xk * (1 / sqrt(norm(transpose(xk) * xk)));
		cout << "Power iteration method for max eigenvalue & eigenvector:" << endl;
		cout << "Given matrix:" << endl;
		print(a);
		cout << "Initial vector:" << endl;
		print(Vector(n, 1));
		cout << "Precision:" << endl;
		cout << eps << endl;
		check(a, transpose(xk)[0], lambda);
	}
	return lambdak;
}

void krilov(Matrix a) {
	int n = a.size();
	Matrix Y(n, Vector(n + 1, 0));
	for (int i = 0; i < n; ++i) Y[i][0] = 1;
	for (int i = 1; i <= n; ++i) {
		Matrix T = a * column(Y, i - 1);
		for (int j = 0; j < n; ++j) {
			Y[j][i] = T[j][0];
		}
	}
	Vector x = gauss(Y);
	double lambda = power_iteration(a, EPS);
	Vector q(n, 1);
	for (int i = 1; i < n; ++i) {
		q[i] = lambda * q[i - 1] - x[n - i];
	}
	Matrix X(n, Vector(1, 0));
	for (int i = 0; i < n; ++i) {
		X = X + column(Y, i) * q[n - i - 1];
	}
	X = X * (1 / sqrt(norm(transpose(X) * X)));
	cout << "Krilov method for eigenvalues:" << endl;
	cout << "Given matrix:" << endl;
	print(a);
	std::reverse(x.begin(), x.end());
	check(a, transpose(X)[0], x, lambda);
}

void danilevski(Matrix a) {
	int n = a.size();
	Matrix B, A(a), C = identity_matrix(n);
	for (int i = n - 2; i >= 0; --i) {
		B = identity_matrix(n);
		for (int j = 0; j < n; ++j) {
			if (j != i) {
				B[i][j] = -1 * A[i + 1][j] / A[i + 1][i];
			} else {
				B[i][j] = 1 / A[i + 1][i];
			}
		}
		C = C * B;
		A = inverse(B) * A * B;
	}
	double lambda = power_iteration(a, EPS);
	Matrix Y(n, Vector(1, 1));
	double vl = lambda;
	for (int i = 1; i < n; ++i) {
		Y[n - i - 1][0] = vl;
		vl *= lambda;
	}
	Matrix X = C * Y;
	X = X * (1. / sqrt(norm(transpose(X) * X)));
	cout << "Danilevski method for eigenvalues:" << endl;
	cout << "Given matrix:" << endl;
	print(a);
	cout << "Frobenius matrix:" << endl;
	print(A);
	check(a, transpose(X)[0], A[0], lambda);
}

double jacobi_eigenvalue(Matrix a, double precision) {
	double max = precision + 1;
	int imax, jmax, n = a.size(), k = 0;
	Matrix A(a);
	while (true) {
		k++;
		max = 0;
		imax = jmax = 0;
		for (int i = 0; i < n; ++i) {
			for (int j = i; j < n; ++j) {
				if (i != j && abs(A[i][j]) > max) {
					max = abs(A[i][j]);
					imax = i; jmax = j;
				}
			}
		}
		if (max < precision) break;
		double phi = 1. / 2 * atan(2 * A[imax][jmax] / (A[imax][imax] - A[jmax][jmax]));
		Matrix U = identity_matrix(n);
		U[imax][imax] = cos(phi);
		U[jmax][jmax] = cos(phi);
```

```
                        U[imax][jmax] = -1 * sin(phi);
                        U[jmax][imax] = sin(phi);
                        A = transpose(U) * A * U;
                }
                Vector X(n);
                for (int i = 0; i < n; ++i) {
                        X[i] = A[i][i];
                }
                cout << "Jacobi method for eigenvalues:" << endl;
                cout << "Given matrix:" << endl;
                print(a);
                cout << "Transformed matrix (" << k << " iterations):" << endl;
                print(A);
                cout << "Precision:" << endl;
                cout << max << endl;
                cout << "Eigenvalues:" << endl;
                print(X);
                cout << endl;
        }
}

using namespace methods;
int main(int argc, char *argv[]) {
        try     {
                if (argc == 2) {freopen (argv[1], "w", stdout);}
                Matrix a = {
                        {0.4997, -0.0658, 0.0132, 0.0263, 0.0921},
                        {0.0684, 0.7824, 0.0000, -0.0526, 0.0526},
                        {0.0395, 0.0000, 0.6286, -0.1841, 0.1052},
                        {-0.0789, 0.1657, 0.0000, 0.6181, -0.0263},
                        {0.3288, 0.0000, 0.1184, 0.0132, 0.7364}
                };
                Matrix aq = a * transpose(a);
                krilov(aq);
                danilevski(aq);
                power_iteration(aq, EPS, true);
                jacobi_eigenvalue(aq, EPS);
                if (argc == 2) {fclose(stdout);}
        } catch (const char* error) {
                cout << error;
        }
}
```