Министерство образования и науки РФ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ
УНИВЕРСИТЕТ «МИФИ»

Р. Г. Козин

ПРОГРАММИРОВАНИЕ ЧИСЛЕННЫХ МЕТОДОВ ЛИНЕЙНОЙ АЛГЕБРЫ

Учебно-методическое пособие

Москва 2010

УДК [512.64:004.4]:519.6(07)
ББК 22.143я7+22.19я7+32.973.202-018.я7
К59

Козин Р.Г. **Программирование численных методов линейной алгебры:** Учебно-методическое пособие. – М.: НИЯУ МИФИ, 2010. – 128 с.

Учебное пособие знакомит с численными методами решения задач линейной алгебры. Рассматриваются алгоритмы этих методов и подробно обсуждаются вопросы их программной реализации.

Пособие предназначено для студентов группы К4-331, обучающихся по специальности «Прикладная математика и информатика», для методической поддержки лекций, практических занятий и лабораторного практикума по курсу «Специальные главы математики».

Рекомендовано к изданию редсоветом НИЯУ МИФИ

Рецензент канд. техн. наук, доцент *В. В. Золотарев*

ISBN 978-5-7262-1354-5

Оглавление

Предисловие

В пособии рассматриваются численные методы решения задач линейной алгебры. Подробно обсуждаются алгоритмы этих методов и все вопросы, связанные с их программной реализацией. Книга предназначена для поддержки курса «Специальные главы математики».

В книге приведены методы решения следующих задач линейной алгебры:

- определение нормы вектора и матрицы, меры обусловленности матрицы;
- нахождение решения системы линейных уравнений;
- вычисление определителя матрицы;
- определение обратной матрицы;
- нахождение максимального и минимального собственных чисел матрицы;
- нахождение характеристического многочлена матрицы;
- определение всех собственных чисел и векторов матрицы;
- выделение линейного и квадратичного полиномов из заданного полинома;
- локализация и последующее уточнение корней полинома

Каждый тематический раздел завершается несколькими вопросами и заданиями для самоконтроля. В заключении приведены темы заданий для лабораторного практикума.

В конце пособия приведен список литературы, с помощью которой читатель может расширить свои познания в области численных методов линейной алгебры и других разделов вычислительной математики.

Автор выражает благодарность канд. техн. наук, доценту В. В. Золотареву, любезно прочитавшему рукопись и сделавшему ряд ценных замечаний.

Введение

Методы решения задач линейной алгебры являются основополагающими в вычислительной прикладной математике, поскольку большинство реальных вычислительных задач из других разделов математики, как правило, сводятся к аппроксимирующим их задачам линейной алгебры, либо результаты линейной алгебры непосредственно используются при вычислении конкретных прикладных величин.

Наиболее востребованными задачами линейной алгебры являются следующие задачи.

1. Найти вектор x с координатами x_i, $i = 1 \div n$, являющийся решением системы линейных уравнений

$$Ax = b,$$

где A (A_{ij}, $i,j = 1 \div n$) – исходная матрица; b (b_i, $i = 1 \div n$) – заданный вектор правых частей системы.

2. Для заданной матрицы A вычислить определитель $\det(A)$.

3. Найти матрицу A^{-1}, обратную заданной A, для которой выполняется матричное равенство $A^{-1}A = E$, где E – единичная матрица ($E_{ij} = \delta_{ij}$).

4. Для заданной матрицы A определить собственные числа λ_i и соответствующие им собственные векторы x_i, которые удовлетворяют матричному уравнению $Ax = \lambda x$.

5. Найти корни полинома

$$P_n(x) = \sum_{i=0}^{n} a_i x^{n-i} = a_0 x^n + a_1 x^{n-1} + \ldots + a_n = 0,$$

выделить из полинома линейные и квадратичные множители.

Глава 1. НОРМЫ ВЕКТОРА И МАТРИЦ, МЕРА ОБУСЛОВЛЕННОСТИ МАТРИЦЫ

Введем понятия нормы вектора и матрицы, которые используются при «интегральной» оценке результатов операций, выполняемых над этими распределенными объектами.

1.1. Норма вектора

Норма вектора $\|x\|$ – положительная скалярная величина, вычисляемая через его компоненты x_i, $i = 1 \div n$ и явно или косвенно характеризующая его длину. Как и длина вектора, она должна удовлетворять следующим соотношениям:

$$\|x\| \geq 0, \quad \|cx\| = |c|\|x\|, \quad \|x + y\| \leq \|x\| + \|y\|. \tag{1.1}$$

Приведем три способа определения нормы вектора:

1) кубическая норма

$$\|x\|_{\text{куб}} = \max_i |x_i|; \tag{1.2}$$

2) октаэдрическая норма

$$\|x\|_{\text{окт}} = \sum_i |x_i|; \tag{1.3}$$

3) сферическая норма

$$\|x\|_{\text{сф}} = \sqrt{\sum_i |x_i|^2} = \sqrt{(x, x)}. \tag{1.4}$$

Концы множества векторов, нормы которых удовлетворяют равенствам $\|x\|_{\text{куб}} = 1$, $\|x\|_{\text{окт}} = 1$ и $\|x\|_{\text{сф}} = 1$, нарисуют в *n*-мерном пространстве соответственно поверхности *n*-мерного куба, *n*-мерного октаэдра и *n*-мерной сферы. Это обстоятельство и объясняет их название. Легко проверить, что все указанные нормы удовлетворяют трем соотношения (1.1).

Из рис. 1.1 видно, что между этими нормами для любого вектора *x* выполняются соотношения

$$\|x\|_{\text{окт}} \geq \|x\|_{\text{сф}} \geq \|x\|_{\text{куб}}.$$

Справедливость соотношения легко подтверждается на примере конкретного вектора $x=[1,2,3]$, для которого соответствующие нормы равны $\|x\|_{\text{куб}}=3$, $\|x\|_{\text{окт}}=6$ и $\|x\|_{\text{сф}}=\sqrt{1+4+9}=\sqrt{14}\approx 3,74$.

Замечание. Из равенства норм двух векторов не следует равенство самих векторов, в то время как равные векторы всегда имеют равные нормы.

Приведем алгоритмы вычисления двух норм $\|x\|_{\text{куб}}$ и $\|x\|_{\text{окт}}$ (поскольку они одновременно демонстрируют, как можно найти максимальное значение и сумму любой последовательности чисел).

1.
```
norm = 0
for i = 1, n
{
   buf = |x_i|
   if buf > norm then norm = buf
}
```

2.
```
norm = 0
for i = 1, n
{
   norm = norm + |x_i|
}
```

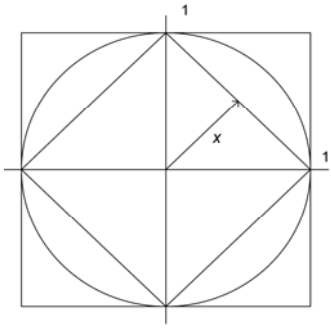


Рис. 1.1. Куб, октаэдр и сфера, соответствующие уравнениям $\|x\|_{\text{куб}}=1$, $\|x\|_{\text{окт}}=1$ и $\|x\|_{\text{сф}}=1$

1.2. Норма матрицы

Норма матрицы $\|A\|$ – положительная скалярная величина, которая устанавливает соответствие между нормами исходного вектора $\|x\|$ и вектора $\|Ax\|$, являющегося результатом воздействия на исходный вектор матрицы. При этом в зависимости от критерия, используемого для нахождения нормы матрицы

$$\|Ax\|\le\|A\|\|x\|, \qquad \|A\|=\sup_x\frac{\|Ax\|}{\|x\|},$$

определяются два вида норм: согласованная и подчиненная (наименьшая из всех согласованных).

Выведем нормы матрицы подчиненные соответственно кубической, октаэдрической и сферической нормам вектора.

1. $$\|Ax\|_{\text{куб}} = \max_i \left|\sum_j A_{ij}x_j\right| \le \max_i \sum_j |A_{ij}||x_j| \le \max_j |x_j| \max_i \sum_j |A_{ij}| =$$
$$= \|x\|_{\text{куб}} \max_i \sum_j |A_{ij}|,$$

отсюда

$$\|A\|_{\text{куб}} = \max_i \sum_j |A_{ij}| \ . \qquad (1.5)$$

2. $$\|Ax\|_{\text{окт}} = \sum_i \left|\sum_j A_{ij}x_j\right| \le \sum_i \sum_j |A_{ij}||x_j| =$$
$$= \sum_j |x_j| \sum_i |A_{ij}| \le \max_j \sum_i |A_{ij}| \sum_j |x_j| =$$
$$= \|x\|_{\text{окт}} \max_j \sum_i |A_{ij}|,$$

отсюда

$$\|A\|_{\text{окт}} = \max_j \sum_i |A_{ij}| \ . \qquad (1.6)$$

3. $\|Ax\|^2_{\text{сф}} = (Ax, Ax) = (x, A^T Ax) = \ldots$

Матрица $A^T A$ симметричная и положительно определенная (так как $(A^T A)^T = A^T (A^T)^T = A^T A$ и $(A^T Ax, x) = (Ax, Ax) > 0$). Поэтому она имеет полную линейно-независимую систему собственных векторов $x^i, i = 1 \div n$, а ее собственные числа действительные и положительные: $\Lambda_1 \ge \Lambda_2 \ge \ldots \ge \Lambda_n > 0$. Представим произвольный вектор x в виде разложения по этим векторам $x = \sum_i c_i x^i$ и подставим его в верхнее скалярное произведение

$$\ldots = (\sum_i c_i x^i, \sum_i c_i A^T Ax^i) = (\sum_i c_i x^i, \sum_i c_i \Lambda_i x^i) \le \Lambda_1 (\sum_i c_i x^i, \sum_i c_i x^i) =$$
$$= \Lambda_1 (x, x) = \Lambda_1 \|x\|^2_{\text{сф}},$$

отсюда

$$\|A\|_{\text{сф}} = \sqrt{\Lambda_1}\ . \qquad (1.7)$$

Замечание. Поскольку для симметричной матрицы $A^T A = A^2$, то $\Lambda_1 = \lambda_1^2$ и $\|A\|_{\text{сф}} = |\lambda_1|$.

В заключение приведем алгоритмы вычисления кубической нормы матрицы (для октаэдрической нормы в этом алгоритме необходимо поменять местами индексы в операторах цикла) и произведения $B = A^T A$, которое используется при определении сферической нормы матрицы

1.
```
norm = 0
for i = 1,n
{
   sum = 0
   for j = 1,n
   {
      sum = sum + |A_ij|
   }
   if sum > norm then norm = sum
}
```

2. $B_{ij} = \sum_k A_{ik}^T A_{kj} = \sum_k A_{ki} A_{kj}$

```
for i = 1,n
{
   for j = 1,n
   {
      B_ij = 0
      for k = 1.n
      {
         B_ij = B_ij + A_ki A_kj
      }
   }
}
```

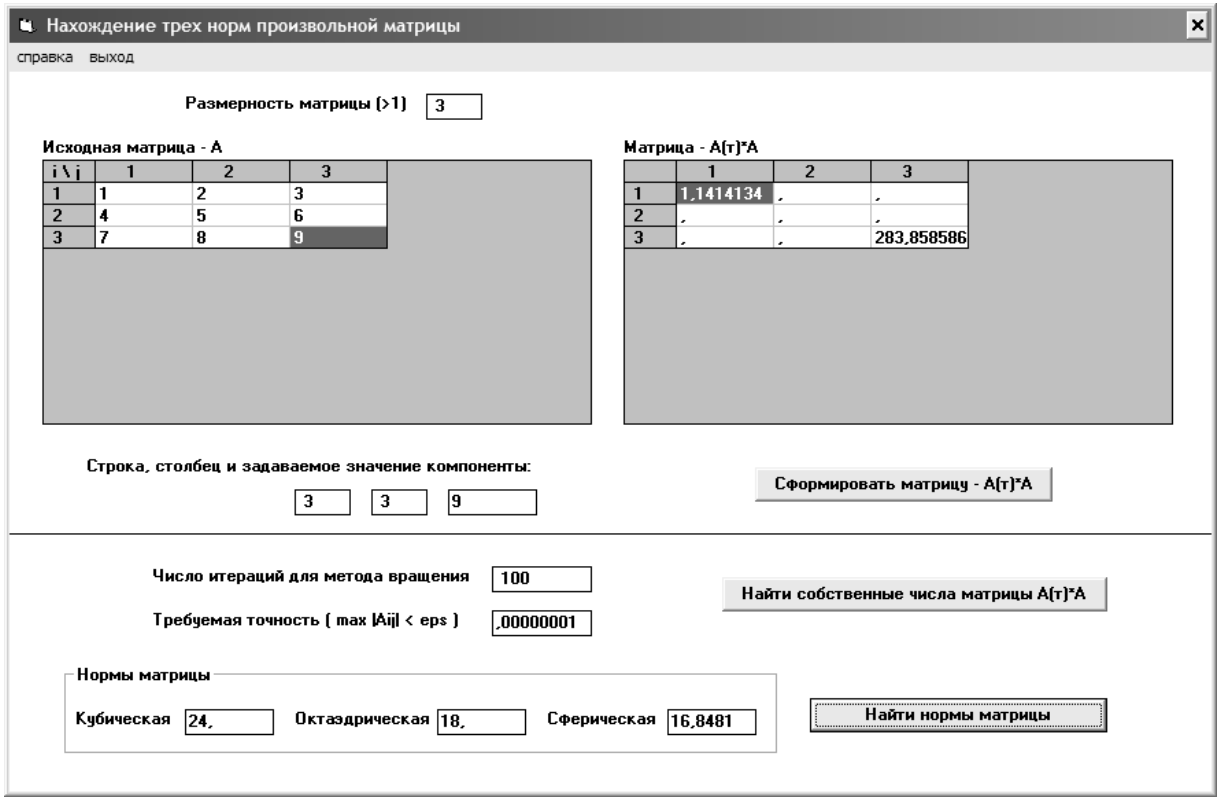


Рис. 1.2. Экранная форма программы, вычисляющей три введенные нормы матрицы

На рис. 1.2. приведен интерфейс программы, позволяющей вычислять три введенные выше нормы матрицы. Для расчета сферической нормы необходимо предварительно умножить матрицу на транспонированную, а затем найти методом вращения все собственные числа полученной таким образом симметричной положительно определенной матрицы. Собственные числа этой матрицы формируются на месте ее диагональных компонент (см. рис. 1.2). Метод вращения описывается в п. 3.6.

1.3. Мера обусловленности матрицы

В зависимости от того, какие соотношения матрица A устанавливает между входными $(\delta A, \delta b)$ и выходными (δx) погрешностями в системе $Ax = b$, матрицы делятся на «плохие» и «хорошие» или в общепринятой терминологии – на плохо и хорошо обусловленные. Качественно это иллюстрируют рис. 1.3 и 1.4, на которых показаны области разброса решений для системы двух уравнений при одинаковых погрешностях в правых частях:

$$F_1(x) \equiv A_{11}x_1 + A_{12}x_2 = b_1;$$
$$F_2(x) \equiv A_{21}x_1 + A_{22}x_2 = b_2.$$

Видно, что чем меньше угол между прямыми $F_1(x) = b_1$ и $F_2(x) = b_1$, тем больше ошибки в решении системы при одних и тех же величинах погрешностей в правых частях, а следовательно, тем хуже матрица.

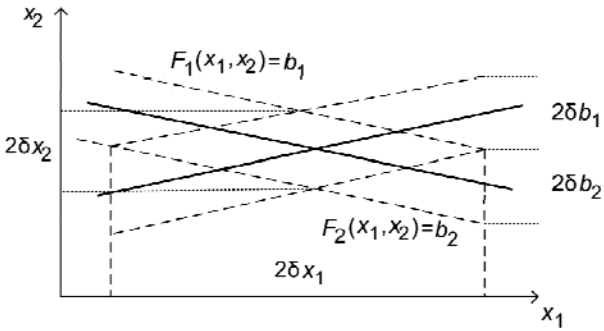


Рис. 1.3. Область разброса плохо обусловленной системы

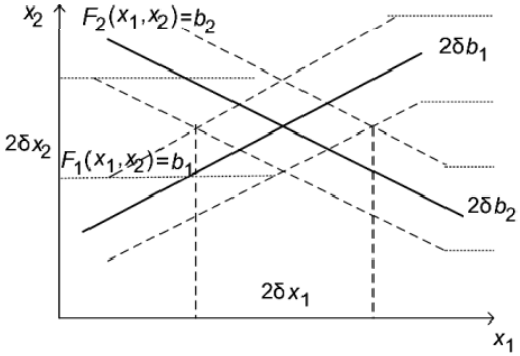


Рис. 1.4. Область разброса хорошо обусловленной системы

Качественно это можно объяснить тем, что с уменьшением угла возрастает линейная зависимость между строками матрицы, и поэтому справедлива следующая цепочка соотношений:

$$\det(A)\to 0\,,\quad A^{-1}\approx\frac{1}{\det(A)}\,,\quad \delta x=A^{-1}\delta b\approx\frac{\delta b}{\det(A)}\to\infty\,.$$

После этих качественных рассуждений перейдем к выводу количественной оценке меры обусловленности матрицы, которая устанавливает количественную зависимость между относительной погрешностью решения $\dfrac{\|\delta x\|}{\|x\|}$ системы линейных уравнений $Ax = b$ и относительными погрешностями исходных параметров системы $\dfrac{\|\delta A\|}{\|A\|}$ и $\dfrac{\|\delta b\|}{\|b\|}$. Для этого запишем два матричных уравнения (исходное и возмущенное погрешностями)

$$Ax=b;$$

$$(A+\delta A)(x+\delta x)=(b+\delta b)\quad\Rightarrow\quad Ax+\delta Ax+A\delta x+\delta A\delta x=b+\delta b.$$

Вычитая первое уравнение из второго и отбрасывая в нем слагаемое второго порядка малости относительно δ, получим

$$A\delta x=\delta b+\delta Ax\quad\text{ или }\quad \delta x=A^{-1}(\delta b+\delta Ax)\,.$$

Отсюда можно построить следующую цепочку неравенств (учтено очевидное неравенство $\|A\|\|x\|\ge\|Ax\|=\|b\|$):

$$\|\delta x\| \le \|A^{-1}\|(\|\delta b\| + \|\delta A\|\|x\|),$$

$$\frac{\|\delta x\|}{\|x\|} \le \|A^{-1}\|(\frac{\|\delta b\|}{\|x\|}\frac{\|b\|}{\|b\|} + \frac{\|\delta A\|\|A\|}{\|A\|}) \le \|A\|\|A^{-1}\|(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}). \quad (1.8)$$

Последнее неравенство устанавливает для системы $Ax = b$ количественное соотношение между входными и выходными относительными погрешностями. Присутствующий в нем коэффициент пропорциональности

$$\nu(A) = \|A\|\|A^{-1}\| \quad (1.9)$$

называется мерой обусловленности матрицы A. Она изменяется в интервале $1 \le \nu(A) \le \infty$.

Замечание. Если учесть очевидное соотношение $(A^T A)^{-1} x = \frac{1}{\Lambda} x$, где x, Λ – собственный вектор и соответствующее собственное число матрицы $(A^T A)$, то, используя определение сферической нормы матрицы (1.7), для меры обусловленности матрицы можно записать выражение, не требующее вычисления обратной матрицы:

$$\nu(A) = \sqrt{\frac{\Lambda_1}{\Lambda_n}}. \quad (1.10)$$

Для симметричной матрицы оно перепишется следующим образом

$$\nu(A) = \left|\frac{\lambda_1}{\lambda_n}\right|,$$

где λ_1, λ_n – максимальное и минимальное собственные числа исходной матрицы A.

Пример. Пусть исходная и возмущенная системы имеют вид

$$\begin{aligned} x_1 - 1{,}001x_2 &= 1, \\ -1{,}001x_1 + x_2 &= 2 \end{aligned} \quad \text{и} \quad \begin{aligned} x_1 - 1{,}0015x_2 &= 1, \\ -1{,}0015x_1 + x_2 &= 2. \end{aligned}$$

Из характеристического уравнения

$$[A - \lambda E] = \begin{bmatrix} 1-\lambda & -1{,}001 \\ -1{,}001 & 1-\lambda \end{bmatrix} \equiv (1-\lambda)^2 + 1{,}002 = 0$$

легко находим собственные числа $\lambda_1 = 2{,}001$ и $\lambda_2 = -0{,}001$ и меру обусловленности исходной симметричной матрицы $\nu(A) = \left|\frac{\lambda_1}{\lambda_2}\right| = 2001.$

Решения исходной и возмущенной систем, соответственно, равны $x_1 = -1500{,}25,\ x_2 = -1499{,}75$ и $x_1 = -1000{,}25,\ x_2 = -999{,}75$.

В итоге легко определяются следующие сферические нормы векторов и матриц

$$\|A\| \approx 2{,}001;\ \|\delta A\| \approx 0{,}0005;\ \|x\| \approx 1500\sqrt{2};\ \|\delta x\| \approx 500\sqrt{2}.$$

После этого неравенство (1.8) записывается в виде

$$\frac{\|\delta x\|}{\|x\|} = 0{,}333 \le \nu(A)\frac{\|\delta A\|}{\|A\|} = 2001 \cdot 0{,}00025 = 0{,}5.$$

Видно, что оно благополучно удовлетворяется.

Вопросы и задания для самоконтроля

1. Подсчитайте три нормы для вектора x = (1, 3, 5, 6) и сравните их между собой.

2. Для какой матрицы A справедлива следующая формула для расчета ее меры обусловленности $\nu(A) = \frac{\lambda_1(A)}{\lambda_n(A)}$?

3. Что характеризует мера обусловленности матрицы?

4. Показать, что кубическая норма вектора удовлетворяет неравенствам (1.1).

5. Найдите двумя способами меру обусловленности матрицы $A = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$.

6. Можно ли утверждать, что из равенства норм следует равенство векторов?

7. Составьте алгоритм нахождения кубической нормы произвольной матрицы.

Глава 2. МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

Все существующие методы решения систем линейных уравнений $x - ?: Ax = b$ делятся на две группы: итерационные и прямые. В первых методах точное решение можно получить после выполнения бесконечного числа операций, для вторых – конечное.

2.1. Итерационные методы решения систем линейных уравнений

2.1.1. Общая схема итерационного процесса и ее исследование

От системы $Ax = b$ перейдем к равносильной системе

$$C\frac{x - x}{\tau} + Ax = b,$$

где C – произвольная невырожденная матрица ($C > 0$), а $\tau > 0$ – числовой параметр. Для последней матрицы можно предложить итерационную схему

$$C\frac{x^{(k+1)} - x^{(k)}}{\tau} + Ax^{(k)} = b$$

или

$$x^{(k+1)} = Bx^{(k)} + b, \qquad (2.1)$$

где $k = 0, 1, 2,...$ – номер итерации, $x^{(0)}$ – начальное приближение, которое задает пользователь,

$$B = E - \tau C^{-1}A, \quad b \equiv \tau C^{-1}b. \qquad (2.2)$$

В дальнейшем, задавая различным образом параметры $C > 0$ $\tau > 0$, мы из схемы (2.1) получим схемы, соответствующие различным, известным итерационным методам. Сейчас же используем ее для решения следующих вопросов, касающихся всех итерационных схем: условие сходимости итерационного процесса, количественная оценка скорости сходимости, выбор критерия окончания итераций и признак расходимости итерационного процесса.

1. Первую проблему решает следующая теорема.

Теорема. Для сходимости процесса (2.1) необходимо и достаточно, чтобы все собственные значения матрицы B были меньше единицы: $|\lambda_i(B)| < 1$.

К сожалению, нахождение собственных значений матрицы – сама по себе не простая задача. Поэтому приведем достаточное условие сходимости, которое не требует вычисления нормы матрицы.

Следствие. Для сходимости итерационного процесса (2.1) достаточно выполнения неравенства $\|B\| < 1$.

Доказательство. Пусть $Bx = \lambda x$. Тогда $\|B\|\|x\| \geq \|Bx\| = \lambda\|x\|$. Отсюда следует неравенство $1 > \|B\| \geq |\lambda|$, которое согласно предыдущей теореме гарантирует сходимость итерационного процесса.

2. Получим оценку скорости сходимости итерационного процесса при условии $\|B\| < 1$.

Если $\|B\| < 1$, то справедливо разложение

$$(E - B)^{-1} = E + B + B^2 + B^3 + \dots .$$

Отсюда можно записать точное решение для задачи $x = Bx + b$ в следующем виде

$$x = (E - B)^{-1} b = (E + B + B^2 + B^3 + \dots)b . \qquad (2.3)$$

С другой стороны, с учетом схемы (2.1) имеем

$$\begin{aligned} x^{(k)} = Bx^{(k-1)} + b = B(Bx^{(k-2)} + b) + b = B^2 x^{(k-2)} + (E + B)b = \dots \\ = B^k x^{(0)} + (E + B + B^2 + B^3 + \dots B^{k-1})b. \end{aligned} \qquad (2.4)$$

С учетом соотношений (2.3), (2.4) можно записать

$$x - x^{(k)} = B^k[(E + B + B^2 + B^3 + \dots)b - x^{(0)}].$$

Отсюда легко выводится оценка скорости сходимости итерационного процесса в зависимости от номера итерации

$$\|x - x^{(k)}\| \leq \|B\|^k [(1 + \|B\| + \|B\|^2 + \|B\|^3 + \dots)\|b\| + \|x^{(0)}\|]$$

или, поскольку $\|B\| < 1$

$$\|x - x^{(k)}\| \leq \|B\|^k [\frac{\|b\|}{(1 - \|B\|}} + \|x^{(0)}\|]. \qquad (2.5)$$

Для частного случая $x^{(0)} = b$ оно преобразуется к виду

$$\left\| x - x^{(k)} \right\| \le \left\| B \right\|^{k+1} \frac{\left\| b \right\|}{(1 - \left\| B \right\|)} \ . \qquad (2.6)$$

3. Критерий окончания итераций.

Метрическое пространство векторов R^n является полным. Поэтому в качестве критерия сходимости последовательности векторов можно использовать критерий Коши: последовательность векторов $\{x^{(k)}\}$ сходится к своему пределу $x \in R^n$ тогда и только тогда, когда для любого $\varepsilon > 0$ существует такое K, что для всех $k > K$ имеет место $\left\| x^{(k+1)} - x^{(k)} \right\| < \varepsilon$.

Наряду с абсолютным критерием целесообразно использовать его относительный аналог

$$\left\| x^{(k+1)} - x^{(k)} \right\| < \varepsilon \left\| x^{(k+1)} \right\| . \qquad (2.7)$$

Кроме того, число итераций итерационного процесса следует ограничивать сверху неким максимальным числом, чтобы исключить возможное зацикливание программы (когда итерационная схема не способна обеспечить требуемую точность) или наступление аварийной ситуации в случае отсутствия сходимости процесса.

4. Итерационный процесс $x^{(k+1)} = Bx^{(k)} + b$ можно считать расходящимся, если неравенство

$$\left\| x^{(k-1)} - x^{(k-2)} \right\| \le \left\| x^{(k)} - x^{(k-1)} \right\|$$

выполняется подряд для трех–пяти последовательных значений k. Справедливость этого утверждения для одномерного случая ($n = 1$) демонстрирует рис. 2.1.

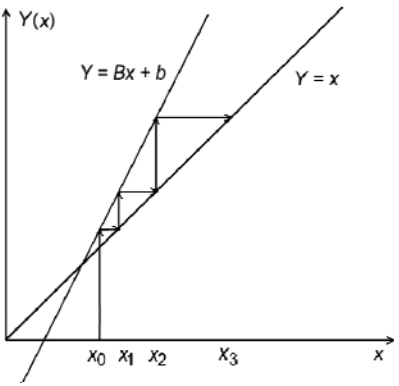


Рис. 2.1. Пример расходящегося итерационного процесса $x^{(k+1)} = Bx^{(k)} + b$ для случая $n = 1$

2.1.2. Частные итерационные схемы

Рассмотрим итерационные схемы, которые вытекают и общей схемы (2.1).

1. **Метод Якоби** получается при следующем выборе параметров: $\tau = 1,\ C = D$. Здесь D – диагональная составляющая исходной матрицы $A = A_L + D + A_H$, где A_L, A_H – соответственно ее нижняя и верхняя составляющие. Отсюда следует

$$B = E - D^{-1}(A_L + D + A_H) = -D^{-1}(A_L + A_H). \qquad (2.8)$$

С учетом очевидного вида обратной диагональной матрицы

$$D^{-1} = \begin{vmatrix} \frac{1}{A_{11}} & 0 & 0 \\ & & \\ 0 & 0 & \frac{1}{A_{nn}} \end{vmatrix}$$

легко выводится покоординатная запись схемы метода Якоби

$$x_i^{(k+1)} = \frac{1}{A_{ii}}(b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}),\ i = 1, n. \qquad (2.9)$$

Достаточное условие сходимости $\|B\| < 1$ для метода имеет следующий вид

$$\|B\|_{\text{куб}} = \max_{\text{i}} \sum_{j \neq i} \left| \frac{A_{ij}}{A_{ii}} \right| < 1$$

или

$$\forall i \quad \sum_{j \neq i} |A_{ij}| < |A_{ii}|. \qquad (2.10)$$

Последнее означает, что для сходимости метода Якоби достаточно преобладания диагональных элементов у исходной матрицы над остальными компонентами.

2. **Метод простой итерации** имеет место при $\tau > 0,\ C = E$. При этом

$$B = E - \tau A \qquad (2.11)$$

покомпонентная запись схемы

$$x_i^{(k+1)} = \tau b_i + \sum_j (\delta_{ij} - \tau A_{ij}) x_j^{(k)},\ i = 1, n, \qquad (2.12)$$

и достаточное условие сходимости метода

$$\|B\|_{\text{куб}} = \max_i \sum_j |\delta_{ij} - \tau A_{ij}| < 1. \qquad (2.13)$$

Если матрица A симметричная и положительно определенная (ее собственные числа $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$), то для нее можно указать оптимальный параметр

$$0 < \tau_{\text{опт}} = \frac{2}{\lambda_1 + \lambda_n}, \qquad (2.14)$$

при котором метод простой итерации всегда сходится, причем с максимальной скоростью, так как в этом случае сферическая норма матрицы B принимает минимально возможное значение

$$(\|B\|_{\text{сф}})_{\text{опт}} = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n} < 1. \qquad (2.15)$$

Покажем это.

Если A – симметричная, то $B = E - \tau A$ также симметричная, и поэтому

$$\|B\|_{\text{сф}} = \max_i |1 - \tau\lambda_i(A)|.$$

Поскольку A – еще и положительно определенная матрица, то ее собственные числа $\lambda_i(A) > 0$, и поэтому все функции $f_i(\tau) = |1 - \tau\lambda_i(A)|$ будут располагаться внутри зоны, очерченной двумя ломаными линиями $|1 - \tau\lambda_1(A)|$ и $|1 - \tau\lambda_n(A)|$ (рис. 2.2). Сферической норме $\|B(\tau)\|_{\text{сф}} = \max_i |1 - \tau\lambda_i(A)|$, как функция параметра τ, на этом рисунке будут соответствовать верхние участки ломаных линий. Видно, что норма принимает минимальное значение при $\tau_{\text{опт}}$, которое задается координатой точки пересечения ломаных линий

$$1 - \tau_{\text{опт}}\lambda_n = \tau_{\text{опт}}\lambda_1 - 1, \ (\|B\|_{\text{сф}})_{\text{опт}} = 1 - \tau_{\text{опт}}\lambda_n.$$

Отсюда легко выводятся выражения (2.14) и (2.15).

Замечание. Если матрица плохо убусловлена, то $\lambda_1 >> \lambda_n$. Тогда согласно выражению (2.15) $(\|B\|_{\text{сф}})_{\text{опт}} \approx 1$ и метод простой итерации для системы будет сходиться плохо.

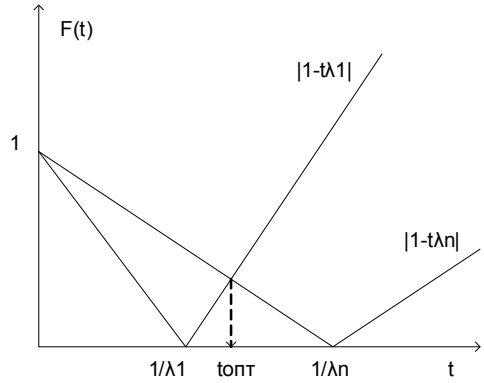


Рис. 2.2. Местоположение функций $f_i(\tau) = \left|1 - \tau\lambda_i(A)\right|$

3. В **методе верхней релаксации** принимается: $\tau > 0$, $C = D + \tau A_L$. В этом случае из уравнения

$$(D + \tau A_L)\frac{x^{(k+1)} - x^{(k)}}{\tau} + Ax^{(k)} = b$$

следует

$$B = E - \tau(D + \tau A_L)^{-1} A, \qquad (2.16)$$

$$Dx^{(k+1)} = \tau b - \tau(A_L x^{(k+1)} + A_H x^{(k)}) + (1-\tau)Dx^{(k)}. \qquad (2.17)$$

Из матричного уравнения (2.17) легко выводится формула для пересчета координат очередной итерации

$$x_i^{(k+1)} = (1-\tau)x^{(k)} + \frac{\tau}{A_{ii}}[b_i - (\sum_{j=1}^{i-1} A_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} A_{ij}x_j^{(k)})], \qquad (2.18)$$

$$i = 1, n.$$

Воспользоваться выражением (2.16) для проверки достаточного условия сходимости метода проблематично, поскольку для этого требуется знание обратной матрицы.

Однако доказано, что если матрица A симметричная и положительно определенная, то метод верхней релаксации всегда сходится при $0 < \tau < 2$.

Частный вариант метода верхней релаксации при $\tau = 1$ известен как метод Зейделя.

В том случае выражения (2.16) и (2.18) преобразуются к виду

$$B = (D + A_L)^{-1} A_H, \tag{2.19}$$

$$x_i^{(k+1)} = \frac{1}{A_{ii}}[b_i - (\sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} A_{ij} x_j^{(k)})], \quad i = 1, n. \tag{2.20}$$

2.1.3. Алгоритмическая реализация методов

Особенности реализации алгоритмов:

1. В качестве нулевого вектора берется вектор правой части матричного уравнения $x^{(0)} = b$.

2. Перед запуском итерационного процесса проводится нормировка каждого уравнения путем деления его на соответствующей диагональный элемент матрицы. При этом, если встречается нулевой диагональный элемент, переменной *code* присваивается единичное значение и осуществляется выход из алгоритма.

3. В алгоритме автоматически отлавливается «не сходимость» итерационного процесса для заданной системы с помощью критерия – увеличение нормы $norm = \left\| x^{(k+1)} - x^{(k)} \right\|_{\text{куб}}$ в течение пяти последовательных итераций. Если последнее имеет место, то переменной *code* присваивается значение двойки и осуществляется выход из алгоритма.

4. Для окончания процесса используются два критерия: число итераций ограничено сверху заданным максимальным числом *K* и *nort* < ε, где *K* и ε задаются пользователем. Окончание процесса согласно одному из этих критериев соответствует *code* = 0. При этом в переменных *K* и ε возвращаются значения использованного числа итераций и достигнутой точности в оценке нормы разности соседних приближений, а в векторе *x* – полученное решение.

5. Используются: буферный вектор $x0_i, i = 1, n$; переменные $norm\_old$, *fl* для хранения соответственно нормы, вычисленной на предыдущей итерации, и числа итераций, при которых последовательно возрастает значение этой нормы.

6. *exit sub* обозначает выход из подпрограммы.

Алгоритм метода Якоби: $x_i^{(k+1)} = \frac{1}{A_{ii}}(b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}),\ i = 1, n.$

Входные параметры процедуры: $A, b, n, x, K, \varepsilon$.

$code = 0, \quad fl = 0, \quad norm\_old = 0$

for $i = 1, n$ – *нормировка уравнений*, *задание* $x0$

{

$x0_i = b_i$

if $|A_{ii}| < 10^{-30} (\approx 0)$ *then* { $code = 1$, *exit* }

$b_i = b_i / A_{ii}$

for $j = 1, n$ { *if* $i \neq j$ *then* $A_{ij} = A_{ij} / A_{ii}$ }

}

for $k = 1, K$ – *цикл по итерациям*

{

$norm = 0$

for $i = 1, n$ – *цикл по уравнениям*

{

$x_i = b_i$

for $j = 1, n$ { *if* $i \neq j$ *then* $x_i = x_i - A_{ij} x0_j$ }

$nt = |x_i - x0_i|$

if $nt > norm$ *then* $norm = nt$

}

if $norm \leq \varepsilon$ *then* { $K = k, \varepsilon = norm$, *exit sub*}

if $norm > norm\_old$ *then* $fl = fl + 1$ *else* $fl = 0$

if $fl > 5$ *then* { $code = 2$, *exit sub*}

$norm\_old = norm$

for $i = 1, n$

{

$x0_i = x_i$

}

}

Алгоритм простой итерации: $x_i^{(k+1)} = \tau b_i + \sum_j (\delta_{ij} - \tau A_{ij}) x_j^{(k)}$, $i = 1, n$ Входные параметры процедуры: $A, b, n, x, \tau, K, \varepsilon$.

$code = 0, \quad fl = 0, \quad norm\_old = 0$
$for\ i = 1, n \quad - \quad нормировка\ уравнений, задание\ x0$
{
$x0_i = b_i$
$b_i = \tau b_i$
$for\ j = 1, n \quad \{ if\ i - j\ then\ A_{ij} = 1 - \tau A_{ij}\ else\ A_{ij} = -\tau A_{ij} \}$
}
$for\ k = 1, K \quad - \quad цикл\ по\ итерациям$
{
$norm = 0$
$for\ i = 1, n \quad - \quad цикл\ по\ уравнениям$
{
$x_i = b_i$
$for\ j = 1, n\ \{ x_i = x_i - A_{ij} x0_j \}$
$nt = |x_i - x0_i|$
$if\ nt > norm\ then\ \ norm = nt$
}
$if\ norm \le \varepsilon\ then\ \ \{ K = k, \varepsilon = norm, exit\ sub \}$
$if\ norm > norm\_old\ then\ fl = fl + 1\ else\ fl = 0$
$if\ fl > 5\ then \quad \{ code = 2,\ exit\ sub \}$
$norm\_old = norm$
$for\ i = 1, n\ \{ x0_i = x_i \}$
}

Алгоритм метода верхней релаксации:

$$x_i^{(k+1)} = (1 - \tau) x_i^{(k)} + \frac{\tau}{A_{ii}} [b_i - (\sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} A_{ij} x_j^{(k)})], \quad i = 1, n.$$

Входные параметры процедуры: $A, b, n, x, \tau, K, \varepsilon$. Используются буферные переменные: $\tau 1, buf, nt$.

$$code = 0, \quad fl = 0, \quad norm\_old = 0, \quad \tau 1 = 1 - \tau$$
$$for\ i = 1, n \quad - \quad нормировка\ уравнений, задание\ x0$$
$$\{$$
$$x_i = b_i$$
$$if\ |A_{ii}| < 10^{-30} (\approx 0)\ then\ \{code = 1,\ exit\}$$
$$b_i = \tau b_i / A_{ii}$$
$$for\ j = 1, n \quad \{if\ i \neq j\ then\ A_{ij} = \tau A_{ij} / A_{ii}\}$$
$$\}$$
$$for\ k = 1, K \quad - \quad цикл\ по\ итерациям$$
$$\{$$
$$norm = 0$$
$$for\ i = 1, n \quad - \quad цикл\ по\ уравнениям$$
$$\{$$
$$buf = \tau 1 x_i + b_i$$
$$for\ j = 1, n\ \{if\ i \neq j\ then\ buf = buf - A_{ij} x_j\}$$
$$nt = |buf - x_i|$$
$$x_i = buf$$
$$if\ nt > norm\ then\ norm = nt$$
$$\}$$
$$if\ norm \leq \varepsilon\ then\ \{K = k, \varepsilon = norm, exit\ sub\}$$
$$if\ norm > norm\_old\ then\ fl = fl + 1\ else\ fl = 0$$
$$if\ fl > 5\ then \quad \{code = 2,\ exit\ sub\}$$
$$norm\_old = norm$$
$$\}$$

Была написана программа, которая позволяет применить любой итерационный метод для решения заданной пользователем системы.

На рис. 2.3 и 2.4 приведены формы этой программы на начальной и конечной стадиях решения системы линейных уравнений методом верхней релаксации. Чтобы матрицу системы сделать симметричной и положительно определенной систему следует умножить на исходную транспонированную матрицу. Если система не является плохо обусловленной, то такое преобразование обеспечит сходимость итерационного процесса. Для запуска выбранного итерационного метода необходимо задать параметр τ, максимальное число итераций и требуемую точность $\varepsilon > \left\| x^k - x^{k-1} \right\|$. По окончанию итерационного процесса в таблице выдается решение и невязка ($\delta = b - Ax^k$).

Замечание. В принципе, в качестве критерия окончания итерационного процесса можно было бы использовать условие $\left\| \delta \right\| = \left\| b - Ax^k \right\| < eps$. Этот критерий является более качественным, но на каждой итерации он потребует значительного объема дополнительных операций. К тому же, как видно из таблицы на рис. 2.4., применяемый критерий обеспечивает выполнение и этого критерия.

С помощью программы было проведено исследование сходимости методов простой итерации и верхней релаксации для модельной системы (см. таблицу на рис. 2.3). Результаты исследования представлены в табл. 2.1 и 2.2.

Для преобразованной системы метод Якоби также сходится. При этом ее решение определяется с заданной точностью за 1895 итераций. Заметим, что в этом случае для матрицы $B = \begin{bmatrix} 0 & 1,4 \\ 0,7 & 0 \end{bmatrix}$ (получена с учетом (2.8)) имеем $\lambda_{1,2} = \pm 0,99 < 1 < \left\| B \right\|_{\text{куб}} = 1,4$, т.е. «необходимое и достаточное условие» выполнено, а «только достаточное» – нет.

Сравнивая все полученные результаты, можно сделать вывод, что для выбранной модельной системы среди всех итерационных методов метод верхней релаксации имеет наиболее высокую абсолютную скорость сходимости. Естественно этот вывод не распространяется на все системы.

Рис. 2.3. Задание исходной системы уравнений

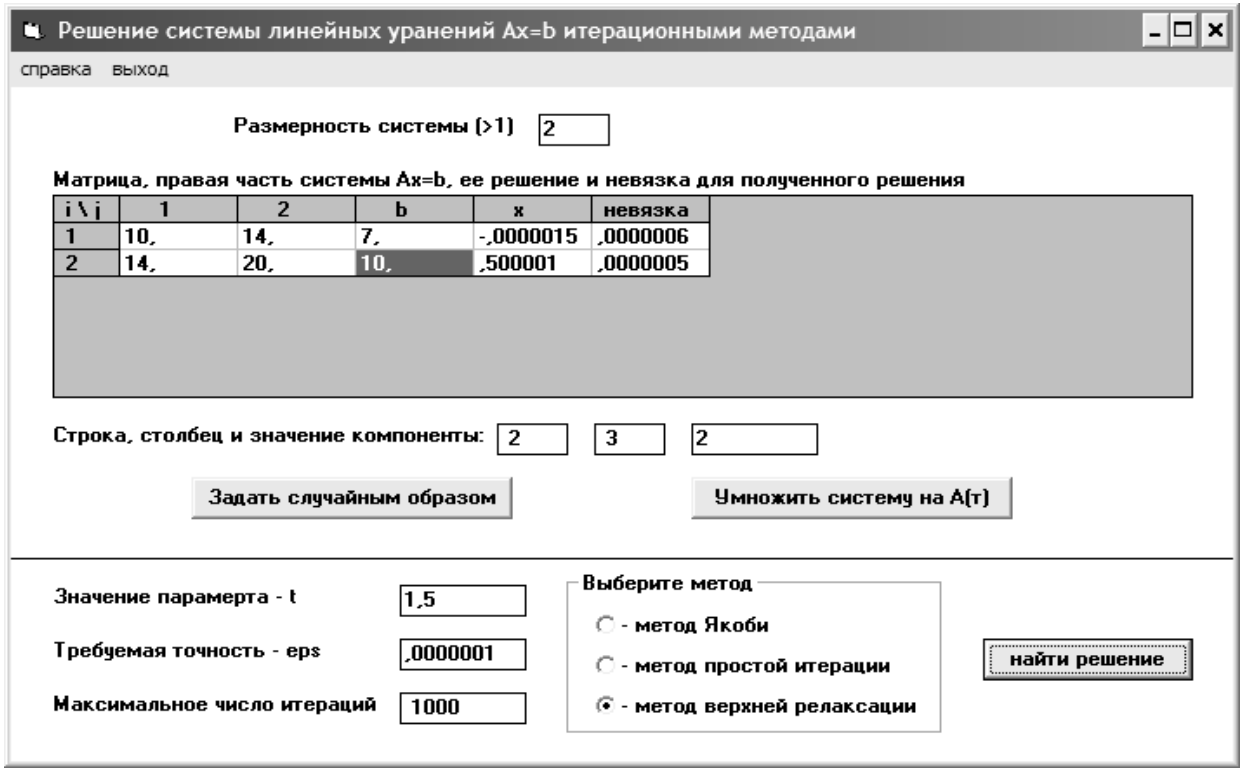


Рис. 2.4. Результаты после окончания итерационного процесса. Система была предварительно умножена на исходную транспонированную матрицу

Таблица 2.1

Зависимость предельного числа итераций от параметра τ (при ε = 0,0000001) для метода верхней релаксации

τ	1	1,25	1,5	1,75	1,8	1,85	1,86	1,87
Итерация	621	397	233	77	75	99	109	расходится

Таблица 2.2

Зависимость предельного числа итераций от параметра τ (при ε = 0,0000001) для метода простой итерации.

τ	0.01	0,02	0,04	0,05	0,06	0,066	0,0665	0,067
итерации	5905	3310	1733	1419	1204	1105	1359	расходится

Примечание: в данном случае для модельной системы имеем: $\lambda_1 = 29{,}866$, $\lambda_2 = 0{,}1339$, $\tau_{опт} = 0{,}066$.

Характер зависимости, зафиксированной в табл. 2.2, хорошо объясняет рис. 2.5.

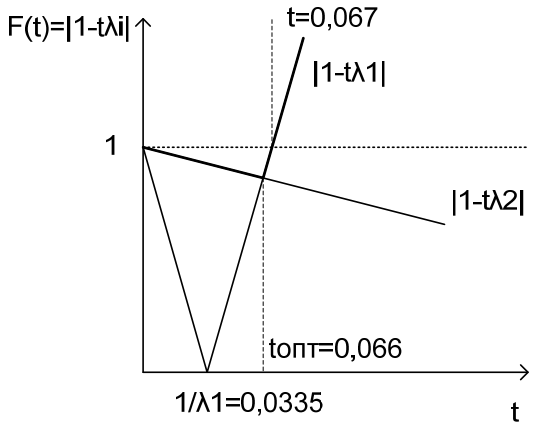


Рис. 2.5. Местоположение функций $f_i(\tau) = \left|1 - \tau\lambda_i(A)\right|$ и $\tau_{опт}$ для метода простой итерации

В качестве второго примера рассмотрим систему

$$\begin{bmatrix} 1 & 0,2 \\ 0,3 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Результаты, полученные с помощью программы для данной системы, приведены в табл. 2.3.

Таблица 2.3

Значения оптимального значения параметра τ и число итераций, использованное для получения решения системы с точностью $\left\| x^k - x^{k-1} \right\| < \varepsilon = 0,0000001$

Метод	$\tau_{опт}$	Число итераций (для ε = 0,0000001)
Якоби	-	12
Простой итерации	0,95	12
Верхней релаксации	1.05	6

Вопросы и задания для самоконтроля

1. Почему для окончания итерационного процесса необходимо использовать два (какие) критерия?
2. Запишите выражение для оценки скорости сходимости итерационного процесса. При каком условии оно получено?
3. Почему последовательное увеличение значений нормы $\left\| x^{(k)} - x^{(k-1)} \right\|$ при трех последовательных значениях k можно считать признаком расходимости итерационного процесса?
4. Чем принципиально отличается программная реализация метода верхней релаксации от остальных методов?
5. Какие две задачи решает предварительная нормировка системы перед запуском итерационного процесса?
6. Почему по окончании итерационного процесса желательно возвращать пользователю число использованных итераций и значение нормы разности двух соседних приближений?
7. Составьте алгоритм решения системы $Ax = b$ методом Зейделя.

2.2. Прямые методы решения системы линейных уравнений

2.2.1. Метод исключения Гаусса

В методе Гаусса исходная система $Ax = b$, начиная с левого столбца, путем $(n–1)$-го последовательного преобразования приводится к системе с верхней треугольной матрицей, Решение последней находится элементарно.

Приведем формулы, по которым выполняется k-е преобразование. К данному моменту текущая расширенная (к матрице добавлен столбец, состоящий из компонент вектора b) матрица исходной системы имеет следующей вид

$$A^{(k-1)} = \begin{bmatrix} A_{11} & - & A_{1k} & - & A_{1j} & - & A_{1n+1} & \\ 0 & - & - & - & - & - & - & \\ 0 & 0 & A_{kk} & - & A_{kj} & - & A_{kn+1} & \ldots k \\ 0 & 0 & - & - & - & - & - & \\ 0 & 0 & A_{ik} & - & A_{ij} & - & A_{in+1} & \ldots i \\ 0 & 0 & - & - & - & - & - & \\ 0 & 0 & A_{nk} & - & - & - & A_{nn+1} & \end{bmatrix} \quad (2.21)$$

При k-ом преобразовании в матрице $A^{(k-1)}$ необходимо обнулить все компоненты k-го нижнего подстолбца, начиная с компоненты A_{k+1k}. Последнее осуществляется путем вычитания из i-й строки компонент k-й строки, умноженной на $\frac{A_{ik}}{A_{kk}}$:

$$A_{ij} = A_{ij} - A_{ik}\frac{A_{kj}}{A_{kk}},$$
$$i = (k+1)\ldots n,$$
$$j = k\ldots(n+1).$$

Алгоритм этой операции представлен на схеме

$$
\begin{array}{l}
for\ i = (k+1), n \\
\{ \\
\quad buf = \dfrac{A_{ik}}{A_{kk}} \\
\quad for\ j = k, (n+1) \\
\quad \{ \\
\qquad A_{ij} = A_{ij} - buf * A_{kj} \\
\quad \} \\
\}
\end{array}
\qquad (2.22)
$$

Замечания.

1. Легко проверить, что преобразование (2.22) эквивалентно матричной операции $M^{(k)}A^{(k-1)}$, где матрица $M^{(k)}$ имеет вид

$$
M^{(k)} = \begin{bmatrix}
1 & 0 & - & - & 0 \\
0 & - & - & - & 0 \\
0 & - & 1 & - & 0 \\
0 & - & M^{(k)}_{(k+1)k} & - & 0 \\
0 & - & - & - & 0 \\
0 & - & M^{(k)}_{nk} & - & 1
\end{bmatrix},
$$

где $M^{(k)}_{ik} = -\dfrac{A^{(k-1)}_{ik}}{A^{(k-1)}_{kk}}$.

Аналогично, операцию приведения матрицы к верхней треугольной можно представить в следующем матричном виде

$$A^{(n-1)} = M^{(n-1)} \ldots M^{(1)} A.$$

2. Поскольку нулевая нижняя подматрица в дальнейших операциях не используется, то нет необходимости ее явно обнулять. Исходя из этого в алгоритме (2.22) внутренний цикл можно начинать с $k + 1$.

Поместив последовательность операций (2.22) внутрь цикла по полному числу преобразований

$$for\ k = 1, (n-1)\ \{ (2.22) \}, \qquad (2.23)$$

получим алгоритм, приводящий исходную систему уравнений к эквивалентной системе с верхней треугольной матрицей.

Для уменьшения влияния ошибок округления и повышения устойчивости работы алгоритма (2.23) в нем перед началом

очередного преобразования необходимо выполнять процедуру «выбора ведущего элемента».

Задачи этой процедуры:

- поставить на место *k*-й строки строку из оставшихся нижних с максимальным по модулю элементом в *k*-ом нижнем подстолбце;

- проверить, что найденный максимальный элемент не равен нулю (матрица невырожденная);

- осуществить подсчет определителя исходной матрицы.

При вычислении определителя учитывается, что:

1) определитель треугольной матрицы равен произведению ее диагональных компонент;

2) значение определителя не изменяется, если к его строке прибавить другую строку, умноженную на произвольный коэффициент

$$\begin{vmatrix} A_{11} & A_{12} \\ A_{21}+\lambda A_{11} & A_{22}+\lambda A_{12} \end{vmatrix} = A_{11}(A_{22}+\lambda A_{12}) - A_{12}(A_{21}+\lambda A_{11}) =$$

$$= A_{11}A_{22} + \lambda A_{11}A_{12} - A_{12}A_{21} - \lambda A_{11}A_{12} = A_{11}A_{22} - A_{12}A_{21};$$

3) при перестановке строк (или столбцов) матрицы знак определителя меняется на противоположный

$$\begin{vmatrix} A_{21} & A_{22} \\ A_{11} & A_{12} \end{vmatrix} = A_{21}A_{12} - A_{11}A_{22} = (-1)(A_{11}A_{22} - A_{21}A_{12});$$

4) умножение всех компонент строки/столбца матрицы на число приводит к умножению определителя этой матрицы на то же число

$$\begin{vmatrix} \lambda A_{11} & \lambda A_{12} \\ A_{21} & A_{22} \end{vmatrix} = \lambda A_{11}A_{22} - \lambda A_{12}A_{21} = \lambda(A_{11}A_{22} - A_{12}A_{21}).$$

Алгоритм процедуры выбора ведущего элемента приведен на схеме (2.24). Здесь *nul* – переменная для хранения машинного нуля, используемого для проверки действительных чисел на нуль (обычно $nul \le 10^{-10}$ и может корректироваться пользователем для фильтрации уровня ошибок округления, возникающих в процессе вычисления).

$\det = 1$ (*начальная инициализация перед запуском цикла* (2.23))
– – процедура – –
$k\max = k$
$A\max = |A_{kk}|$
if $k < n$ *then*
{
  for $i = (k+1), n$
  {
    $A\mathrm{mod} = |A_{ik}|$
    if $A\mathrm{mod} > A\max$ *then*
    {
      $A\max = A\mathrm{mod}$
      $k\max = i$
    }
  }
}
if $|A\max| < nul\ (\approx 0)$ *then*
{
  $\det = 0$ (*матрица вырожденная*)
  выход из процедуры
}
if $k\max \neq k$ *then*
{
  for $j = k, (n+1)$ (*переставляем строки*)
  {
    $buf = A_{kj}$
    $A_{kj} = A_{(k\max)\,j}$
    $A_{(k\max)\,j} = buf$
  }
  $\det = -\det$
}
$\det = \det^* A_{kk}$ (*вычисление определителя*)

(2.24)

Замечание. Перестановка i и j строк эквивалентна матричной операции

$$T(i, j)A,$$

где

$$T(i,j)=\begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{matrix} & & i & & j & \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}\begin{array}{l} \\ \\ i \\ \\ j \\ \\ \end{array}$$

– матрица перестановок (на примере матрицы 6×6 и $i=3,\ j=5$).

Она обладает следующими свойствами:

1. Матричная операция $T(i,j)A$ переставляет строки, например:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}=\begin{bmatrix} A_{21} & A_{22} \\ A_{11} & A_{12} \end{bmatrix}.$$

2. Матричная операция $AT(i,j)$ переставляет столбцы, например,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}=\begin{bmatrix} A_{12} & A_{11} \\ A_{22} & A_{21} \end{bmatrix}.$$

3. $T(i,j)T(i,j)=E$, например,

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

После приведения системы уравнений к системе с треугольной матрицей

$$\begin{array}{lllllll} A_{11}x_1+ & \ldots & +A_{1i}x_i+ & \ldots & +A_{1j}x_j+ & \ldots & =A_{1(n+1)} \\ & & \ldots & \ldots & \ldots & \ldots & \ldots \\ & & A_{ii}x_i+ & \ldots & +A_{ij}x_j+ & \ldots & =A_{i(n+1)} \\ & & & & \ldots & \ldots & \ldots \\ & & & & & A_{nn}x_n & =A_{n(n+1)} \end{array} \qquad (2.25)$$

легко записать алгоритм решения последней (учтено, что правые части уравнений хранятся в дополнительном столбце расширенной матрицы $A_{i(n+1)}$):

$$
\begin{array}{l}
if\ \left|A_{nn}\right| < nul\ then\ \{\det = 0, exit\ sub\}\ else\ \det = \det^{*} A_{nn} \\
x_n = \dfrac{A_{n(n+1)}}{A_{nn}} \\
for\ i = (n-1),1\ step = -1 \\
\{ \\
\quad x_i = \dfrac{1}{A_{ii}}(A_{i(n+1)} - \sum_{j=(i+1)}^{n} A_{ij} x_j) \\
\}
\end{array}
\qquad (2.26)
$$

Здесь первый условный оператор завершает проверку матрицы и подсчет значения определителя.

2.2.2. Метод оптимального исключения Гаусса–Жордана

В отличие от метода Гаусса в данном методе исходная система $Ax = b$ с помощью n последовательных преобразований сводится к системе с единичной матрицей, решение которой располагается на месте правого крайнего столбца расширенной матрицы (добавлен столбец, состоящий из компонент вектора b) преобразованной системы.

Приведем формулы, по которым выполняется k-е преобразование. К данному моменту текущая расширенная матрица исходной системы имеет следующей вид

$$
A^{(k-1)} = \begin{bmatrix}
1 & - & A_{1k} & - & A_{1j} & - & A_{1n+1} \\
- & - & - & - & - & - & - \\
0 & - & A_{kk} & - & A_{kj} & - & A_{kn+1} \\
- & - & - & - & - & - & - \\
0 & - & A_{ik} & - & A_{ij} & - & A_{in+1} \\
- & - & - & - & - & - & - \\
0 & - & A_{nk} & - & - & - & A_{nn+1}
\end{bmatrix}
\begin{array}{l}
\\ \\ \ldots k \\ \\ \ldots i \\ \\ \\
\end{array}
$$

Первоначально компонента матрицы A_{kk} приводится к единице, путем деления k-й правой подстроки на компонент A_{kk}:

$$
A_{kj} = \frac{A_{kj}}{A_{kk}}, \quad j = k \ldots (n+1). \qquad (2.27)
$$

Затем обнуляются все остальные компоненты k-го столбца расширенной матрицы, путем вычитания из каждой i-й строки k-й строки, умноженной на A_{ik}:

$$A_{ij} = A_{ij} - A_{ik}A_{kj}, \quad i = 1...n, i \neq k;\ j = k...(n+1). \tag{2.28}$$

Замечание. Легко проверить, что преобразования матрицы (2.27) и (2.28) эквивалентно матричной операции $L^{(k)}A^{(k-1)}$, где $A^{(k-1)}$ – исходная матрица после $(k-1)$-го преобразования, а матрица $L^{(k)}$ имеет вид (отличается от единичной матрицы, размерности $n \times n$, только k-ым столбцом):

$$L^{(k)} = \begin{vmatrix} 1 & - & L_{1k}^{(k)} & - & 0 \\ - & - & - & - & - \\ 0 & - & L_{kk}^{(k)} & - & 0 \\ - & - & - & - & - \\ 0 & 0 & L_{nk}^{(k)} & - & 1 \end{vmatrix}, \tag{2.29}$$

где $L_{ik}^{(k)} = -\dfrac{A_{ik}^{(k-1)}}{A_{kk}^{(k-1)}},\ \ i = 1,n, i \neq k, \quad L_{ik}^{(k)} = \dfrac{1}{A_{kk}^{(k-1)}}$.

Приведем алгоритм, реализующий данный метод:

```
det = 1   (переменная для хранения значения определителя)
for k = 1,n
{
   вызов процедуры выбора ведущего элемента
   for j = k+1,n+1
   {
      A_kj = A_kj / A_kk
   }
   for i = 1,n
   {
      if i ≠ k then
      {
         for j = k+1,n+1
         {
            A_ij = A_ij - A_ik * A_kj
         }
      }
   }
}
```

(2.30)

Здесь:

1) в пределах каждого *k*-го преобразования *k*-й столбец явно не обрабатывается, поскольку он в последующих преобразованиях не используется;

2) каждое очередное преобразование матрицы предваряется процедурой выбора ведущего элемента, которая приведена ниже. Переменная det используется процедурой выбора ведущего элемента для хранения значения определителя.

$k\max = k$
$A\max = |A_{kk}|$
if $k < n$ *then*
{
  for $i = (k+1), n$
  {
    $A\text{mod} = |A_{ik}|$
    if $A\text{mod} > A\max$ *then*
    {
      $A\max = A\text{mod}$
      $k\max = i$
    }
  }
}
if $|A\max| < nul\ (\approx 0)$ *then*
{
  $\det = 0$ (*матрица вырожденная*)
  выход из процедуры
}
if $k\max \neq k$ *then*
{
  for $j = k, (n+1)$ (*переставляем строки*)
  {
    $buf = A_{kj}$
    $A_{kj} = A_{(k\max)\,j}$
    $A_{(k\max)\,j} = buf$
  }
  $\det = -\det$
}
$\det = \det * A_{kk}$ (*вычисление определителя*)

Здесь *nul* – переменная для хранения машинного нуля, используемого для проверки действительных чисел на нуль

(обычно $nul \le 10^{-10}$ и может корректироваться пользователем для фильтрации уровня ошибок округления, возникающих в процессе вычисления).

Программа, использующяя метод оптиамльного исключения, приведена в п. 2.2.4.

2.2.3. Нахождение обратной матрицы методом оптимального исключения Гаусса–Жордана

Полное преобразование матрицы методом оптимального исключения может быть записано в матричном виде

$$L^{(n)}T^{(n)}...L^{(1)}T^{(1)}A = E,$$

из которого следует матричная форма нахождения обратной матрицы (так как по определению $A^{-1}A = E$)

$$A^{-1} = L^{(n)}T^{(n)}...L^{(1)}T^{(1)}E, \qquad (2.31)$$

где $T^{(i)}$ – матрица перестановок при i-ом преобразовании, а $L^{(i)}$ задано (2.29).

Согласно выражению (2.31), если исходную матрицу A дополнить справа единичной матрицей E размерности $n \times n$, то в качестве алгоритма нахождения обратной матрицы можно использовать алгоритм (2.30), заменив в нем правые пределы циклов с $n + 1$ на $n + n$. При этом ту же замену необходимо выполнить и в процедуре выбора ведущего элемента (2.24) при перестановке строк.

В противовес сказанному, можно предложить более экономичный (по размеру используемой памяти и количеству операций) алгоритм нахождения обратной матрицы. В нем выбор ведущего элемента осуществляется в k-й подстроке с перестановкой соответствующих столбцов. В этом случае полное преобразование исходной матрицы будет записываться в следующем матричном виде

$$L^{(n)}...L^{(1)}AT^{(1)}...T^{(n)} = E.$$

Отсюда последовательно получаем

$$
\begin{array}{l}
AT^{(1)}...T^{(n)} = (L^{(n)}...L^{(1)})^{-1}E \\
\Downarrow \\
(AT^{(1)}...T^{(n)})^{-1} = (L^{(n)}...L^{(1)})E \\
\Downarrow \\
(T^{(1)}...T^{(n)})^{-1}A^{-1} = (L^{(n)}...L^{(1)})E \\
\Downarrow \\
A^{-1} = (T^{(1)}...T^{(n)})(L^{(n)}...L^{(1)})E
\end{array}
$$

Таким образом, если единичную матрицу подвергнуть тем же преобразованиям (кроме перестановки столбцов), что и матрицу A, то для окончательного построения обратной матрицы из преобразованной единичной матрицы необходимо в последней переставить между собой строки с номерами, которые соответствуют номерам переставленных в процессе преобразования столбцов, но в обратной последовательности. Естественно, номера переставляемых столбцов предварительно должны быть сохранены в двумерном вспомогательном массиве.

Обсудим детали оптимального алгоритма расчета обратной матрицы. Обозначим соответствующие матрицы после $(k-1)$-го преобразования

$$A^{(k-1)} = L^{(k-1)}...L^{(1)}AT^{(1)}...T^{(k-1)}, \qquad B^{(k-1)} = L^{(k-1)}...L^{(1)}E.$$

Они имеют вид

$$
A^{(k-1)} = \begin{bmatrix}
1 & - & 0 & A_{1k}^{(k-1)} & - & A_{1n}^{(k-1)} \\
- & - & - & - & - & - \\
0 & - & 1 & A_{k-1k}^{(k-1)} & - & A_{k-1n}^{(k-1)} \\
0 & - & 0 & A_{kk}^{(k-1)} & - & A_{kn}^{(k-1)} \\
- & - & - & - & - & - \\
0 & - & 0 & A_{nk}^{(k-1)} & - & A_{nm}^{(k-1)}
\end{bmatrix},
$$

$$
B^{(k-1)} = \begin{bmatrix}
B_{11}^{(k-1)} & - & B_{1k-1}^{(k-1)} & 0 & - & 0 \\
- & - & - & - & - & - \\
B_{k-11}^{(k-1)} & - & B_{k-1k-1} & 0 & - & 0 \\
B_{1k}^{(k-1)} & - & B_{kk-1}^{(k-1)} & 1 & - & 0 \\
- & - & - & - & - & - \\
B_{n1}^{(k-1)} & - & B_{nk-1}^{(k-1)} & 0 & - & 1
\end{bmatrix}.
$$

Обратим внимание на следующие обстоятельства.

1. В первой матрице левая часть, а во второй правая совпадают с фрагментом единичной матрицы, причем эти фрагменты таковы, что дополняют друг друга до полной единичной матрицы.

2. В процессе *k*-го преобразования в первой матрице *k*-й столбец переходит в *k*-й столбец единичной матрицы (поскольку он в дальнейшем не используется, то его явно можно не обрабатывать), а во второй матрице появляется *k*-й столбец, отличный от *k*-го столбца единичной матрицы.

Все это позволяет сделать вывод, что обе матрицы можно хранить на месте исходной и во время *k*-го преобразования обрабатывать по единым формулам (2.32), которые получены в результате обобщения матричных операций $A^{(k)} = L^{(k)}(A^{(k-1)}T^{(k)})$, $B^{(k)} = L^{(k)}B^{(k-1)}$:

выбор ведущего элемента в к – ой правой подстроке $(\neq 0)$

$$
\begin{aligned}
&A_{ij} = A_{ij} - \frac{A_{ik}A_{kj}}{A_{kk}} \quad i, j = 1, n;\ i \neq k,\ j \neq k \\
&A_{ik} = -\frac{A_{ik}}{A_{kk}} \quad i = 1, n;\ i \neq k \\
&A_{kj} = \frac{A_{kj}}{A_{kk}} \quad j = 1, n;\ j \neq k \\
&A_{kk} = \frac{1}{A_{kk}}
\end{aligned}
\tag{2.32}
$$

Теперь приведем оптимальный алгоритм в развернутом виде (если возвращается нулевое значение определителя матрицы, то матрица – вырожденная или плохо обусловлена):

```
det = 1, m = 0 (инициализация переменных)
for k = 1, n
{
   вызов процедуру выбора ведущего элемента в подстроке
(см. ниже)
   for i = 1, n
   {
```

if $i \neq k$ *then*

{

$A_{ik} = -\dfrac{A_{ik}}{A_{kk}}$

for $j = 1, n$

{

if $j \neq k$ *then* $\{ A_{ij} = A_{ij} + A_{ik} A_{kj} \}$

}

}

}

for $j = 1, n$

{

if $j \neq k$ *then* $\{ A_{kj} = \dfrac{A_{kj}}{A_{kk}} \}$

}

$A_{kk} = \dfrac{1}{A_{kk}}$

}

вызов процедуры, переставляющей строки в обработанной матрице (*см. ниже*)

Алгоритм процедуры выбора ведущего элемента. Здесь для хранения номеров переставляемых столбцов используется массив $M_{ij}, i = 1, 2;\ j = 1, n$, а в переменной *m* хранится число выполненных перестановок:

$k\max = k$

$A\max = |A_{kk}|$

if $k < n$ *then*

{

for $j = (k+1), n$

{

$A\mathrm{mod} = |A_{jk}|$

if $A\mathrm{mod} > A\max$ *then*

{

$A\max = A\mathrm{mod}$

$k\max = j$

}

}

```
}
if |A max| < nul (= 1E^-30 ≈ 0) then
{
    det = 0 (матрица вырожденная)
    выход из процедуры
}
if k max ≠ k then
{
    for i = 1, n (переставляем столбцы)
    {
        buf = A_ik
        A_ik = A_ik max
        A_ik max = buf
    }
    det = −det
    m = m + 1
    M_1m = k
    M_2m = k max
}
det = det* A_kk (вычисление определителя)
```

Процедура, переставляющая строки в преобразованной матрице

```
if m = 0 then выход из процедуры
for l = m, 1 step = −1 (переставляем строки в обратном порядке)
{
   m1 = M_1l
   m2 = M_2l
   for j = 1, n
   {
      buf = A_m1 j
     A_m1 j = A_m2 j
     A_m2 j = buf
   }
}
```

В заключение в качестве примера приведем экранную форму программы (рис. 2.6), которая для введенной матрицы рассчитывает ее определитель (использованы описанные выше алгоритмы) и обратную матрицу. Пользователь последовательно задает размерность матрицы и вводит значения ее компонент (выбирает ячейку в

таблице, вводит значение компоненты в соответствующем текстовом поле и кнопкой ENTER на клавиатуре возвращает это значение в таблицу). После «кликания» управляющей кнопки «найти обратную матрицу» программа на месте исходной матрицы формирует рассчитанную обратную матрицу. Одновременно выдается значение определителя исходной матрицы и число использованных перестановок столбцов.

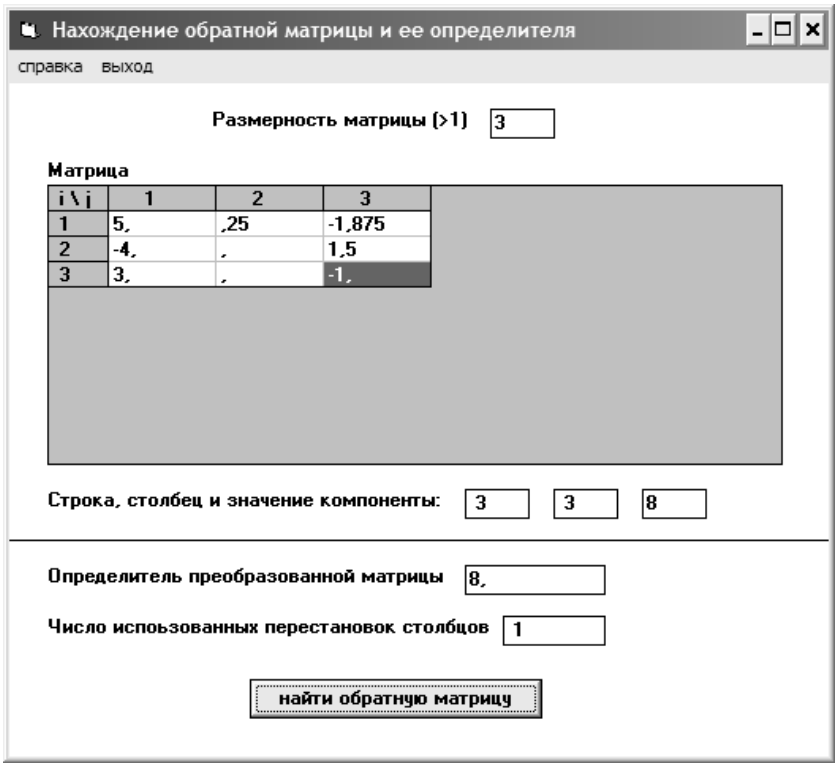


Рис. 2.6. Экранная форма программы, на которой сформирована обратная матрица для исходной матрицы $\begin{bmatrix} 0 & 2 & 3 \\ 4 & 5 & 0 \\ 0 & 6 & 8 \end{bmatrix}$

Вопросы и задания для самоконтроля

1. В чем суть метода исключения Гаусса?
2. Для каких целей в прямых методах используется процедура «выбора ведущего элемента»?
3. Запишите алгоритм для *k*-го преобразования исходной матрицы в методе Гаусса.
4. Запишите алгоритм нахождения решения системы с нижней треугольной матрицей.
5. Чем метод Гаусса отличается от оптимального метода исключения Гаусса-Жордана?
6. Что такое матрица перестановок? Каковы ее свойства?
7. Напишите алгоритм, который меняет местами *i*-ю и *k*-ю строки матрицы *A*.
8. Каким образом выбирается ведущий элемент при нахождении обратной матрицы методом Гаусса–Жордана?
9. Почему при нахождении обратной матрицы необходимо запоминать номера переставляемых столбцов при выборе ведущего элемента?

2.2.4. Первый метод ортогонализации

Представим систему $Ax = b$ в следующем виде

$$\sum_{i=1}^{n} A_{ij} x_j + A_{in+1} = 0, \quad i = 1, n, \tag{2.33}$$

где $A_{in+1} = -b_i$.

Если ввести $(n+1)$-мерные векторы $A^i = (A_{i1}, .., A_{in}, A_{in+1})$, $i = 1, n$; $y = (x_1, ..., x_n, 1)$, то эти уравнения можно заменить системой скалярных произведений

$$(A^i, y) = 0, \quad i = 1, n. \tag{2.34}$$

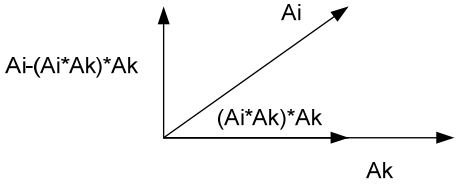
Соотношения (2.34) позволяют дать другую формулировку для исходной задачи: в $(n+1)$-мерном пространстве R^{n+1} найти $(n+1)$-мерный вектор y с $y_{n+1} = 1$, перпендикулярный n-мерной гиперплоскости, «натянутой» на векторы A^i, $i = 1, n$. Если удастся построить такой вектор, то первые n компонент этого вектора будут являться компонентами решения исходной системы уравнений.

Предлагается следующая схема решения этой задачи.

1. К исходной расширенной матрице добавляем снизу строку с компонентами (0,0,…,1). Если матрица исходной системы невырожденная, то строки этой новой матрицы в пространстве R^{n+1} образуют полную линейно-независимую систему векторов. Это следует из того, что ее определитель равен определителю исходной матрицы, который для невырожденной системы отличен от нуля.

2. Применяя к указанной полной линейно-независимой системе векторов процедуру ортогонализации Грама–Шмидта (2.35), строим ортонормированный базис для пространства R^{n+1} (рис. 2.7).

Рис. 2.7. Графическое пояснение операции процедуры Грамма–Шмидта



$$
\begin{array}{l}
\textit{для } k = 1, n \\
\quad A^k = \dfrac{A^k}{\left\| A^k \right\|}, \quad \textit{где} \left\| A^k \right\| = \sqrt{(A^k, A^k)} \\
\quad \textit{для } i = k+1, n+1 \\
\qquad A^i = A^i - (A^i, A^k) A^k \\
\quad \textit{конец } i \\
\textit{конец } k \\
A^{n+1} = \dfrac{A^{n+1}}{\left\| A^{n+1} \right\|}
\end{array}
\qquad (2.35)
$$

Последний орт этой системы (последняя строка расширенной матрицы) будет перпендикулярен n-мерной гиперплоскости, «натянутой» на векторы $A^i, i = 1, n$.

Детализируем отдельные операции алгоритма процедуры ортогонализации.

1) расчет скалярного произведения $sk(i,k) = (A^i, A^k) = \sum_{j=1}^{n+1} A_{ij} A_{kj}$

$s = 0$
для $j = 1, n+1$
$s = s + A_{ij} A_{kj}$
конец j
$sk = s$

2) нормировка вектора $A^k = \dfrac{A^k}{\left\| A^k \right\|}$

$norm = sk(k,k)$
если $norm \approx 0$, *то* "*матрица вырожденная*", *иначе* $norm = \sqrt{norm}$
для $j = 1, n+1$
$A_{kj} = \dfrac{A_{kj}}{norm}$
конец j

Нулевое значение подкоренного выражения означает линейную зависимость исходных векторов, т.е. вырожденность матрицы системы $Ax=b$;

3) корректировка вектора $A^i = A^i - (A^i, A^k) A^k$

$sk\_ik = sk(i,k)$
для $j = 1, n+1$
$A_{ij} = A_{ij} - sk\_ik\, A_{kj}$
конец j

3. Поделив последнюю строку матрицы на ее последнюю компоненту, получаем решение исходной системы

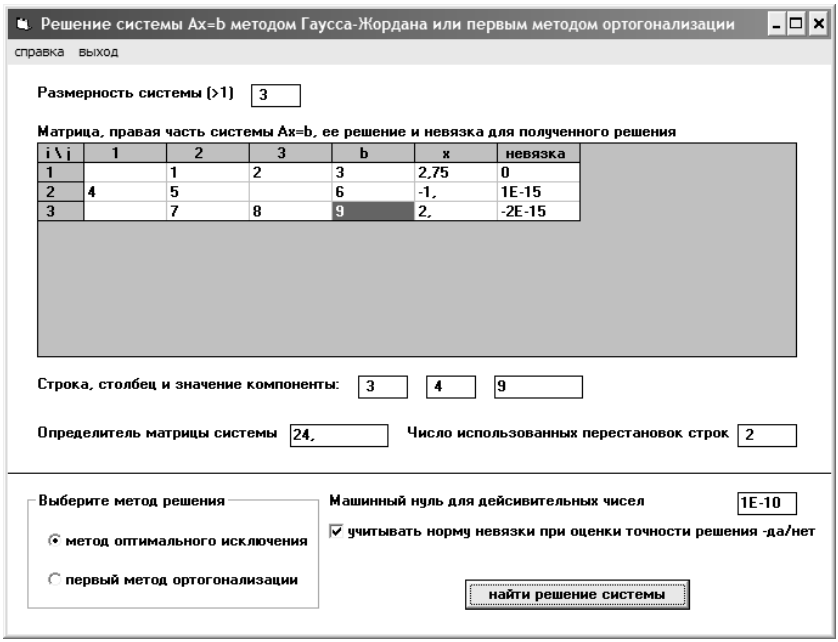
для $k = 1, n$

$$x_k = \frac{A_{n+1k}}{A_{n+1n+1}} \qquad (2.36)$$

конец k

Замечание. Алгоритм (2.36) необходимо поставить вместо последнего оператора в схеме (2.35), которая является общей записью алгоритма ортогонализации.

На рис. 2.8 приведены экранные формы программы (основное окно и окно справки), которая с помощью рассмотренных выше алгоритмов (пп. 2.2.2 и 2.2.4) находит решение и компоненты вектора невязки ($\delta_i = b_i - \sum_{j=1}^{n} A_{ij} x_j$) для заданной системы уравнений $Ax=b$.



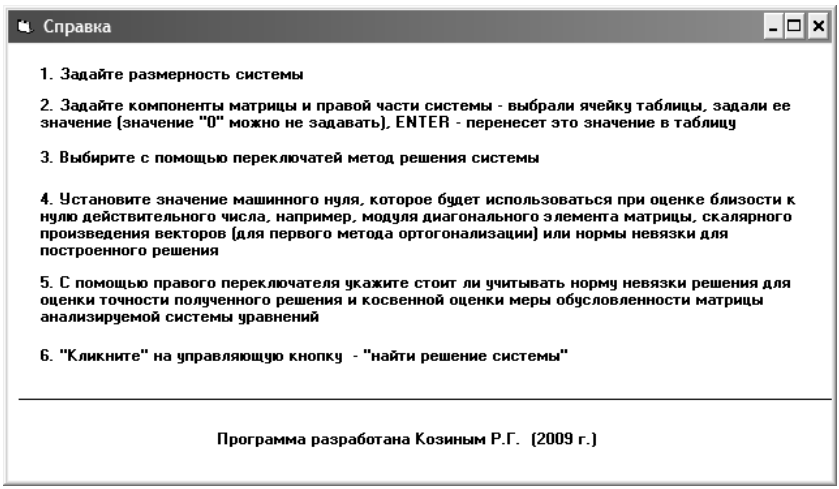


Рис. 2.8. Экранные формы программы, которая находит решения произвольной системы $Ax = b$

Здесь предусмотрены возможность корректировки значения машинного нуля (переменной *nul*), используемого для сравнения действительных чисел с нулем, а также включение и отключение дополнительного критерия плохо обусловленной матрицы $\|\delta\|_{\text{куб}} < nul$ (наряду с критериями $|A_{kk}| < nul$ или $norm < nul$ соответственно для методов Гаусса–Жордана и ортогонализации).

На рис. 2.9 и 2.10 приведены результаты использования этих дополнительных возможностей для системы уравненей с вырожденной матрицей. Сранение этих результатов наглядно показывает, к каким существенным измениниям в решении x приводит незначительная входная погрешность в правой части системы b.

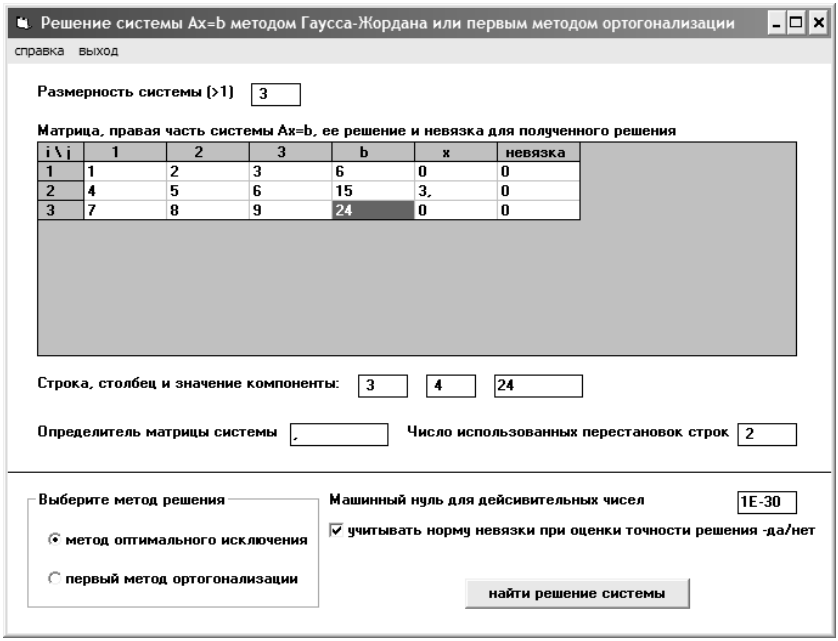


Рис. 2.9. Применение метода исключения к системе с вырожденной матрицой (ее определитель равен нулю) при $nul = 1E^{-30}$. Благодаря «пропуску» ошибок округления удалось получить для данной системы одно из частных ее решений (другое – $x = (1,1,1)$)

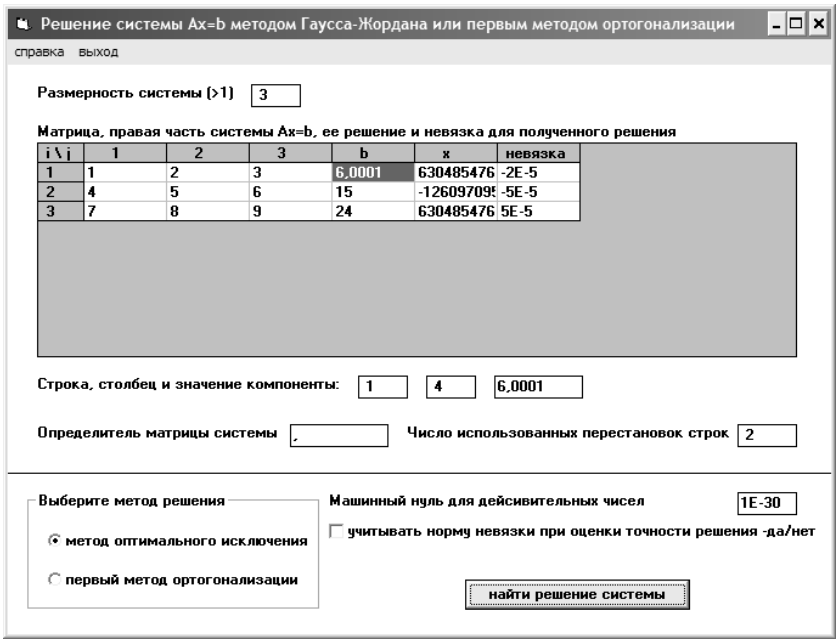


Рис. 2.10. Результат применения метода исключения к системе с вырожденной матрицей и отключенным вторым критерием по невязке. Небольшое изменение правой части системы *b* привело к полной перестройке решения системы *x* (сравните с рис. 2.9)

2.2.5. Второй метод ортогонализации

Этот метод применим к системам с симметричной положительно определенной матрицей. Поскольку только с помощью такой матрицы можно определить следующее скалярное произведение (x, Ay), удовлетворяющее всем свойствам, присущим скалярному произведению:

1) $(x, Ay) = (y, Ax)$;

2) $(x + z, Ay) = (x, Ay) + (z, Ay)$;

3) $(\lambda x, Ay) = \lambda (x, Ay)$;

4) $(x, Ax) > 0$, при $\|x\| > 0$.

Например, проверим свойство 1, так как выполнение остальных свойств очевидно:

$$(x, Ay) = \sum_i x_i \sum_k A_{ik} y_k = \sum_i \sum_k A_{ik} x_i y_k;$$

$$(y, Ax) = \sum_k y_k \sum_i A_{ki} x_i = \sum_i \sum_k A_{ki} x_i y_k.$$

Видно, что для симметричной матрицы правые, а следовательно, и левые части этих выражений равны.

Теперь рассмотрим, как строится решение системы $Ax = b$ данным методом.

1. Берем в пространстве R^n полную линейно-независимую систему векторов E^j, $j = 1, n$, в виде столбцов единичной матрицы (ее определитель равен 1)

$$E = \begin{matrix} E^1 \ldots\ldots E^n \\ \begin{vmatrix} 1\ 0 \ldots\ldots 0 \\ 0\ 1 \ldots\ldots 0 \\ \ldots\ldots\ldots\ldots \\ 0\ 0 \ldots\ldots 1 \end{vmatrix} \end{matrix}.$$

2. Применяя процедуру ортогонализации Грама–Шмидта к данной системе векторов, строим в пространстве R^n ортонормированный базис в смысле скалярного произведения (x, Ay):

для $k = 1, n-1$

$$E^k = \frac{E^k}{\left\| E^k \right\|}, \quad \text{где} \left\| E^k \right\| = \sqrt{(E^k, AE^k)}$$

для $j = k+1, n$

$$E^j = E^j - (E^j, AE^k) E^k$$

конец j

конец k

$$E^n = \frac{E^n}{\left\| E^n \right\|}$$

При реализации алгоритма ортогонализации все матричные операции следует заменить циклами по компонентам векторов, например:

$$E^j = E^j - (E^j, AE^k)E^k \Rightarrow \text{для } i = 1,n \{E_{ij} = E_{ij} - (E^j, AE^k)E_{ik}\}),$$

а перед вычислением корня, необходимо проверять, что подкоренное выражение положительное. Невыполнение этого условия означает, что матрица системы $Ax = b$ не является положительно определенной.

Приведем алгоритм расчета скалярного произведения $sk(j,k) \equiv (E^j, AE^k) = \sum_{i=1}^{n} E_{ij} \sum_{m=1}^{n} A_{im} E_{mk}$ с учетом хранения векторов в виде столбцов единичной матрицы:

$s = 0$
для $i = 1,n$
  $buf = 0$
  для $m = 1,n$
    $buf = buf + A_{im} E_{mk}$
  конец m
  $s = s + E_{ij} buf$
конец i
$sk = s$

3. Ищем решение системы в виде разложения по векторам построенного базиса, сформированного на месте единичной матрицы

$$x = \sum_{j=1}^{n} c_j E^j \tag{2.37}$$

или через компоненты вектора x

$$x_i = \sum_{j=1}^{n} E_{ij} c_j, \quad i = 1,n. \tag{2.38}$$

Для определения неизвестных коэффициентов разложения подставим разложение (2.37) в исходную систему $Ax = b$, а затем умножим ее скалярно последовательно на вектора $E^i, i = 1,n$:

$$\sum_{j=1}^{n} c_j (E^i, AE^j) = (E^i, b), \quad i = 1,n. \tag{2.39}$$

В виду ортонормированности нового базиса в левой части уравнения (2.39) останется только одно слагаемое с $j = i$, равное c_i. Отсюда следует, что

$$c_i = (E^i, b) = \sum_{k=1}^{n} E_{ki} b_k. \qquad (2.40)$$

Видно, что метод потребует значительного количества операций, но в методическом плане он представляет определенный интерес. По этой причине с ним полезно познакомиться.

2.2.6. Метод квадратного корня (метод Холецкого)

Метод применим только для систем с симметричной положительно определенной матрицей.

Согласно этому методу исходная матрица представляется в виде произведения двух матриц

$$A = BB^T. \qquad (2.41)$$

где B – нижняя треугольная матрица.

Такое разложение возможно только для симметричной положительно определенной матрицы:

1) $A^T = (BB^T)^T = (B^T)^T B^T = BB^T = A$;

2) $(Ax, x) = (BB^T x, x) = (B^T x, B^T x) \geq 0$.

Чтобы найти рекуррентные формулы для определения компонент матрицы B, запишем матричное соотношение (2.41) для четырехмерного случая

$$\begin{vmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ A_{31} & A_{32} & A_{33} & \\ A_{41} & A_{42} & A_{43} & A_{44} \end{vmatrix} = \begin{vmatrix} B_{11} & & & \\ B_{21} & B_{22} & & \\ B_{31} & B_{32} & B_{33} & \\ B_{41} & B_{42} & B_{43} & B_{44} \end{vmatrix} * \begin{vmatrix} B_{11} & B_{21} & B_{31} & B_{41} \\ & B_{22} & B_{32} & B_{42} \\ & & B_{33} & B_{43} \\ & & & B_{44} \end{vmatrix} =$$

$$\begin{vmatrix} B_{11}^2 & & & \\ B_{11}B_{21} & B_{21}^2 + B_{22}^2 & & \\ B_{11}B_{31} & B_{31}B_{21} + B_{32}B_{22} & B_{31}^2 + B_{32}^2 + B_{33}^2 & \\ B_{11}B_{41} & B_{41}B_{21} + B_{42}B_{22} & B_{41}B_{31} + B_{42}B_{32} + B_{43}B_{33} & B_{41}^2 + B_{42}^2 + B_{43}^2 + B_{44}^2 \end{vmatrix}$$

Отсюда, сравнивая компоненты левой и правой матриц, получим

$$B_{11}=\sqrt{A_{11}};$$

$$B_{21}=\frac{A_{21}}{B_{11}},\quad B_{22}=\sqrt{A_{22}-B_{21}^2}\left(=\sqrt{\frac{A_{11}A_{22}-A_{21}^2}{A_{11}}}\right);$$

$$B_{31}=\frac{A_{31}}{B_{11}},\quad B_{32}=\frac{A_{32}-B_{31}B_{21}}{B_{22}},\quad B_{33}=\sqrt{A_{33}-B_{31}^2-B_{32}^2};$$

$$B_{41}=\frac{A_{41}}{B_{11}},\quad B_{42}=\frac{A_{42}-B_{41}B_{21}}{B_{22}},\quad B_{43}=\frac{A_{43}-B_{41}B_{31}-B_{42}B_{32}}{B_{33}},$$

$$B_{44}=\sqrt{A_{44}-B_{41}^2-B_{42}^2-B_{43}^2}.$$

Здесь в двух верхних строчках под корнем явно присутствуют определители первых миноров исходной матрицы, а согласно критерию Сильвестра для положительно определенной матрицы они положительны. Последнее утверждение справедливо и для остальных подкоренных выражений. Это еще раз подтверждает необходимость того, чтобы матрица была положительно определенной.

Эти формулы по индукции позволяют записать следующий алгоритм расчета всех компонент матрицы B для n-мерного случая

$$\begin{gathered}
B_{11}=\sqrt{A_{11}};\\
B_{21}=\frac{A_{21}}{B_{11}},\quad B_{22}=\sqrt{A_{22}-B_{21}^2};\\
\text{для } k=3,\ n\\
\{\\
B_{k1}=\frac{A_{k1}}{B_{11}};\\
\text{для } i=2,\ k-1\\
\{\\
B_{ki}=\frac{A_{ki}-\sum_{j=1}^{i-1}B_{kj}B_{ij}}{B_{ii}}\\
\}\\
B_{kk}=\sqrt{A_{kk}-\sum_{j=1}^{k-1}B_{kj}^2}\\
\}
\end{gathered}\qquad (2.42)$$

При программной реализации этого алгоритма перед вычислением корня необходимо предварительно проверять является ли подкоренное выражение положительным. Если это условие не выполняется, то исходная матрица не положительно определенная. Отметим, что этот алгоритм можно использовать для проверки положительной определенности произвольной симметричной матрицы.

После того как разложение матрицы A выполнено, система $Ax = b$ распадается на две системы с треугольными матрицами

$$Ax = b \Rightarrow BB^T x = b \Rightarrow \begin{array}{l} By = b, \\ B^T x = y \end{array}$$

или в покомпонентной записи

$$\begin{array}{ll} B_{11}y_1 & = b_1 \\ B_{21}y_1 + B_{22}y_2 & = b_2 \\ \ldots\ldots\ldots\ldots & \\ B_{i1}y_1 + B_{i2}y_2 + \ldots + B_{ii}y_i & = b_i \\ \ldots\ldots\ldots\ldots & \end{array}$$

$$\begin{array}{l} \ldots\ldots\ldots\ldots \\ B_{ii}x_i + B_{i+1i}x_{i+1} + \ldots. + B_{ni}x_n = y_i \\ \ldots\ldots\ldots\ldots \\ B_{n-1n-1}x_{n-1} + B_{nn-1}x_n = y_{n-1} \\ B_{nn}x_n = y_n \end{array}$$

Их решение рассчитывается элементарно

$$y_1 = \frac{b_1}{B_{11}};$$

для $i = 2, n$ $$\{ y_i = \frac{b_i - \sum_{j=1}^{i-1} B_{ij}y_j}{B_{ii}} \};$$

$$x_n = \frac{y_n}{B_{nn}};$$

для $i = (n-1), 1$ $$\{ x_i = \frac{y_i - \sum_{j=i+1}^{n} B_{ji}x_j}{B_{ii}} \}.$$

Последовательное вычисление компонент промежуточного вектора *y* можно выполнять в алгоритме (2.42) после определения компонент соответствующей строки матрицы *B* (см. приведенный ранее алгоритм). При этом для экономии памяти матрицу *B* можно хранить на месте исходной матрицы *A*, а вектор *y* формировать на месте вектора *x*.

Необходимо помнить, что все суммы, встречающиеся в алгоритмах, вычисляется путем введения соответствующих внутренних циклов.

В заключение привем листинг на Visual Basic процедуры-функции, реализующей метод квадратного корня с использованием приведенных алгоритмов:

```
Function kvroot () As Integer
  Dim i%, j%, k%
   kvroot = 0    'матрица положительно-определенная
   If Not (a(0, 0) > 0) Then
      kvroot = 1 'матрица не положительно-
                 определенная
      Exit Function
   End If
   a(0, 0) = Sqr(a(0, 0))
   x(0) = a(0, n) / a(0, 0)
   a(1, 0) = a(1, 0) / a(0, 0)
   x(1) = a(1, n)
   a(1, 1) = a(1, 1) - a(1, 0) * a(1, 0)
   If Not (a(1, 1) > 0) Then
      kvroot = 1  'матрица не положительно-
                  определенная
      Exit Function
   End If
   a(1, 1) = Sqr(a(1, 1))
   x(1) = (x(1) - a(1, 0) * x(0)) / a(1, 1)
   If n = 2 Then GoTo cont
   For k = 2 To n - 1
     a(k, 0) = a(k, 0) / a(0, 0)
     x(k) = a(k, n) - a(k, 0) * x(0)
     For i = 1 To k - 1
       For j = 0 To i - 1
```

```
        a(k, i) = a(k, i) - a(k, j) * a(i, j)
      Next j
      a(k, i) = a(k, i) / a(i, i)
      x(k) = x(k) - a(k, i) * x(i)
    Next i
    For j = 0 To k - 1
      a(k, k) = a(k, k) - a(k, j) * a(k, j)
    Next j
    If Not (a(k, k) > 0) Then
      kvroot = 1 'матрица не положительно-
                 определенная
      Exit Function
    End If
    a(k, k) = Sqr(a(k, k))
    x(k) = x(k) / a(k, k)
  Next k
cont:
  x(n - 1) = x(n - 1) / a(n - 1, n - 1)
  For i = n - 2 To 0 Step -1
    For j = i + 1 To n - 1
      x(i) = x(i) - a(j, i) * x(j)
    Next j
    x(i) = x(i) / a(i, i)
  Next i
End Function
```

Здесь систама хранится в расширенной матрице с компонентами: `a(0,0),…a(n-1,n)`.

На рис. 2.11 представлен скриншот программы, которая позволяет найти решение проивольной системы линейных уравнений как вторым методом ортогонализации, так и методом квадратного корня. Для преобразования произвольной системы в равносильную систему с симметричной и положительно определенной матрицей следует использовать кнопку «Умножить систему на А(т)» – умножить систему на транспонированную исходную матрицу. Есть возможность задать систему уравнений случайным образом. Это позволяет провести сравнительный анализ точности (по норме невязки полученного решения) используемых методов.

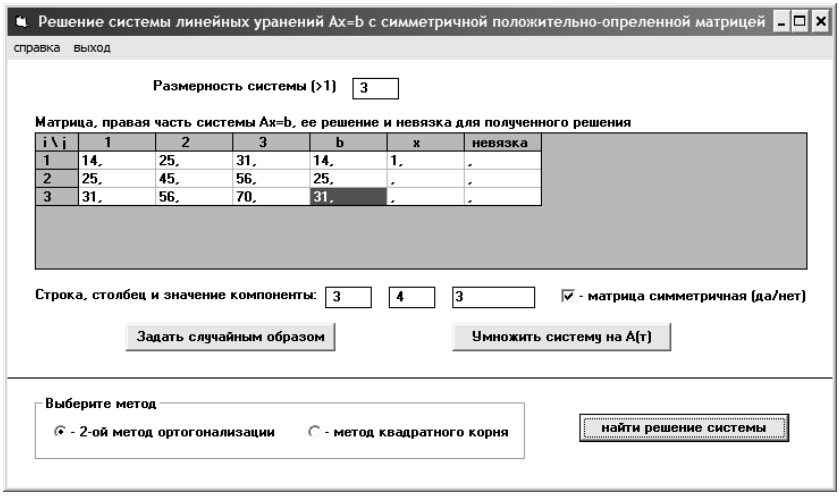


Рис. 2.11. Экранная форма программы, которая реализуют два метода, используемые для положительно определенной симметричной матрицы.

2.2.7. Метод прогонки

Метод прогонки является самым экономичным способом решения систем с трех диагональной матрицей

$$b_1 x_1 + c_1 x_2 = d_1;$$

.....

$$a_k x_{k-1} + b_k x_k + c_k x_{k+1} = d_k, \quad k = 2,\ n-1; \qquad (2.43)$$

.....

$$a_n x_{n-1} + b_n x_n = d_n.$$

Здесь для хранения системы использованы четыре вектора a, b, c, d, причем компоненты $a_1 = c_n = 0$ не используются.

К таким системам сводится разностное решение краевых задач для дифференциальных уравнений второго порядка.

Введем рекуррентные зависимости

$$x_k = \alpha_k x_{k+1} + \beta_k, \quad k = 1,\ n-1. \qquad (2.44)$$

Найдем формулы для расчета входящих в (2.44) неизвестных параметров α_k, β_k. Сравнивая первое уравнение в (2.43) с соотношением (2.44), сразу получаем

$$\alpha_1 = -\frac{c_1}{b_1}, \quad \beta_1 = \frac{d_1}{b_1}. \tag{2.45}$$

Далее, заменим в k-ом уравнении системы (2.43) неизвестное x_{k-1} согласно формуле (2.44)

$$a_k(\alpha_{k-1}x_k + \beta_{k-1}) + b_k x_k + c_k x_{k+1} = d_k.$$

Отсюда легко выводим рекуррентные формулы для последовательного определения остальных неизвестных параметров α_k, β_k :

$$\alpha_k = -\frac{c_k}{t_k}, \quad \beta_k = \frac{d_k - a_k\beta_{k-1}}{t_k}, \quad k = 2, n-1, \tag{2.46}$$
$$t_k = b_k + a_k\alpha_{k-1}.$$

Вычисления по формулам (2.45), (2.46) называется прямым ходом прогонки.

Теперь, подставляя в последнее уравнение (2.43) соотношение (2.44) с $k = n - 1$, получим

$$a_n(\alpha_{n-1}x_n + \beta_{n-1}) + b_n x_n = d_k \quad \Rightarrow \quad x_n = \frac{d_n - a_n\beta_{n-1}}{b_n + a_n\alpha_{n-1}}. \tag{2.47}$$

Формулы (2.47) и (2.44) (с $k = n$ –1, n – 2,...,1) позволяют рассчитать все компоненты x_k неизвестного решения системы (2.43). Эти расчеты называются обратным ходом прогонки.

Покажем методом индукции, что для устойчивости метода прогонки ($|b_k + a_k\alpha_{k-1}| > 0$) достаточно выполнения следующих соотношений между параметрами системы

$$|b_k| \geq |a_k| + |c_k| > 0 \quad \text{при } \forall k. \tag{2.48}$$

При выполнении условия (2.48) имеем (см. (2.45)) $|b_1| > 0$ и $|\alpha_1| < 1$. Предположим, что $|\alpha_{k-1}| < 1$. Тогда $|b_k + a_k\alpha_{k-1}| > 0$. Кроме того, $|b_k + a_k\alpha_{k-1}| > |c_k|$ и, следовательно, $|\alpha_k| < 1$ и т.д.

Заметим, что метод прогонки легко обобщается на системы с пяти (и более) диагональными матрицами. В этом случае исходное рекуррентное соотношение следует брать в виде

$$x_k = \alpha_k x_{k+1} + \beta_k x_{k+2} + \gamma_k .$$

Замечание. При программной реализации метода, в целях экономии памяти, параметры α_k , β_k можно сохранять на месте компонент векторов b_k , d_k, а решение вернуть в векторе *c*. В этом случае алгоритм метода будет иметь вид

прямой ход прогонки

$d_1 = \dfrac{d_1}{b_1}, b_1 = -\dfrac{c_1}{b_1}$

для $k = 2, (n-1)$

{

$t = b_k + a_k b_{k-1}$

$d = \dfrac{d_k + a_k d_{k-1}}{t}$

$b_k = -\dfrac{c_k}{t}$

}

обратный ход прогонки

$c_n = \dfrac{d_n - a_n d_{n-1}}{b_n + a_n b_{n-1}}$

для $k = (n-1), 1$

{

$c_k = b_k c_{k+1} + d_k$

}

2.3 Метод регуляризации для решения плохо обусловленных систем

Решение системы уравнений *Ax = b* с плохо обусловленной матрицей существенно зависит от ошибок в исходных данных и ошибок округления (рис. 2.12, где для таких систем схематично показаны соответствующие области входных и выходных ошибок).

Согласно методу регуляризации для уменьшения этой зависимости, т.е. сужения области поиска, на решение системы накладывается дополнительное вполне разумное требование, чтобы его норма наименее уклонялось от нуля. В такой постановке

исходную систему можно свести к задаче минимизации следующего квадратичного функционала

$$\min_x F(x) = (Ax - b, Ax - b) + \alpha(x, x) \geq 0, \qquad (2.49)$$

где $\alpha > 0$ – весовой коэффициент, называемый параметром регуляризации. Его значение должен быть таким, чтобы с одной стороны обеспечить достаточную обусловленность системы, а с другой – хорошую аппроксимацию исходной задачи.

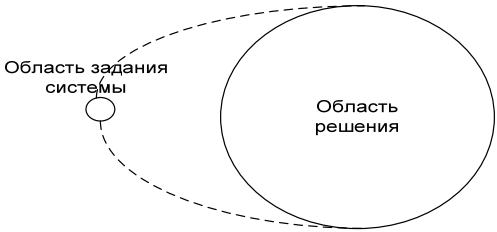


Рис. 2.12. Области входных и выходных ошибок

Для квадратичного функционала (2.49) решение, обеспечивающее минимум функционалу, находится аналитически из условия $\frac{dF}{dx} = 0$. Для этого распишем (2.49) в виде

$$F(x) = (x, A^T Ax) - 2(x, A^T b) + (b, b) + \alpha(x, x). \qquad (2.50)$$

Отсюда, беря производную $\frac{dF}{dx}$ с учетом очевидного преобразования $(x, A^T Ax) = (A^T Ax, x) = (x, (A^T A)^T x) = (x, A^T Ax)$ и приравнивая ее нулю, получим новую систему уравнений

$$(A^T A + \alpha E)x = A^T b. \qquad (2.51)$$

Ее решение x_α, называемое нормальным, зависит от α и может быть получено любым рассмотренным ранее методом.

В заключение укажем способ выбора параметра α. Обычно на практике проводится ряд вычислений с различными значениями α. После этого оптимальным считают либо то его значение, при

котором выполняется условие приближенного равенства невязки решения сумме погрешностей исходных данных

$$\left\|\delta(x_\alpha)\right\| \equiv \left\|Ax_\alpha - b\right\| \approx \left\|\delta b\right\| + \left\|\delta A\, x_\alpha\right\|, \qquad (2.52)$$

либо то из них, для которого невязка минимальна (рис. 2.13).

Соответствующее этому α решение x_α принимается за нормальное решение исходной системы.

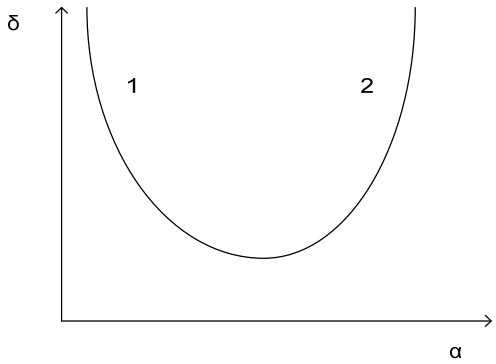


Рис. 2.13. Зависимость навязки δ от параметра α, где в зоне 1 преобладает ошибка, связанная с особенностью матрицы системы, а в зоне 2 – ошибки аппроксимации, связанные с изменением исходной системы

На рис. 2.14 и 2.15 приведены результаты работы программы, позволяющей использовать метод регуляризации для нахождения решения систем уравнений с плохо обусловленными матрицами.

Вопросы и задания для самоконтроля

1. При каком условии вектор $y = (x_1, ..., x_n, 1)$, используемый в первом методе ортогонализации, будет линейно-независимым от векторов-строк расширенной матрицы A?

2. Какие манипуляции с векторами выполняет процедура Грамма–Шмидта?

3. Как с помощью процедуры Грамма–Шмидта можно проверить: является ли система векторов линейно независимой?

4. Для каких систем линейных уравнений можно использовать второй метод ортогонализации и почему?

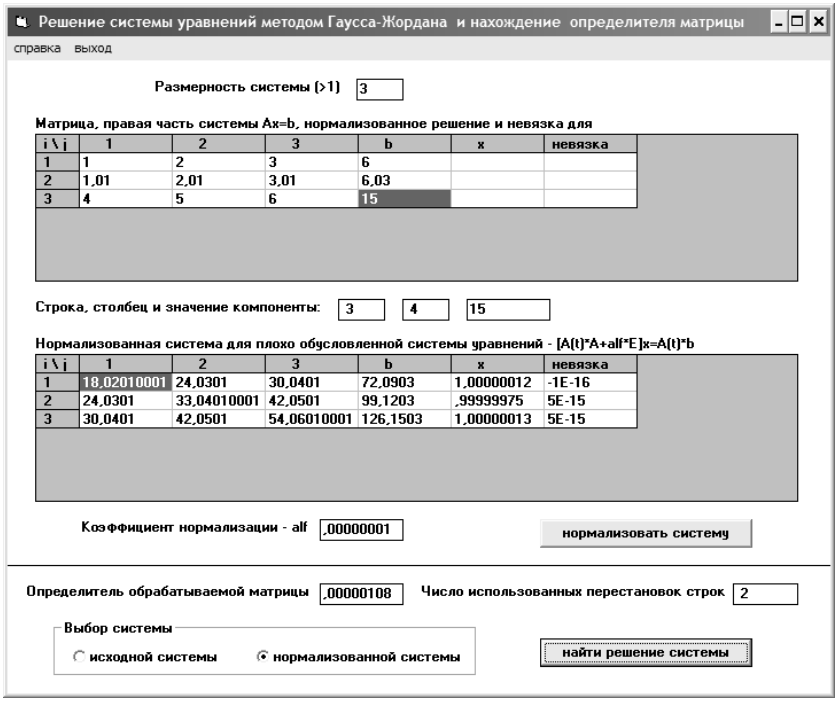


Рис. 2.14. Получено нормальное решение системы с плохо обусловленной матрицей. Заметим, что точное решение исходной системы равно (1; 1; 1). Видно, что решить непосредственно исходную систему методом Гаусса-Жордано не удалось.

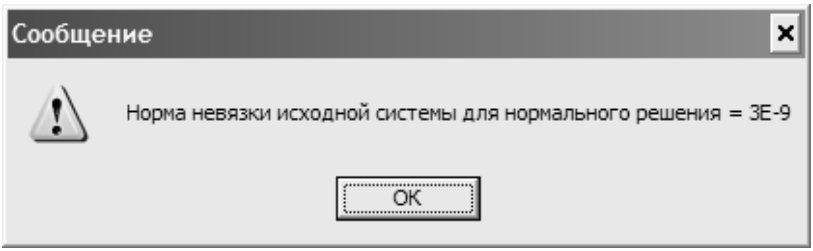


Рис. 2.15. Норма невязки исходной системы для нормального решения

5. В чем сущность метода квадратного корня и почему он применяется только для систем с симметричной положительно определенной матрицей?

6. Запишите алгоритм модифицированного метода ортогонализации Грамма–Шмидта.

7. Как подбирается параметр α в методе регуляризации, используемом при решении плохо обусловленных систем линейных уравнений?

8. Запишите условие сходимости метода прогонки.

9. Составьте алгоритм метода прогонки.

10. Распишите через компоненты скалярное произведение двух столбцов единичной матрицы $sk(j,k) \equiv (E^j, AE^k)$ и составьте алгоритм для вычисления его значения.

Глава 3. МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ

3.1. Собственные значения и собственные векторы матрицы

Собственным числом (значением) матрицы называется действительное или комплексное число λ, которое удовлетворяет системе

$$Ax = \lambda x \quad \Rightarrow \quad (Ax - \lambda E)x = 0. \tag{3.1}$$

При этом нетривиальный вектор x является собственным вектором, соответствующим данному λ.

Для того чтобы система (3.1) имела нетривиальное решение, должно выполнятся равенство

$$\det(Ax - \lambda E) = 0. \tag{3.2}$$

Уравнение (3.2), называемое **характеристическим полиномом** матрицы A, представляет собой полином степени n относительно λ и имеет n корней. Приведем некоторые свойства собственных значений и векторов.

1. Матрица A порядка $n \times n$ имеет n собственных значений.

2. Для каждого уникального λ существует, по крайней мере, один собственный вектор (так как не всегда k-кратное собственное число имеет k собственных векторов). Каждый собственный вектор определяет для данной матрицы только собственное направление в пространстве R^n.

3. Собственные векторы, соответствующие различным λ, линейно-независимы.

Доказательство этого утверждения проведем «по принципу индукции».

Имеем $a_1 x_1 = 0 \quad \Rightarrow \quad a_1 = 0$.

Пусть $\sum_{i=1}^{k} a_i x_i = 0$ при $a_i = 0, i = 1, k$, т.е. векторы x_1, ..., x_k – линейно-независимы.

Докажем, что векторы x_1, ..., x_k, x_{k+1} – также линейно-независимые. Доказательство проведем «от противного». Предположим, что это не так, т.е. что

$$\sum_{i=1}^{k+1} a_i x_i = 0 \qquad (3.3)$$

и не все $a_i = 0$. Умножим равенство (3.3) на матричный оператор $(A - \lambda_{k+1}E)$:

$$(A-\lambda_{k+1}E)\sum_{i=1}^{k+1} a_i x_i = \sum_{i=1}^{k+1} A a_i x_i - \sum_{i=1}^{k+1} \lambda_{k+1} a_i x_i = \sum_{i=1}^{k+1} (\lambda_i - \lambda_{k+1}) a_i x_i = \\ = \sum_{i=1}^{k} (\lambda_i - \lambda_{k+1}) a_i x_i = 0. \qquad (3.4)$$

Но по предположению вектора $x_1, ..., x_k$ – линейно-независимые, и поэтому

$$(\lambda_i - \lambda_{k+1}) a_i = 0 \quad \Rightarrow \quad a_i = 0,$$

так как $(\lambda_i - \lambda_{k+1}) \neq 0$ для $i = 1, k$. Отсюда согласно равенству (3.3) $a_{k+1} = 0$ и, следовательно, в (3.3) все $a_i = 0$, $i = 1, k+1$. А это означает, что все векторы $x_1, ..., x_k, x_{k+1}$ – линейно-независимые.

4. Если все собственные числа матрицы простые, то соответствующие им собственные вектора образуют полную линейно-независимую систему векторов в пространстве R^n, так что любой вектор из этого пространства может быть представлен в виде линейной комбинации этих векторов.

5. Собственные числа действительной, симметричной матрицы являются действительными числами. Кроме того, если матрица положительно определенная, то ее собственные числа положительные. Пусть x – собственный вектор матрицы A. Тогда

$$(Ax, x) = \lambda(x, x). \qquad (3.5)$$

Отсюда, так как (Ax, x) для симметричной матрицы и (x, x) действительные числа, следует, что λ – также действительное. Если матрица положительно определенная, то в (3.5) $(Ax, x) > 0$, $(x, x) > 0$, и поэтому $\lambda > 0$.

6. Симметричная, действительная матрица всегда имеет полную ортонормированную систему собственных векторов, которую можно использовать как базис для пространства R^n.

7. Матрицы A, A^{-1} и $B = \sum_{i=1}^{k} c_i A^i$ имеют одни и те же собственные векторы, а собственные числа двух последних матриц $\lambda(A^{-1})$, $\lambda(B)$

связаны с собственными числами матрицы A ($\lambda(A)$) соотношениями:

$$Ax = \lambda(A)x \Rightarrow x = \lambda(A)A^{-1}x \Rightarrow A^{-1}x = \frac{1}{\lambda(A)}x \Rightarrow \lambda(A^{-1}) = \frac{1}{\lambda(A)};$$

$$Bx = \lambda(B)x \Rightarrow \sum_{i=1}^{k} c_i A^i x = \lambda(B)x \Rightarrow \sum_{i=1}^{k} c_i \lambda(A)^i x = \lambda(B)x \Rightarrow \Rightarrow \lambda(B) = \sum_{i=1}^{k} c_i \lambda(A)^i. \quad (3.6)$$

8. Преобразование матрицы A вида $C = B^{-1}AB$, где матрица B – невырожденная ($\det(B) \neq 0$), называется подобным, а сами матрицы A и C подобными. Подобные матрицы имеют одинаковые характеристические полиномы, так как

$$\det(C - \lambda E) = \det(B^{-1}AB - \lambda E) = \det(B^{-1}AB - \lambda B^{-1}EB) =$$
$$= \det(B^{-1})\det(A - \lambda E)\det(B) = \det(A - \lambda E)$$

учтено, что $\det(B^{-1}) = \frac{1}{\det(B)}$, и, следовательно, одинаковые собственные числа, а их собственные векторы (соответственно x и y) связаны соотношением

$$Cy = \lambda y \Rightarrow B^{-1}ABy = \lambda y \Rightarrow A(By) = \lambda(By) \Rightarrow x = By.$$

Если $B^{-1} = B^T$, то преобразование, выполняемое над A с помощью такой матрицы, называется ортогональным подобным преобразованием.

9. Для любой нормы матрицы справедливо неравенство $|\lambda_i| \leq \|A\|$. Оно следует из цепочки соотношений:

$$Ax = \lambda x \Rightarrow \|A\|\|x\| \geq \|Ax\| = \|\lambda x\| = |\lambda|\|x\|.$$

В заключение приведем два полезных соотношения, которые на практике могут быть использованы для нахождения двух последних неизвестных собственных значений или проверки надежности рассчитанной совокупности собственных значений:

$$\det(A) = \lambda_1 ... \lambda_n; \qquad trace(A) \equiv \sum_{i=1}^{n} A_{ii} = \lambda_1 + ... + \lambda_n.$$

Подумайте: будут ли эти соотношения выполняться для действительной матрицы при наличии у нее комплексных собственных значений?

3.2. Решение частичной проблемы собственных чисел для симметричной матрицы

Рассмотрим итерационный метод нахождения максимального по модулю собственного числа симметричной матрицы. Все собственные числа такой матрицы действительные, а собственные вектора e_i, $i = 1, n$, образуют в пространстве R^n ортонормированный базис.

Замечание. Этот метод применим и для обычной матрицы, если у нее существует полный набор собственных векторов и собственные числа действительные. В этом случае может пригодиться теорема Пиррона: если все компоненты матрицы положительные ($A_{ij} > 0$), то максимальное собственное число матрицы простое и положительное $\lambda_1 > 0$.

Случай 1. Пусть $\lambda_1 = \lambda_2 = ... = \lambda_r$ и $|\lambda_1| > |\lambda_{r+1}| > ... > |\lambda_n|$. Возьмем произвольный вектор $x^{(0)} = \sum_{i=1}^{n} c_i e_i$ и будем последовательно умножать его на матрицу A:

$$x^{(k)} = Ax^{(k-1)} = \sum_{i=1}^{n} c_i \lambda_i^k e_i. \qquad (3.7)$$

Для полученных таким образом векторов можно построить скалярные произведения (учтено, что векторы $e_i, i = 1, n,$ ортонормированны)

$$(x^{(k-1)}, x^{(k-1)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \lambda_i^{k-1}\lambda_j^{k-1}(e_i, e_j) = \sum_{i=1}^{n} c_i^2 \lambda_i^{2k-2} =$$

$$= \lambda_1^{2k-2}(c_1 + c_2 + ... + c_r + O(\gamma^{2k-2}));$$

$$(x^{(k)}, x^{(k-1)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \lambda_i^{k}\lambda_j^{k-1}(e_i, e_j) =$$

$$= \sum_{i=1}^{n} c_i^2 \lambda_i^{2k-1} = \lambda_1^{2k-1}(c_1 + c_2 + ... + c_r + O(\gamma^{2k-1})), \qquad (3.8)$$

где $\gamma = \frac{\lambda_{r+1}}{\lambda_1} < 1$.

Отношение этих скалярных произведений при $k \to \infty$ стремится к максимальному по модулю собственному числу матрицы A:

$$\frac{(x^{(k)}, x^{(k-1)})}{(x^{(k-1)}, x^{(k-1)})} = \lambda_1 + O(\gamma^{2k-2}) \approx \lambda_1. \qquad (3.9)$$

При этом один из собственных векторов, соответствующих данному кратному собственному числу (поскольку любая линейная комбинация этих собственных векторов также является собственным вектором $A(\sum_{i=1}^{r} c_i e_i) = \lambda_1(\sum_{i=1}^{r} c_i e_i)$, т.е. эти векторы определяют r-мерную собственную гиперплоскость для λ_1), определяется выражением

$$\frac{x^{(k)}}{\left\|x^{(k)}\right\|} = \frac{\sum_{i=1}^{n} c_i \lambda_i^k e_i}{\sqrt{\sum_{i=1}^{n} c_i^2 \lambda_i^{2k}}} = \frac{\lambda_1^k (c_1 e_1 + c_2 e_2 + ... + c_r e_r + O(\gamma^k))}{\sqrt{\lambda_1^{2k}(c_1^2 + c_2^2 + ... + c_r^2 + O(\gamma^{2k}))}} = \qquad (3.10)$$

$$= e_1 + O(\gamma^k) \approx e_1.$$

Чтобы для λ_1 найти остальные собственные векторы, необходимо повторить описанную итерационную процедуру, но с другими начальными векторами.

В заключение приведем модифицированный вариант итерационной процедуры, который позволяет избежать возможных аварийных ситуаций, связанных с переполнением (когда $|\lambda_1| > 1$) или «исчезновением порядка» (когда $|\lambda_1| < 1$):

$e_1^{(0)} = \frac{x^{(0)}}{\left\|x^{(0)}\right\|}$, *где* $\left\|x^{(0)}\right\| = \sqrt{(x^{(0)}, x^{(0)})}$;

$\lambda_1^{(0)} = 10^{30}$;

для $k = 1, K_{\max}$ - максимальное число итераций;

$$x^{(k)} = Ae_1^{(k-1)},\ \lambda_1^{(k)} = (x^{(k)}, e_1^{(k-1)}),\ e_1^{(k-1)} = \frac{x^{(k)}}{\left\|x^{(k)}\right\|}$$

если $\left|\lambda_1^{(k)} - \lambda_1^{(k-1)}\right| < eps$, *то выход из цикла* (3.11)

иначе $\lambda_1^{(k-1)} = \lambda_1^{(k)}$.

конец цикла по k

Такая запись справедлива, так как

$$\frac{(x^{(k)}, x^{(k-1)})}{(x^{(k-1)}, x^{(k-1)})} = \frac{(Ax^{(k-1)}, x^{(k-1)})}{\left\|x^{(k-1)}\right\|\left\|x^{(k-1)}\right\|} = (Ae^{(k-1)}, e^{(k-1)})$$

Случай 2 не возможен для положительно определенной матрицы, так как у нее все $\lambda_i > 0$.

Пусть $\lambda_1 = -\lambda_2$ и $|\lambda_1| = |\lambda_2| > |\lambda_3| ... > |\lambda_n|$. Тогда

$$s1 \equiv \frac{(x^{(k)}, x^{(k-1)})}{(x^{(k-1)}, x^{(k-1)})} = \frac{\lambda_1^{2k-1}(c_1^2 - c_2^2 + O(\gamma^{2k-1}))}{\lambda_1^{2k-2}(c_1^2 + c_2^2 + O(\gamma^{2k-2}))} \to$$

$$\to \lambda_1\left(\frac{c_1^2 - c_2^2}{c_1^2 + c_2^2}\right) + O(\gamma^{2k-2});$$

$$s2 \equiv \frac{(x^{(k)}, x^{(k)})}{(x^{(k-1)}, x^{(k-1)})} = \frac{\lambda_1^{2k}(c_1^2 + c_2^2 + O(\gamma^{2k-1}))}{\lambda_1^{2k-2}(c_1^2 + c_2^2 + O(\gamma^{2k-2}))} \to \quad (3.12)$$

$$\to \lambda_1^2 + O(\gamma^{2k-2});$$

$$e^{(k)} \equiv \frac{x^{(k)}}{\left\|x^{(k)}\right\|} = \frac{\lambda_1^k(c_1e_1 + (-1)^k c_2e_2 + O(\gamma^k))}{\sqrt{\lambda_1^{2k}(c_1^2 + c_2^2 + O(\gamma^{2k}))}} \to$$

$$\to a(c_1e_1 + (-1)^k c_2e_2) + O(\gamma^k),$$

где a = const.

Видно, что в этом случае при $k \to \infty$ $(s_1)^2 \neq s_2$, и вектор $e^{(k)}$ поочередно меняет свое «значение». Оба собственных вектора e_1 и e_2 можно определить из соотношений (здесь k – четное):

$$e^{(k)} = ac_1e_1 + ac_2e_2;$$

$$e^{(k-1)} = ac_1e_1 - ac_2e_2.$$

следующим образом (так как собственные вектора определяются с точность до константы)

$$\tilde{e}_1 \equiv 2ac_1e_1 = e^{(k)} + e^{(k-1)};$$
$$\tilde{e}_2 \equiv 2ac_2e_2 = e^{(k)} - e^{(k-1)}. \qquad (3.13)$$

Ниже представлен обобщенный модифицированный алгоритм нахождения максимального по модулю собственного числа, учитывающий возможность появления любого из случаев

$e_1^{(0)} = \dfrac{x^{(0)}}{\left\|x^{(0)}\right\|}$, *где* $\left\|\mathrm{x}^{(0)}\right\| = \sqrt{(x^{(0)}, x^{(0)})}$

$(\tilde{\lambda}_1^{(0)})^2 = 10^{30}$

для $k = 1, K\max$ - *максимальное число итераций*

$x^{(k)} = Ae_1^{(k-1)}$, $\lambda_1^{(k)} = (x^{(k)}, e^{(k-1)})$

$(\tilde{\lambda}_1^{(k)})^2 \equiv \left\|x^{(k)}\right\|^2 = (x^{(k)}, x^{(k)})$, $e_1^{(k-1)} = \dfrac{x^{(k)}}{\left\|x^{(k)}\right\|}$

если $\left|(\tilde{\lambda}_1^{(\mathrm{k})})^2 - (\tilde{\lambda}_1^{(\mathrm{k-1})})^2\right| < eps$, *то выход из цикла*

иначе $(\tilde{\lambda}_1^{(k-1)})^2 = (\tilde{\lambda}_1^{(k)})^2$

конец цикла по к

если $\left|(\lambda_1^{(k)})^2 - (\tilde{\lambda}_1^{(k)})^2\right| = 0{,}0001$, *то* $\lambda_1 = \lambda_1^{(k)}$ – *имеет место случай* 1

иначе $\lambda_1 = \sqrt{(\tilde{\lambda}_1^{(k)})^2}$ – *имеет место случай* 2

(3.14)

Посмотрим, какую информацию относительно собственных значений матрицы A можно получить, если применить процедуру (3.14) к симметричным матрицам вида:

$$B = E - \frac{A^2}{\lambda_1^2} \quad \left(B_{ij} = \delta_{ij} - \frac{\sum_{m=1}^{n} A_{im}A_{mj}}{\lambda_1^2},\ i, j = 1, n\right) \quad \text{и} \quad C = \lambda_1 E - A\,.$$

Принадлежность собственных значений матрицам будем обозначать следующим образом: $\lambda_i(A)$, $\lambda_i(B)$, $\lambda_i(C)$.

1. Определение минимального по модулю собственного числа матрицы A. Его можно рассчитать как

$$\lambda_1(B) = 1 - \frac{\lambda_n^2(A)}{\lambda_1^2(A)} \quad \Rightarrow \quad \left|\lambda_n(A)\right| = \left|\lambda_1(A)\right|\sqrt{1 - \lambda_1(B)}\,.$$

Или если матрица A положительно определенная ($A > 0$), то матрица $C \geq 0$, и поэтому

$$\lambda_1(C) = \lambda_1(A) - \lambda_n(A) > 0 \quad \Rightarrow \quad \lambda_n(A) = \lambda_1(A) - \lambda_1(C)\,.$$

2. Определение границ отрезка, содержащего все собственные значения матрицы A.

- Если при реализации процедуры (3.14) имеет место случай 2 ($\lambda_1 = -\lambda_2$), то

$$\lambda_{\min}(A) = -\lambda_1(A),\ \ \lambda_{\max}(A) = \lambda_1(A).$$

- Если матрица A положительно определенная, то

$$\lambda_{\min}(A) = \lambda_n(A) = \lambda_1(A) - \lambda_1(C) > 0,$$
$$\lambda_{\max}(A) = \lambda_1(A) > 0.$$

- Если $\lambda_1(C) > 0, \lambda_1(A) > 0$, то

$$\lambda_{\min}(A) = \lambda_1(A) - \lambda_1(C) < 0,$$
$$\lambda_{\max}(A) = \lambda_1(A) > 0.$$

- Если $\lambda_1(A) < 0, \lambda_1(C) < 0$, то

$$\lambda_{\min}(A) = \lambda_1(A) < 0,$$
$$\lambda_{\max}(A) = \lambda_1(A) - \lambda_1(C) < 0 \text{ или } > 0.$$

На рис. 3.1 и 3.2 приведены скриншоты программы, в которой использован алгоритм (3.14). Программа позволяет задать «в ручную» или случайным образом произвольную исходную матрицу A, найти для нее максимальное по модулю собственное число, с помощью этих данных сформировать два варианта вспомогательной матрицы B и определить для нее максимальное по модулю собственное число. Используя эту информацию, пользователь может рассчитать минимальное по модулю собственное число матрицы A и границы отрезка, содержащего все ее собственные значения. Например, согласно результатам, представленным на рис. 3.1 и 3.2, следует, что все собственные числа использованной матрицы лежат на отрезке

$$\lambda_{\min}(A) = \lambda_1(A) = -0,83092;$$
$$\lambda_{\max}(A) = \lambda_1(A) - \lambda_1(B) = -0,83092 - (-1,41574) = 0,58482.$$

В табл. 3.1 приведены результаты небольшого исследования, выполненного с помощью указанной программы. Строгая регулярность в представленных результатах отсутствует.

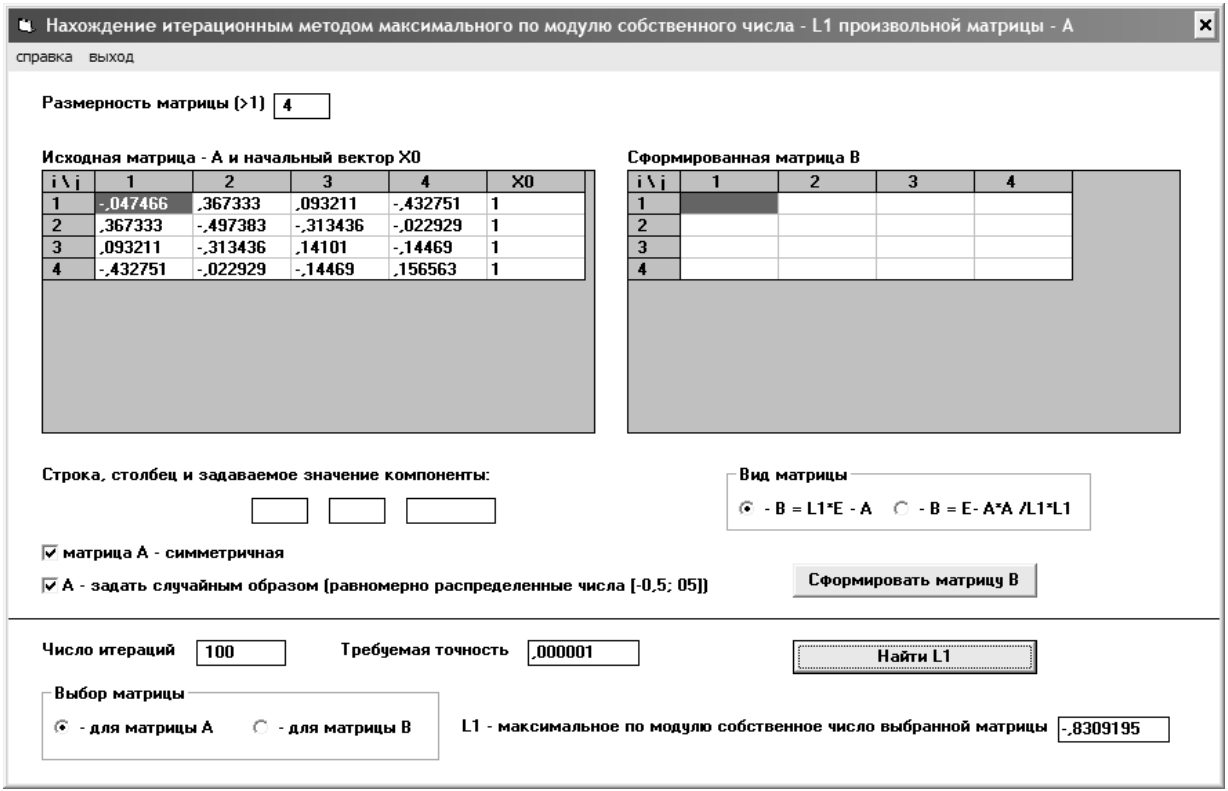


Рис. 3.1. Задана случайным образом симметричная исходная матрица *A* и найдено ее максимальное по модулю собственное число

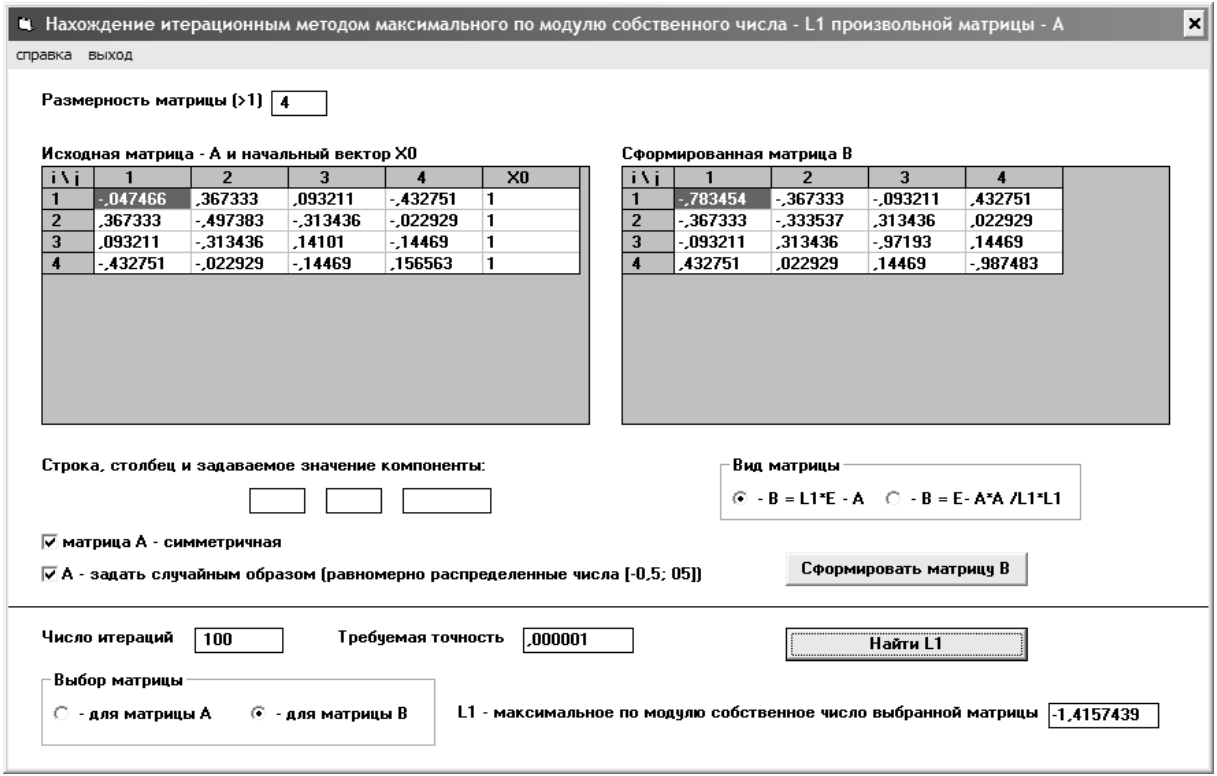


Рис. 3.2. Для исходной матрицы сформирована вспомогательная матрица *B* и найдено ее максимальное по модулю собственное число

Таблица 3.1

**Зависимость числа итераций
от размерности случайной матрицы для esp = 0,000001.
Величина выборки –5**

Размерность A	5	10	15	20
Минимум итераций	17	26	49	27
Максимум итераций	44	96	196	123

Вопросы и задания для самоконтроля

1. Для чего производят нормировку текущего вектора $e_1^{(k)} = \dfrac{x^{(k)}}{\left\| x^{(k)} \right\|}$ при реализации итерационного метода нахождения максимального собственного значения?

2. Что такое характеристический полином матрицы?

3. Почему рассмотренный итерационный метод нахождения максимального по модулю собственного значения не всегда можно применить для произвольной матрицы?

4. Как с помощью рассмотренного итерационного метода найти минимальное по модулю собственное значение матрицы?

5. Как связаны между собой собственные значения и собственные вектора следующих матриц A и $B = \sum_{i=0}^{k} A^i$?

6. Составьте алгоритмы для вычисления компонент матриц $B = E - \dfrac{A^2}{\lambda_1^2}$ и $C = \lambda_1 E - A$.

7. Какие матрицы называются подобными?

8. Чем можно объяснить отсутствие строгой регулярности в результатах, представленных в таблице 3.1?

3.3. Решение задачи на собственные значения методом Данилевского

Сущность метода Данилевского состоит в приведении матрицы A с помощью $(n-1)$ подобных преобразований к подобной ей матрице Фробениуса

$$P=\begin{bmatrix} p_1 & p_2 & p_3 & \dots & p_{n-1} & p_n \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \qquad (3.15)$$

Для последней характеристический полином записывается просто путем разложения по элементам первой строки

$$\det(P-\lambda E)=\begin{bmatrix} p_1-\lambda & p_2 & p_3 & \dots & p_{n-1} & p_n \\ 1 & -\lambda & 0 & \dots & 0 & 0 \\ 0 & 1-\lambda & \dots & 0 & 0 & \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & -\lambda \end{bmatrix}=$$

$$=(p_1-\lambda)(-\lambda)^{n-1}-p_2(-\lambda)^{n-2}+\dots+(-1)^{n+1}p_n \Rightarrow \qquad (3.16)$$

$$\Rightarrow \lambda^n-p_1\lambda^{n-1}-p_2\lambda^{n-2}-\dots-p_n=0.$$

Подобное преобразование матрицы A представляет собой последовательный перевод ее строк, начиная с n-й, в строки матрицы Фробениуса. Детально рассмотрим k-е $(k=1,2,\dots(n-1))$ преобразование, формирующее $(n-(k-1))$-ю строку матрицы P.

После $(k-1)$-го преобразования матрица имеет вид $(A^{(0)}=A)$:

$$A^{(k-1)}=\begin{bmatrix} A_{11}^{(k-1)} & \dots & A_{1n-k}^{(k-1)} & A_{1n-(k+1)}^{(k-1)} & \dots & A_{1n\_1}^{(k-1)} & A_{1n}^{(k-1)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ A_{n-(k-1)1}^{(k-1)} & \dots & A_{n-(k-1)n-k}^{(k-1)} & A_{n-(k-1)n-(k+1)}^{(k-1)} & \dots & A_{n-(k-1)n-1}^{(k-1)} & A_{n-(k-1)n}^{(k-1)} \\ 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \qquad (3.17)$$

Над ней выполняются следующие операции:

1) «сажаем» единицу на место компоненты $A^{(k-1)}_{n-(k-1)n-k}$ путем деления на нее всего $(n-k)$-го столбца

$$\tilde{A}^{(k)}_{in-k} = \frac{A^{(k-1)}_{in-k}}{A^{(k-1)}_{n-(k-1)n-k}}, \quad i = 1, n-(k-1); \tag{3.18}$$

2) обнуляем все остальные компоненты приводимой строки. Для этого вычитаем из каждого j-го столбца $(n-k)$-й столбец, умноженный на соответствующую компоненту $A^{(k-1)}_{n-(k-1)j}$ $(n-(k-1))$-й строки

$$\begin{gathered} \tilde{A}^{(k)}_{ij} = A^{(k-1)}_{ij} - \tilde{A}^{(k)}_{in-k} A^{(k-1)}_{n-(k-1)j}, \\ i = 1, n-(k-1),\ j = 1, n\ (j \neq n-k). \end{gathered} \tag{3.19}$$

Эти операции равносильны матричной операции

$$\tilde{A}^{(k)} = A^{(k-1)} M^{(n-k)},$$

где матрица $M^{(n-k)}$ отличается от единичной только $(n-k)$-й строкой, компоненты которой равны

$$\begin{aligned} M^{(n-k)}_{n-kj} &= -\frac{A^{(k-1)}_{n-(k-1)j}}{A^{(k-1)}_{n-(k-1)n-k}}, \quad j = 1, n\ (j \neq n-k); \\ M^{(n-k)}_{n-kn-k} &= \frac{1}{A^{(k-1)}_{n-(k-1)n-k}}. \end{aligned} \tag{3.20}$$

Чтобы завершить k-е подобное преобразование необходимо умножить промежуточную матрицу $\tilde{A}^{(k)}$ на матрицу $(M^{(n-k)})^{-1}$

$$A^{(k)} = (M^{(n-k)})^{-1} A^{(k-1)} M^{(n-k)} = (M^{(n-k)})^{-1} \tilde{A}^{(k)}. \tag{3.21}$$

Последняя получается из единичной матрицы подстановкой в ее $(n-k)$-ю строку компонент $(n-(k-1))$-й строки матрицы $A^{(k-1)}$. Легко проверить, что в этом случае будет выполняться необходимое соотношение

$$(M^{(n-k)})^{-1} M^{(n-k)} = E.$$

Матричная операция (3.20) приводит к изменению компонент только одной $(n-k)$-й строки $\tilde{A}^{(k)}$

$$A^{(k)}_{n-kj} = \sum_{m=1}^{n} A^{(k-1)}_{n-(k-1)m} \tilde{A}^{(k)}_{mj}, \quad j = 1, n, \tag{3.22}$$

или с учетом нулевых компонент в нижних подстолбцах матрицы $\tilde{A}^{(k)}$

$$A_{n-k\,j}^{(k)} = \begin{cases} \sum_{m=1}^{n-k} A_{n-(k-1)m}^{(k-1)}\, \tilde{A}_{m\,j}^{(k)}, \quad j = 1, n-(k+1);\ j = n; \\ \sum_{m=1}^{n-k} A_{n-(k-1)m}^{(k-1)}\, \tilde{A}_{m\,j}^{(k)} + A_{n-(k-1)\,j}^{(k-1)}, \quad j = (n-k), (n-1). \end{cases} \tag{3.23}$$

После построения характеристического полинома и нахождения его корней – собственных чисел матрицы A, легко определяются собственные вектора матрицы $P(y)$, а затем и матрицы A, так как

$$\begin{aligned} Py = \lambda y \ &\Rightarrow (M^{-1}AM)y = \lambda y \ \Rightarrow \\ &\Rightarrow A(My) = \lambda(My) \Rightarrow x = My. \end{aligned} \tag{3.24}$$

где $M = M^{(n-1)}M^{(n-2)}...M^{(2)}M^{(1)}$.

Для определения векторов y распишем систему $(P - \lambda E)y = 0$

$$\begin{cases} (p_1 - \lambda)y_1 + p_2 y_2 + ... + p_n y_n = 0; \\ y_1 - \lambda y_2 = 0; \\ ..... \\ y_{n-1} - \lambda y_n = 0. \end{cases} \tag{3.25}$$

Положив $y_n = 1$ (это можно сделать, так как собственные векторы определяются с точностью до постоянной), из последнего и последующих (до первого) уравнений легко находим

$$y = \left\{ y_n = 1,\ y_{n-1} = \lambda,\ y_{n-2} = \lambda^2, ..., y_1 = \lambda^{n-1} \right\}. \tag{3.26}$$

При таких значениях y_i первое уравнение системы (3.25) удовлетворяется автоматически, поскольку оно совпадает с характеристическим уравнением (3.2).

Теперь с учетом вида матриц $M^{(n-k)}$

$$M^{(n-k)} = \begin{bmatrix} 1 & 0 & ..... & 0 & 0 \\ ..... & & & & \\ M_{n-k1}^{(n-k)} & M_{n-k2}^{(n-k)} & ..... & M_{n-kn-1}^{(n-k)} & M_{n-kn}^{(n-k)} \\ ..... & & & & \\ 0 & 0 & ..... & 0 & 1 \end{bmatrix} \tag{3.27}$$

легко рассчитываются компоненты собственных векторов x (первоначально устанавливается $x = y$)

$$x_{n-k} = \sum_{j=1}^{n} M_{n-k\,j}^{(n-k)} x_j, \quad k = (n-1),1. \tag{3.28}$$

Остановимся на особенностях программной реализации данного метода.

1. Обрабатываемая матрица хранится на месте исходной.

2. Для повышения устойчивости алгоритма перед выполнением очередного k-го подобного преобразования необходимо выполнять процедуру выбора «ведущего» элемента: найти среди компонент $A_{n-(k-1)\,j}^{(k-1)}, j = 1,(n-k),$ максимальный по модулю элемент и, если $j_{\max} \neq (n-k)$, то переставить местами верхние части столбцов и строки с номерами $(n - k)$ и $j_{\max}$. Такая перестановка ($TA^{(k-1)}T$) будет подобной, поскольку для матрицы перестановок T справедливо соотношение $T^{-1} = T$. Если окажется, что «ведущий» элемент равен нулю, то имеет место сингулярный случай, когда характеристический полином распадается на произведение двух характеристических полиномов:

$$\det(A - \lambda E) = \begin{bmatrix} A^{(n-k)} - \lambda E & B \\ 0 & P^{(k)} - \lambda E \end{bmatrix} =$$

$$= \det(A^{(n-k)} - \lambda E)\ \det(P^{(k)} - \lambda E),$$

причем для одного из них матрица Фробениуса $P^{(k)}$ уже получена, а для подматрицы $A^{(n-k)}$ ее предстоит еще найти с помощью того же метода. В этом особом случае собственные векторы матрицы следует искать вне данного метода (см. п. 3.5).

3. Строку $M_{n-k\,j}^{(n-k)}, j = 1,n,$ (см. формулы (3.20)) матрицы $M^{(n-k)}$ можно первоначально формировать в дополнительном векторе m, а в конце k-го преобразования сохранять на месте $(n - (k - 1))$-й обработанной строки исходной матрицы. При этом формулы (3.18), (3.19) перепишутся следующим образом:

$$\begin{aligned} &\tilde{A}_{i\,j}^{(k)} = A_{i\,j}^{(k-1)} + A_{i\,n-k}^{(k-1)} m_j, \quad i = 1,(n-k), \quad j = 1,n\ (j \neq n-k); \\ &\tilde{A}_{i\,n-k}^{(k)} = A_{i\,n-k}^{(k-1)} m_{n-k}, \quad i = 1,(n-k). \end{aligned} \tag{3.29}$$

Здесь исключена явная обработка (n – (k – 1))-й приводимой строки, поскольку ее компоненты используются при завершении k-го подобного преобразования (см. формулы (3.23)).

В соответствии с изложенным выше, была разработана программа, позволяющая построить матрицу Фробениуса для произвольной матрицы, а затем определить методом парабол корни характеристического полинома этой матрицы. Программа успешно работает и случаи вырождения маирицы Фробениуса в две и более подматриц. Скриншоты этой программы представлены на рис. 3.3 – 3.7. Обратите внимание, что, как и положено, сумма следов двух матриц Фробениуса равна следу исходной матрицы (21+7=28), а суммы корней каждой из подматриц совпадает с ее следом.

Ниже приведены листинги на Visual Basic алгоритма, вычисляющего матрицу Фробениуса, и прицедуры выбора ведущего элемента, используемой на каждом шаге алгоритма.

```
For k = 1 To n - 1
  kk = kk + 1
  If v_element(k) = 1 Then   'выделена подматрица Фробениуса
    code = 1
    Exit For
  End If
  For j = 0 To n - 1
    ma(j) = a(n - k, j)  'сохраняем обрабатываемую строку
  Next j
  'ставим 1 в обрабатываемой строке
  For i = 0 To n - k
    a(i, n - 1 - k) = a(i, n - 1 - k) / a(n - k, n - 1 - k)
  Next i
  'ставим 0 вместо остальных элементов обрабатываемой строки
  For i = 0 To n - k
    For j = 0 To n - 1
      If Not (j = n - 1 - k) Then
        a(i, j) = a(i, j) - a(i, n - 1 - k) * a(n - k, j)
      End If
    Next j
  Next i
  'умножаем на обратную матрицу (M)
  For j = 0 To n - 1
    buf = 0
    For m = 0 To n - 1
```

```
      buf = buf + ma(m) * a(m, j)
    Next m
    a(n - 1 - k, j) = buf
  Next j
Next k

Function v_element (k As Integer) As Integer
  Dim amax#, jmax%, i%, j%, buf#, cd%
  cd = 0
  amax = Abs(a(n - k, n - 1 - k))
  jmax = n - 1 - k
  For j = 0 To n - 2 - k
    buf = Abs(a(n - k, j))
    If buf > amax Then
      amax = buf
      jmax = j
    End If
  Next j
  If Not (k = jmax) Then
    For i = 0 To n - 1
      buf = a(i, n - 1 - k)
      a(i, n - 1 - k) = a(i, jmax)
      a(i, jmax) = buf
    Next i
    For j = 0 To n - 1
      buf = a(n - 1 - k, j)
      a(n - 1 - k, j) = a(jmax, j)
      a(jmax, j) = buf
    Next j
  End If
  If amax < 1E-30 Then cd = 1
  code = cd
  v_element = cd
End Function
```

Здесь `ma(n)` – вспомогательный массив для временного хранения обрабатываемой строки; `n` – размерность обрабатываемой матрицы; `v_element` – функция, реализующая процедуру выбора ведущего элемента.

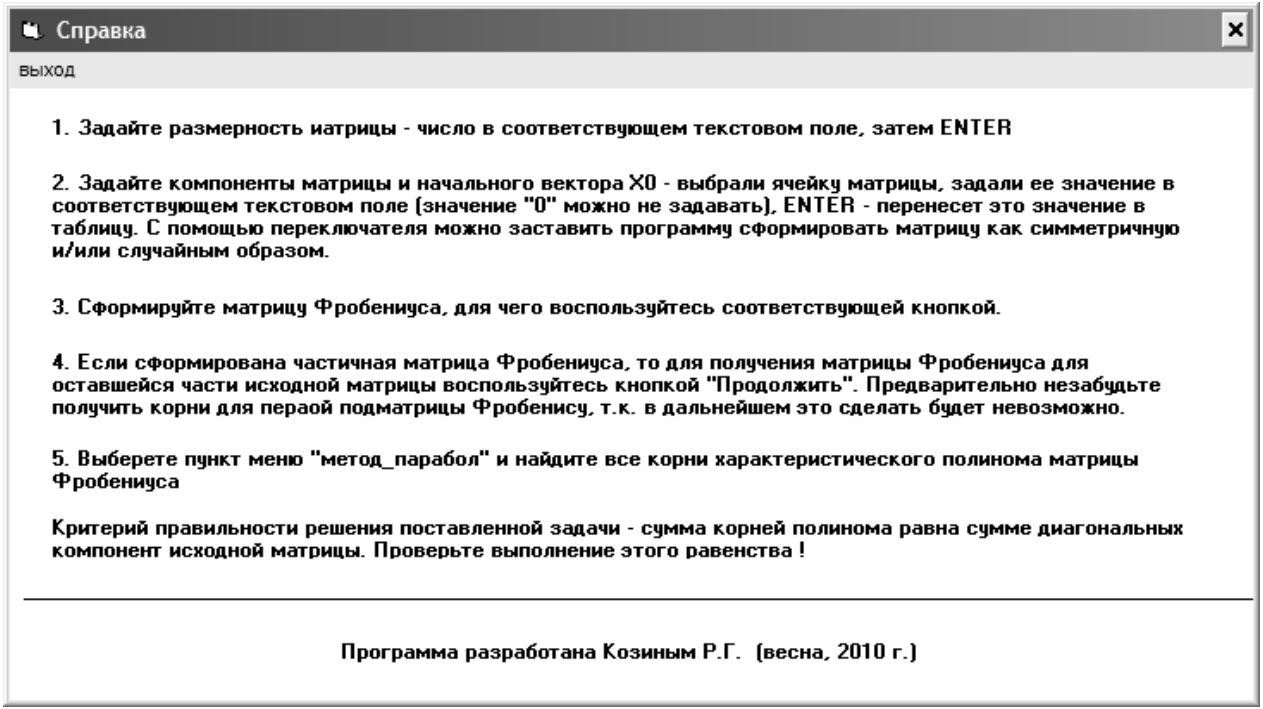


Рис. 3.3. Форма, с описанием порядка работы с программой

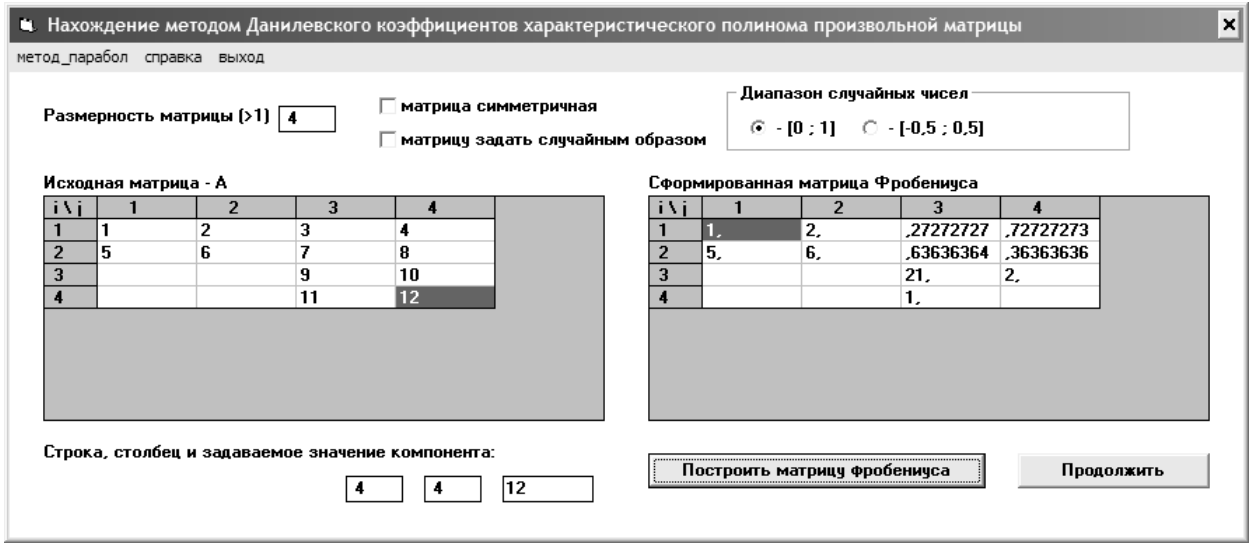


Рис. 3.4. Главная форма программы, на которой приведен пример матрицы, для которой матрица Фробениуса распадается на две подматрицы. Здесь приведен этап получения первой подматрицы

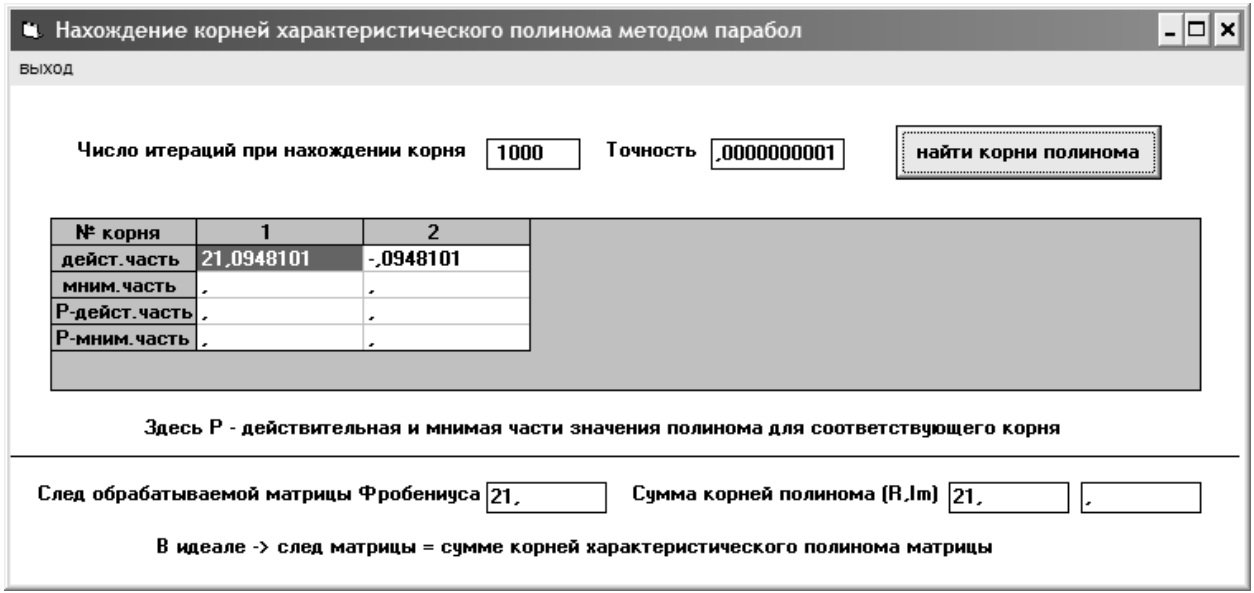


Рис. 3.5. Корни характеристического полинома, полученные методом парабол для первой подматрицы Фробениуса

Рис. 3.6. Главная форма программы после получения второй подматрицы Фробениуса

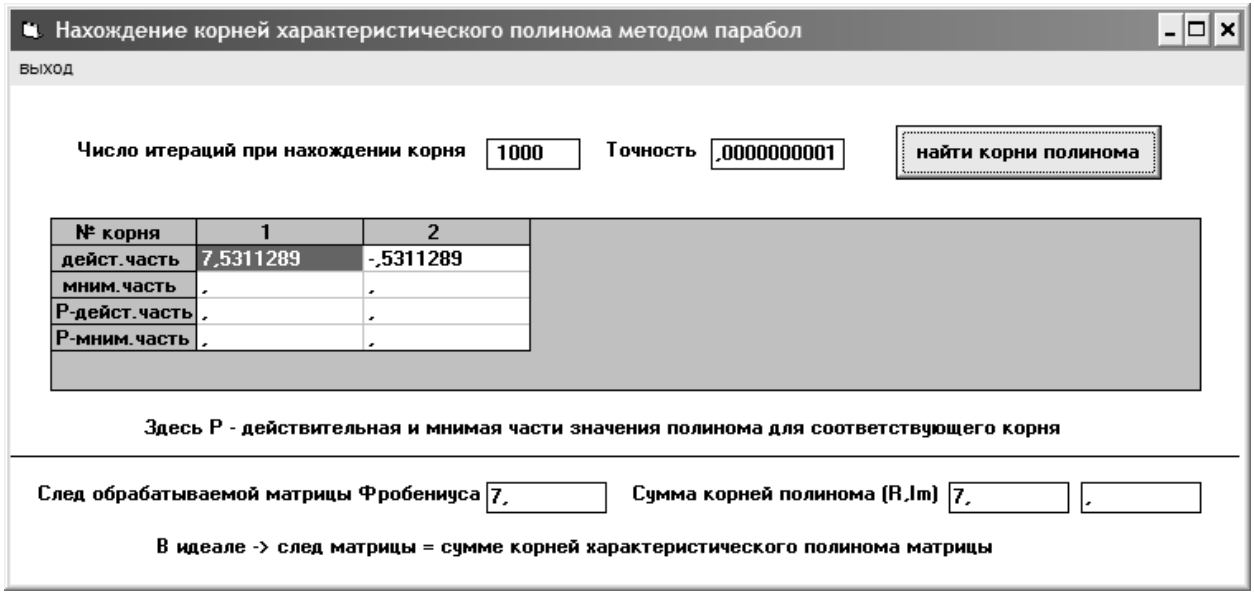


Рис. 3.7. Корни характеристического полинома, полученные методом парабол для второй подматрицы Фробениуса

3.4. Метод Крылова

В данном методе нахождение коэффициентов характеристического полинома базируется на использовании матричного тождества Гамильтона–Кели

$$A^n + p_1 A^{n-1} + p_2 A^{n-2} + ... + p_n E = 0. \qquad (3.30)$$

где p_i – коэффициенты характеристического полинома

$$\det(A - \lambda E) = \lambda^n + p_1 \lambda^{n-1} + p_2 \lambda^{n-2} + ... + p_n = 0.$$

Возьмем произвольный вектор $y^{(0)} \neq 0$ и умножим на него (3.30). В результате получим векторное уравнение относительно неизвестных p_i:

$$A^n y^{(0)} + p_1 A^{n-1} y^{(0)} + p_2 A^{n-2} y^{(0)} + ... + p_n y^{(0)} = 0$$

или в новых обозначениях $y^{(k)} = A y^{(k-1)} = A^k y^{(0)}$

$$p_1 y^{(n-1)} + p_2 y^{(n-2)} + ... + p_n y^{(0)} = -y^{(n)}. \qquad (3.31)$$

Эту систему линейных уравнений можно переписать в привычной форме как

$$Yp = -y^{(n)}, \qquad (3.32)$$

где

$$Y = \left[y^{(n-1)}\ y^{(n-2)} \ldots y^{(0)} \right];$$

$$p = \left(p_1, p_2, ..., p_n \right).$$

Система (3.32) решается любым известным методом, например, методом исключения Гаусса. Если матрица Y окажется особенной (это происходит, если векторы $y^{(0)}, y^{(1)}, ..., y^{(n-1)}$ линейно-зависимые), то следует сменить начальный вектор $y^{(0)}$.

Чтобы не произошло переполнения в процессе вычисления векторов $y^{(k)}$, исходную матрицу целесообразно предварительно отнормировать: $A_{ij} = \dfrac{A_{ij}}{\max\limits_{i,j} \left| A_{ij} \right|}$. После этого собственные значения отнормированной и исходной матриц будут связаны соотношением $\lambda_{\text{новое}} = \dfrac{\lambda_{\text{старое}}}{\max\limits_{i,j} \left| A_{ij} \right|}$ (так как $(cA)x = (c\lambda)x$).

После определения собственных значений матрицы A ее собственные векторы можно искать в виде разложения по полной линейно-независимой системе векторов $y^{(0)}, y^{(1)}, ..., y^{(n-1)}$:

$$x = \beta_1 y^{(n-1)} + \beta_2 y^{(n-2)} + ... + \beta_n y^{(0)}. \qquad (3.33)$$

Для нахождения неизвестных коэффициентов β_i, подставим (3.33) в уравнение $Ax = \lambda x$. Тогда с учетом процедуры построения векторов $y^{(k)}$, получим

$$\beta_1 y^{(n)} + \beta_2 y^{(n-1)} + ... + \beta_n y^{(1)} = \lambda(\beta_1 y^{(n-1)} + \beta_2 y^{(n-2)} + ... + \beta_n y^{(0)}). \qquad (3.34)$$

Заменим в (3.34) $y^{(n)}$ выражением (3.31) и соберем подобные слагаемые

$$(\beta_2 - p_1\beta_1 - \lambda\beta_1) y^{(n-1)} + (\beta_3 - p_2\beta_1 - \lambda\beta_2) y^{(n-2)} + ...$$
$$+(\beta_n - p_{n-1}\beta_1 - \lambda\beta_{n-1}) y^{(1)} - (p_n\beta_1 + \lambda\beta_n) y^{(0)} = 0.$$

Поскольку векторы $y^{(0)}, y^{(1)}, ..., y^{(n-1)}$ – линейно-независимые, то коэффициенты в этом равенстве при всех $y^{(k)}$ должны быть равны нулю. Отсюда получаем

$$\begin{aligned}
&\beta_2 = (\lambda + p_1)\beta_1;\\
&\beta_3 = \lambda\beta_2 + p_2\beta_1 \equiv (\lambda^2 + p_1\lambda + p_2)\beta_1;\\
&.....\\
&\beta_n = \lambda\beta_{n-1} + p_{n-1}\beta_1 \equiv (\lambda^{n-1} + p_1\lambda^{n-2} + ... + p_{n-1})\beta_1;\\
&(\lambda^n + p_1\lambda^{n-1} + ... + p_n)\beta_1 = 0.
\end{aligned} \qquad (3.35)$$

Здесь в последнем соотношении в скобках стоит характеристический полином, который равен нулю. Поэтому, не нарушая это равенство, можно положить $\beta_1 = 1$. Тогда остальные неизвестные коэффициентов можно рассчитать с помощью рекуррентной формулы

$$\beta_k = \lambda\beta_{k-1} + p_{k-1}, \quad k = 2, n. \qquad (3.36)$$

Для кратных λ недостающие собственные векторы (если они существуют) следует искать, изменяя вектор $y^{(0)}$. В этом случае необходимо проверить, что полученные собственные вектора $x^{(i)}$ являются линейно-независимыми. Для этой цели можно использовать процедуру ортогонализации, которая проходит до конца только для линейно-независимых векторов.

На рис. 3.8 и 3.9 приведены скриншоты программы, которая использует метод Крылова для построения характеристического полинома для производной матрицы, введенной пользователем или случайным образом сформированной самой программой. После ввода матрицы можно построить «систему Крылова» (см. формулу (3.32)), а затем найти коэффициенты характеристического полинома исходной матрицы.

Система Крылова решается методом оптимального исключения Гаусса–Жордана (см. п. 2.2.2). Далее методом парабол (см. п. 4.3) можно определить все корни построенного полинома (собственные значения исходной матрицы). Вместе с корнями программа выдает значения следа матрицы ($\sum_{i=1}^{n} A_{ii}$) и сумму этих корней. Сравненительный анализ этих величин позволяет оценить корректность выполнения всех промежуточных этапов решения задачи нахождения собственных значений исходной матрицы.

В заключение приведем результаты (табл. 3.2) вычислительного эксперимента, целью которого была сравнительная косвенная оценка точности определения собственных значений случайной матрицы методом Крылова и Данилевского в зависимости от размерности матрицы. Они оказались вполне ожидаемыми, поскольку метод Крылова требует почти в 5 раз больше операций, чем метод Данилевского. Кроме того, в случае метода Крылова свои ошибки добовляет метод исключения, который используется при рассчете коэффициентов характеристического полинома матрицы.

Таблица 3.2

Косвенная оценка погрешности методов ($\left|\sum_{i=1}^{n} A_{ii} - \sum_{i=1}^{n} \lambda(A)_i\right|$) для случайных матриц, компонентами которых являлись псевдослучайные равномерно распределенные числа из интервала [-0,5; 0,5]

Размерность матрицы	3	5	7	10	15
Метод Данилевского	0	0	0	0	0
Метод Крылова	0	<=0.000001	<0,0000014	<0,000006	<0,000007

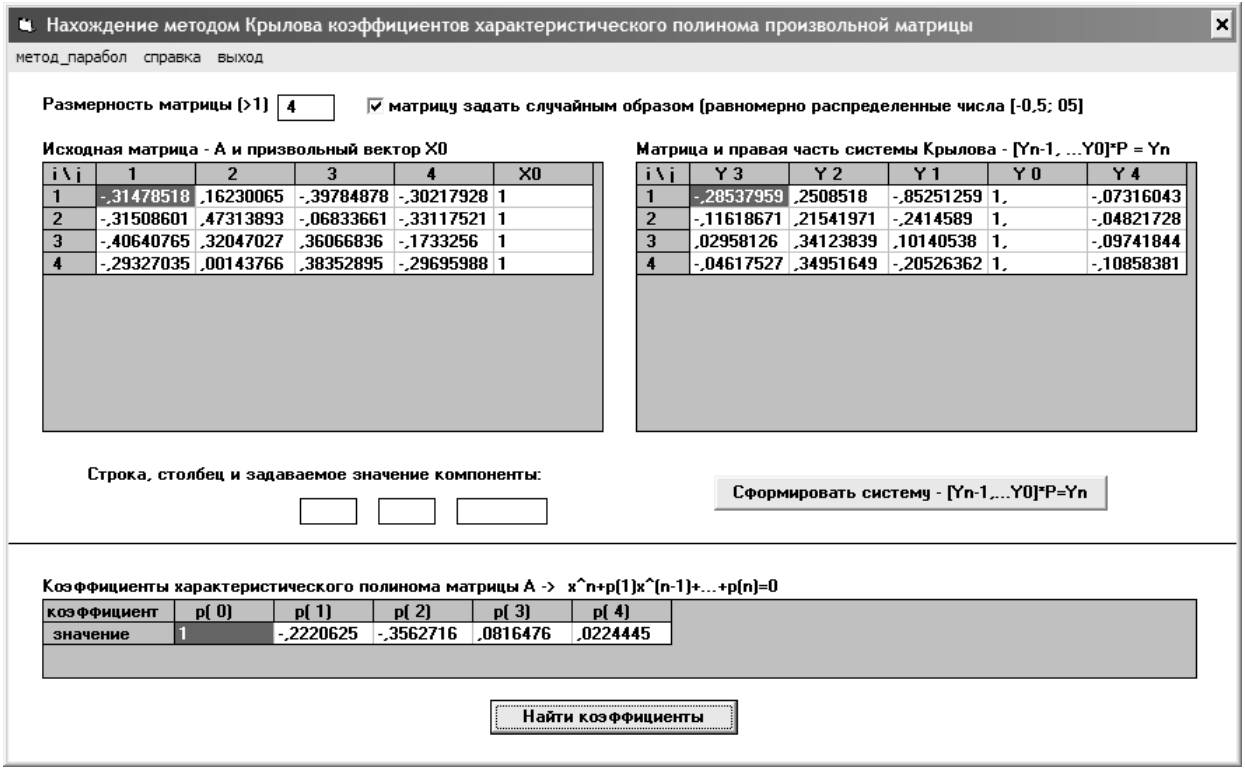


Рис. 3.8. Главная экранная форма программы для нахождения коэффициентов характеристического полинома произвольной матрицы методом Крылова

Рис. 3.9. Вспомогательная экранная форма программы, для определения корней характеристического полинома исходной матрицы. Форма вызывается пунктом меню «метод парабол» на главной форме рис. 3.1

3.5. Определение собственных векторов в общем случае

Способ 1. Собственный вектор для известного собственного значения λ должен удовлетворять системе уравнений

$$\begin{aligned} &(A_{11}-\lambda)x_1 + A_{12}x_2 + \ldots + A_{1n}x_n = 0;\\ &A_{21}x_1 + (A_{22}-\lambda)x_2 + \ldots + A_{2n}x_n = 0;\\ &\ldots\ldots\ldots\ldots\ldots\ldots\\ &A_{n1}x_1 + A_{n2}x_2 + \ldots + (A_{nn}-\lambda)x_n = 0. \end{aligned} \qquad (3.37)$$

Определитель этой системы равен нулю, т.е. уравнения линейно зависимы. Поэтому собственный вектор x можно найти следующим образом.

1. Поскольку собственный вектор x задается с точностью до множителя, то положим одну из компонент вектора x равной единице, например, $x_1 = 1$.

2. Отбросим одно из уравнений, так чтобы оставшиеся уравнения были линейно независимыми, например, последнее.

3. Далее решаем получившуюся неоднородную систему уравнений любым известным методом:

$$\begin{aligned} &A_{12}x_2 + \ldots \quad + A_{1n}x_n = (A_{11}-\lambda);\\ &(A_{22}-\lambda)x_2 + \ldots + A_{2n}x_n = A_{21};\\ &\ldots\ldots\ldots\ldots\ldots\ldots\\ &A_{n-12}x_2 + \ldots + (A_{n-1n}-\lambda)x_n = A_{n-11} \end{aligned} \qquad (3.38)$$

и находим оставшиеся компоненты собственного вектора.

Если решение системы невозможно, то следует сменить номер фиксируемой компоненты или отбросить другое уравнение в исходной системе. Также следует поступить при нахождении другого собственного вектора, если собственное значение кратно. Поскольку не всегда произвольная матрица имеет полную систему собственных векторов, то после нахождения нескольких вариантов собственных векторов необходимо проверить их линейную независимость, например, с помощью процедуры ортогонализации.

Способ 2. Поскольку собственные значения $\tilde{\lambda}_i$, как правило, (например, из-за ошибок округления), определяются с погрешностью, то определитель матрицы $(A-\tilde{\lambda}_i E)$ всегда будет отличен

от нуля. С учетом этого обстоятельства для нахождения соответствующего собственного вектора предложен метод обратной итерации: берется произвольный вектор $x^{(0)} = \dfrac{x^{(0)}}{\left\| x^{(0)} \right\|}$ и выполняется итерационный процесс для $k = 1, K$,

$$\{ (A - \tilde{\lambda}_i E) x^{(k)} = x^{(k-1)},\ x^{(k)} = \frac{x^{(k)}}{\left\| x^{(k)} \right\|} \}.$$

Для его сходимости обычно бывает достаточно 2–3 итераций.

Сходимость метода обосновывается следующим образом. Пусть матрица A имеет полную линейно-независимую систему собственных векторов $x^{(s)}$. Тогда векторы $x^{(k)}$, $x^{(k-1)}$ можно представить в виде разложений

$$x^{(k)} = \sum_s a_s^{(k)} x^{(s)}, \quad x^{(k-1)} = \sum_s a_s^{(k-1)} x^{(s)}.$$

Подставив эти разложения в систему, получим

$$(A - \tilde{\lambda}_i E) x^{(k)} = x^{(k-1)} \quad \Rightarrow \quad \sum_s [a_s^{(k)} (\lambda_s - \tilde{\lambda}_i) - a_s^{(k-1)}] x^{(s)} = 0. \qquad (3.39)$$

Отсюда, поскольку векторы $x^{(s)}$ линейно-независимые, следуют соотношения

$$a_s^{(k)} = \frac{a_s^{(k-1)}}{(\lambda_s - \tilde{\lambda}_i)}, \quad s = 1, n.$$

Но, так как $\tilde{\lambda}_i \approx \lambda_i$, то $a_i^{(k)} >> a_s^{(k)}$, $s \neq i$.

3.6. Метод вращения

Любая симметричная действительная матрица может быть приведена к диагональному виду с помощью подобного ортогонального преобразования

$$D = T^{-1} A T, \qquad (3.40)$$

где T – ортогональная матрица, удовлетворяющая условию $T^{-1} = T^T$.

Если такая матрица T найдена, то диагональные компоненты матрицы D совпадают с собственными значениями матрицы A, а

столбцы матрицы T – с ее соответствующими собственными векторами. Действительно, пусть $\lambda_k = D_{kk}$. Тогда очевидно, что

$$De^{(k)} = \lambda_k e^{(k)},$$

$$e^{(k)} = \left\{ e_i^{(k)} = \delta_{ik}, i = 1, n \right\}.$$

Отсюда следует

$$T^{-1} A T e^{(k)} = \lambda_k e^{(k)} \quad \Rightarrow \quad A T e^{(k)} = \lambda_k T e^{(k)}, \tag{3.41}$$

т.е. λ_k – собственное значение матрицы A, а $Te^{(k)} = \left\{ T_{ik}, i = 1, n \right\}$ – соответствующий ему собственный вектор.

В методе вращения матрица T находится с помощью серии подобных ортогональных преобразований, которые последовательно уменьшают сумму квадратов недиагональных элементов исходной матрицы A. Указанные преобразования выполняются с использованием матриц вращения (отличается от единичной матрицы только четырьмя компонентами $T_{ll}, T_{kk}, T_{lk}, T_{kl}$):

$$T(l,k,\varphi) = \begin{array}{c} \\ \end{array} \begin{array}{cccccccc} & l & & & & k & & \\ \end{array}$$

$$T(l,k,\varphi) = \begin{bmatrix} 1 & 0 & 0 & - & 0 & 0 & 0 & 0 \\ 0 & \cos\varphi & 0 & - & 0 & -\sin\varphi & 0 & 0 \\ 0 & 0 & 1 & - & 0 & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & - & 1 & 0 & 0 & 0 \\ 0 & \sin\varphi & 0 & - & 0 & \cos\varphi & 0 & 0 \\ 0 & 0 & 0 & - & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & - & 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{c} \\ ...l \\ \\ \\ \\ ...k \\ \\ \\ \end{array} \tag{3.42}$$

(столбцы l и k отмечены над матрицей)

Матрица $T(l,k,\varphi)$ называется матрицей вращения, поскольку с помощью такой матрицы осуществляется пересчет проекций произвольного вектора при повороте системы координат на угол φ (рис. 3.10)

$$x_1 = x\cos\varphi - y\sin\varphi;$$

$$y_1 = x\sin\varphi + y\cos\varphi.$$

Легко проверить, что для матрицы вращения выполняется условие ортогональности, например:

$$T^T T = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

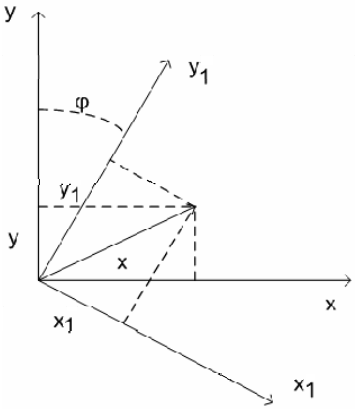


Рис. 3.10. Изменение проекций произвольного вектора x при повороте системы координат на угол φ

Распишем формулы, по которым текущие матрицы A и T (T первоначально задается в виде единичной матрицы) пересчитываются в процессе одного преобразования.

1. Первая фаза преобразования

$$\tilde{A} = AT(l,k,\varphi) = \begin{bmatrix} A_{11} & - & - & - & A_{1n} \\ - & A_{il} & - & A_{ik} & - \\ - & - & - & - & - \\ - & - & - & - & - \\ A_{n1} & - & - & - & A_{nn} \end{bmatrix} \begin{bmatrix} 1 & - & - & - & 0 \\ - & \cos\varphi & - & -\sin\varphi & - \\ - & - & 1 & - & 0 \\ - & \sin\varphi & - & \cos\varphi & - \\ 0 & - & - & - & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ ...l \\ \\ ...k \\ \end{matrix}$$

и аналогично $\tilde{T} = TT(l,k,\varphi)$ меняет компоненты текущих матриц A и T только в l-ом и k-ом столбцах ($i = 1,n$)

$$\begin{aligned} \tilde{A}_{il} &= A_{il}\cos\varphi + A_{ik}\sin\varphi; \\ \tilde{A}_{ik} &= -A_{il}\sin\varphi + A_{ik}\cos\varphi; \end{aligned} \tag{3.43}$$

$$\begin{aligned} \tilde{T}_{il} &= T_{il}\cos\varphi + T_{ik}\sin\varphi; \\ \tilde{T}_{ik} &= -T_{il}\sin\varphi + T_{ik}\cos\varphi. \end{aligned} \tag{3.44}$$

2. Вторая фаза преобразования

$$\tilde{A} = T^T(l,k,\varphi)\tilde{A} = \begin{bmatrix} 1 & - & - & - & 0 \\ - & \cos\varphi & - & -\sin\varphi & - \\ - & - & 1 & - & 0 \\ - & \sin\varphi & - & \cos\varphi & - \\ 0 & - & - & - & 1 \end{bmatrix} \begin{bmatrix} \tilde{A}_{11} & - & - & - & \tilde{A}_{1n} \\ - & \tilde{A}_{ij} & - & - & - \\ - & - & - & - & - \\ - & \tilde{A}_{kj} & - & - & - \\ \tilde{A}_{n1} & - & - & - & \tilde{A}_{nn} \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \ldots l \\ \\ \ldots k \\ \end{matrix}$$

меняет компоненты текущей матрицы $\tilde{A}$ только в *l*-й и *k*-й строках ($j = 1, n$):

$$\begin{aligned} \tilde{\tilde{A}}_{lj} &= \tilde{A}_{lj}\cos\varphi + \tilde{A}_{kj}\sin\varphi; \\ \tilde{\tilde{A}}_{kj} &= -\tilde{A}_{lj}\sin\varphi + \tilde{A}_{kj}\cos\varphi. \end{aligned} \tag{3.45}$$

С помощью формул (3.43) и (3.45) легко получить следующие соотношения.

1. Для $i \neq l, k$

$$\begin{aligned} &\tilde{A}_{il}^2 + \tilde{A}_{ik}^2 = A_{il}^2\cos^2\varphi + A_{ik}^2\sin^2\varphi + 2A_{il}A_{ik}\sin\varphi\cos\varphi + \\ &+A_{il}^2\sin^2\varphi + A_{ik}^2\cos^2\varphi A - 2A_{il}A_{ik}\sin\varphi\cos\varphi = A_{il}^2 + A_{ik}^2 \end{aligned} \tag{3.46}$$

и аналогично для $j \neq l, k$

$$\tilde{\tilde{A}}_{lj}^2 + \tilde{\tilde{A}}_{kj}^2 = A_{lj}^2 + A_{kj}^2. \tag{3.47}$$

2. $$\begin{aligned} \tilde{\tilde{A}}_{lk} &= \tilde{A}_{lk}\cos\varphi + \tilde{A}_{kk}\sin\varphi = \\ &= -A_{ll}\sin\varphi\cos\varphi + A_{lk}\cos^2\varphi - A_{kl}\sin^2\varphi + A_{kk}\sin\varphi\cos\varphi = \\ &= A_{lk}(\cos^2\varphi - \sin^2\varphi) + (A_{kk} - A_{ll})\sin\varphi\cos\varphi = \\ &= A_{lk}\cos 2\varphi - \frac{1}{2}(A_{ll} - A_{kk})\sin 2\varphi. \end{aligned} \tag{3.48}$$

С учетом формул (3.47) и (3.48) сумму квадратов недиагональных элементов матрицы после текущего подобного преобразования можно представить как

$$\sum_{i \neq j} \tilde{\tilde{A}}_{ij}^2 = \sum_{i \neq j} A_{ij}^2 - 2A_{lk}^2 + \frac{1}{2}[2A_{lk}\cos 2\varphi - (A_{ll} - A_{kk})\sin 2\varphi]^2. \tag{3.49}$$

Отсюда легко выводятся условия, обеспечивающие на каждой итерации уменьшение суммы квадратов недиагональных элементов:

1) выбор параметров l и k:

$$A_{lk} = \max_{i \neq j} \left| A_{ij} \right|; \tag{3.50}$$

2) выбор параметра φ

$$\varphi = \frac{1}{2} \operatorname{arc\,tg}\left(\frac{2A_{lk}}{A_{ll} - A_{kk}}\right); \tag{3.51}$$

Если $A_{ll} = A_{kk}$, то $|\varphi| = \frac{\pi}{4}$ и $\sin\varphi = \cos\varphi = \frac{\sqrt{2}}{2}$.

Замечание. Для расчета $\sin\varphi, \cos\varphi$ можно (но не обязательно) использовать формулы, не требующие вычисления угла φ:

$$\sin\varphi = sign(p)\sqrt{\frac{1}{2}\left(1 - \frac{1}{\sqrt{(1+p^2)}}\right)}, \quad \cos\varphi = \sqrt{(1-\sin^2\varphi)},$$

где $p = \frac{2A_{lk}}{(A_{ll} - A_{kk})}$.

Алгоритм метода вращения схематично можно представить в виде

for it $= 1, Iter$
{
находим A_{lk}
if $|A_{lk}| < eps$ *then* "*выход из цикла*"
вычисление $\sin\varphi, \cos\varphi$
for $i = 1, n$
{
вычисляем по формулам (3.43), (3.44)
}
for $j = 1, n$
{
вычисляем по формулам (3.45)
}
}

Здесь *Iter* – максимальное число итераций – преобразований, *esp* – допустимое значение максимального по модулю недиагонального элемента. Результаты вычислений по первым формулам в (3.43)–(3.45) на каждом шаге вначале следует сохранять в буферной переменной и лишь затем переносить в соответствующее место исходной матрицы и формируемой матрицы вращения.

Поиск максимального по модулю недиагонального элемента можно выполнить следующим образом

$$\max = -1, l = 0, k = 0$$
$$for\ i = 1,(n-1)$$
$$\{$$
$$for\ j = i+1, n$$
$$\{$$
$$buf = \left|A_{ij}\right|$$
$$if\ buf > \max\ then\ \{\max = buf, l = i, k = j\}$$
$$\}$$
$$\}$$

Метод устойчив к ошибкам округления и обладает асимптотической скоростью сходимости. Действительно, из соотношения

$$\sum_{i\neq j}\tilde{\tilde{A}}_{ij}^2 = \sum_{i\neq j}A_{ij}^2 - 2A_{lk}^2$$

имеем (так как $\sum_{i\neq j}A_{ij}^2 \le n(n-1)A_{lk}^2 \Rightarrow A_{lk}^2 \ge \frac{\sum_{i\neq j}A_{ij}^2}{n(n-1)}$)

$$\sum_{i\neq j}\tilde{\tilde{A}}_{ij}^2 \le \sum_{i\neq j}A_{ij}^2[1-\frac{2}{n(n-1)}] \equiv q\sum_{i\neq j}A_{ij}^2$$

и, следовательно, после m-й итерации

$$(\sum_{i\neq j}\tilde{\tilde{A}}_{ij}^2)_{\text{текущая}} \le q^m(\sum_{i\neq j}A_{ij}^2)_{\text{исходная}}. \qquad (3.52)$$

Ниже в табл. 3.3 приведены значения параметра $q = [1-\frac{2}{n(n-1)}]$ для различных значений размерности n исходной матрицы.

Таблица 3.3

Зависимость параметра *q* от размерности матрицы *n*

n	2	3	4	5	10	20
q	0	0,67	0,83	0,9	0,98	0,995

Из таблицы видно, что с ростом n скорость сходимости метода быстро падает.

На рис. 3.11 и 3.12 приведены экраны программы, реализующей метод вращения (на начальной и конечной стадиях). Использована случайная матрица, компонентами которой являются псевдо-случайные числа, равномерно распределенные на отрезке [0,1]. Видно, что после 90 итераций выполняется известный инвариант – сумма найденных собственных значений (они расположены на диагонале преобразованной матрицы) равна сумме диагональных элементов исходной матрицы, т.е. следу матрицы. Кроме того, проведено дополнительное тестирование надежности полученного решения (пункт меню «дополнительное тестирование»). Его результаты представлены на рис. 3.13 и 3.14.

С помощью этой программы исследована зависимость среднего числа итераций, требуемых для нахождения с заданной точностью собственных значений случайной матрицы, от ее размерности. Результаты исследования представлены в табл. 3.4 и в графическом виде на рис. 3.15.

Таблица 3.4

Экспериментальная зависимость среднего числа требуемых итераций N в методе вращения от размерности n случайной матрицы.

n	2	4	6	8	10	12
N	1	15	46	84	143	211

Вопросы и задания для самоконтроля

1. В чем смысл метода Данилевского?
2. Составьте алгоритм процедуры выбора ведущего элемента для метода Данилевского.
3. О чем сигнализирует нулевой ведущий элемент в методе Данилевского?
4. Каким образом в общем случае можно найти собственный вектор для заданного собственного значения?
5. В чем смысл метода Крылова?
6. В чем смысл метода вращения?
7. Каким образом выбирается текущая матрица вращения в методе вращения?

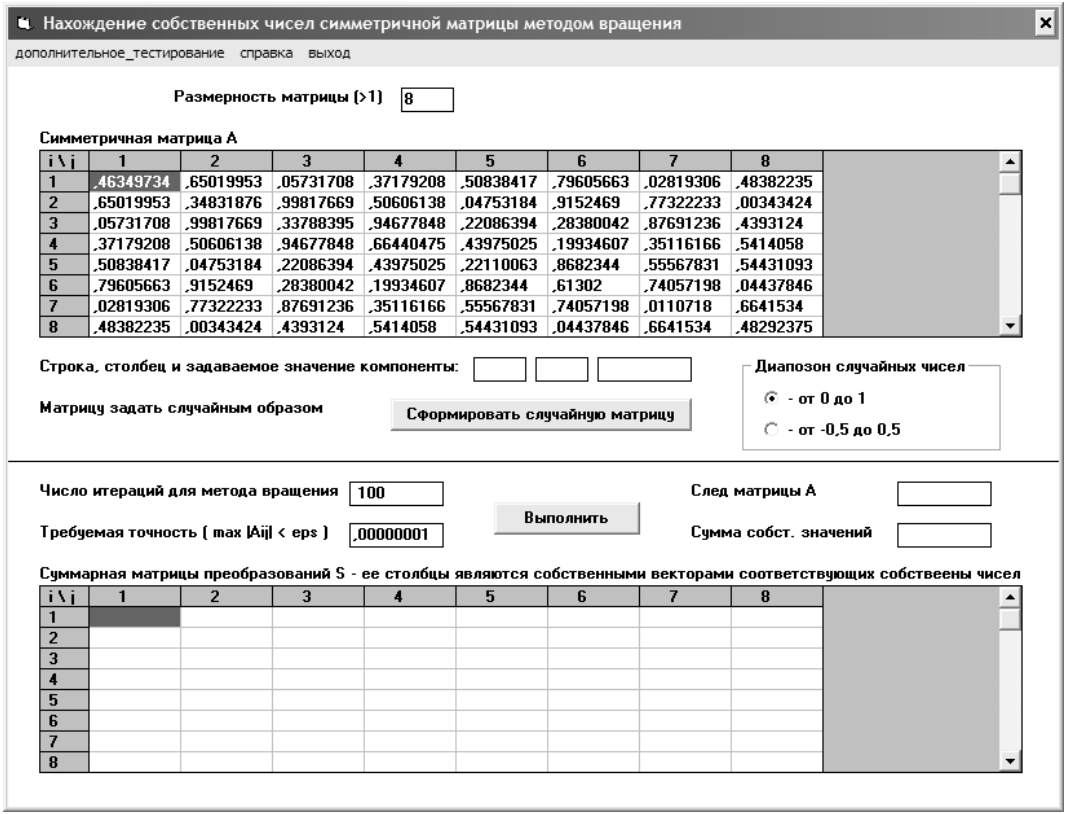
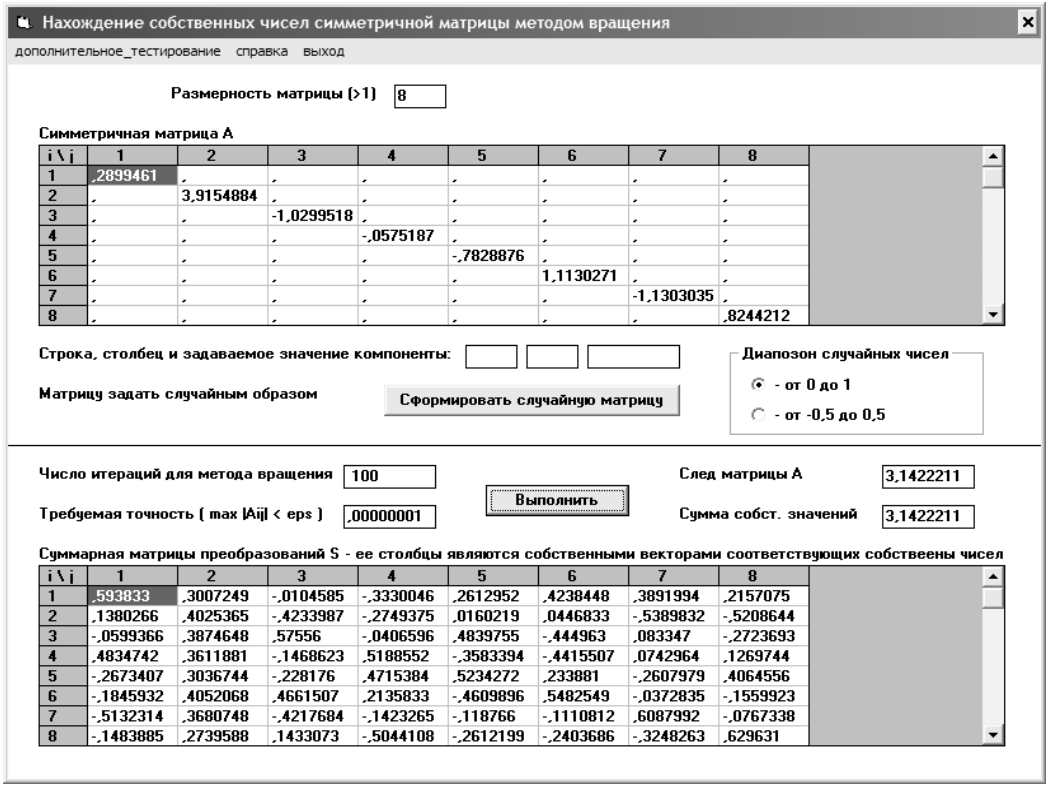


Рис. 3.11. Задание исходной случайной матрицы



i \ j	1	2	3	4	5	6	7	8
1	,2899461	,	,	,	,	,	,	,
2	,	3,9154884	,	,	,	,	,	,
3	,	,	-1,0299518	,	,	,	,	,
4	,	,	,	-,0575187	,	,	,	,
5	,	,	,	,	-,7828876	,	,	,
6	,	,	,	,	,	1,1130271	,	,
7	,	,	,	,	,	,	-1,1303035	,
8	,	,	,	,	,	,	,	,8244212

i \ j	1	2	3	4	5	6	7	8
1	,593833	,3007249	-,0104585	-,3330046	,2612952	,4238448	,3891994	,2157075
2	,1380266	,4025365	-,4233987	-,2749375	,0160219	,0446833	-,5389832	-,5208644
3	-,0599366	,3874648	,57556	-,0406596	,4839755	-,444963	,083347	-,2723693
4	,4834742	,3611881	-,1468623	,5188552	-,3583394	-,4415507	,0742964	,1269744
5	-,2673407	,3036744	-,228176	,4715384	,5234272	,233881	-,2607979	,4064556
6	-,1845932	,4052068	,4661507	,2135833	-,4609896	,5482549	-,0372835	-,1559923
7	-,5132314	,3680748	-,4217684	-,1423265	-,118766	-,1110812	,6087992	-,0767338
8	-,1483885	,2739588	,1433073	-,5044108	-,2612199	-,2403686	-,3248263	,629631

Рис. 3.12. Собственные значения исходной, случайной матрицы, найденные методом вращения. Видно, что сумма собственныз значений равна следу матрицы

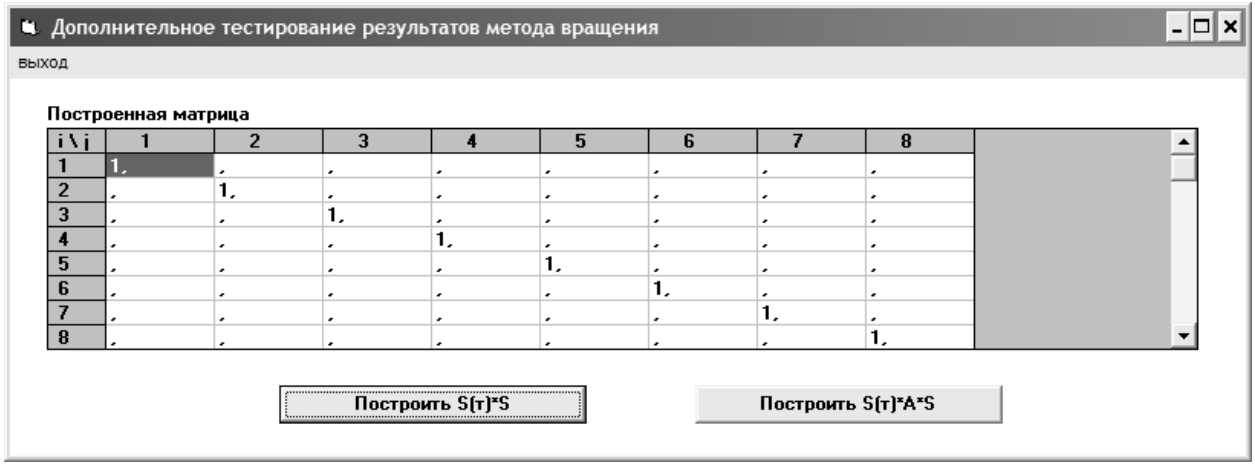


Рис. 3.13. Видно, что суммарная матрица преобразования *S*, полученная итерационным методом вращения, действительно является ортогональной

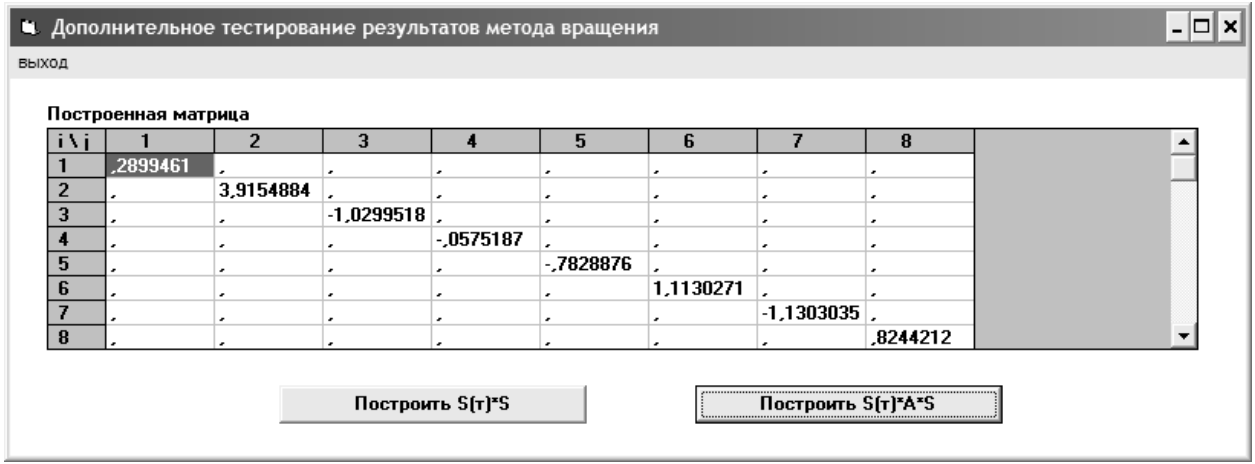


Рис. 3.14. Видно, подобное преобразование исходной матрицы, выполненное с помощью суммарной матрицы *S*, дает те же результаты, что и итерационная процедура метода вращения (см. рис. 3.12)

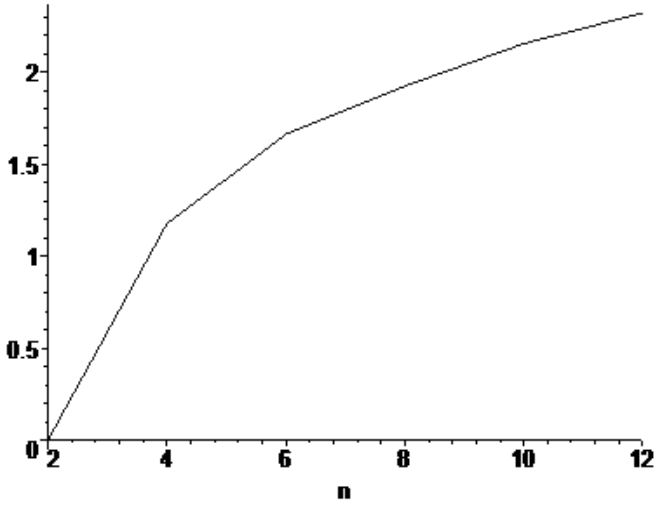


Рис. 3.15. Графическое представление результатов таблицы 3.3 ($\log(N)$ от n)

Глава 4. АЛГОРИТМЫ ДЛЯ РАБОТЫ С ПОЛИНОМАМИ

4.1. Подсчет значения полинома и деление на множители

1. Подсчет значения полинома $P_n(x)$ в точке x (схема Горнера). Если полином представить в виде

$$P_n(x) = \sum_{i=0}^{n} a_i x^{n-i} = a_0 x^n + a_1 x^{n-1} + \ldots + a_n = \\ = (\ldots(a_0 x + a_1)x + a_2)x + \ldots + a_{n-1})x + a_n), \tag{4.1}$$

то для подсчета значения полинома $P_n(x)$ в точке x легко записать наиболее экономичный (по количеству операций) алгоритм

$$P = a_0; \\ for\ i = 1, n\ \{P = Px + a_i\}. \tag{4.2}$$

Этот алгоритм легко обобщается на комплексный случай $x = \alpha + i\beta$:

$$\begin{array}{l} P1 = a_0 \\ P2 = 0 \\ for\ i = 1, n \\ \{ \\ \quad R1 = P1 \\ \quad P1 = \alpha P1 - \beta P2 + a_i \\ \quad P2 = \alpha P2 + \beta R1 \\ \} \end{array} \tag{4.3}$$

Здесь использовано представление $P(\alpha + i\beta) = P1 + iP2$.

2. Деление полинома $P_n(x)$ на линейный множитель $(x - \alpha)$. Запишем очевидное соотношение между исходным полиномом и результатом его деления на линейный множитель

$$P_n(x) = \sum_{i=0}^{n} a_i x^{n-i} = a_0 x^n + a_1 x^{n-1} + \ldots + a_n = \\ = (x - \alpha)(b_0 x^{n-1} + b_1 x^{n-2} + \ldots + b_{n-1}) + b_n = \\ = b_0 x^n + b_1 x^{n-1} + \ldots + b_{n-1} x + b_n - b_0 \alpha x^{n-1} - b_1 \alpha x^{n-2} - \ldots - b_{n-1}\alpha. \tag{4.4}$$

Отсюда, сравнивая сомножители при одинаковых степенях x в крайних частях равенства, получим рекуррентные формулы для определения коэффициентов b_i:

$$
\begin{aligned}
&b_0 = a_0;\\
&b_1 = a_1 + b_0\alpha;\\
&b_2 = a_2 + b_1\alpha;\\
&\ldots\ldots\\
&b_n = a_n + b_{n-1}\alpha = P_n(\alpha) \ - \textit{остаток}.
\end{aligned}
\tag{4.5}
$$

Остаток – коэффициент b_n равен нулю, если α – корень полинома (см. (4.4)).

Приведем алгоритм расчета коэффициентов b_i, который в общих чертах совпадает с алгоритмом (4.2)

$$
\begin{aligned}
&b_0 = a_0\\
&for\ i = 1, n\ \{b_i = b_{i-1}\alpha + a_i\}
\end{aligned}
\tag{4.6}
$$

3. Деление полинома $P_n(x)$ на квадратичный множитель $(x^2 + px + q)$. Аналогично предыдущему случаю имеем

$$
\begin{aligned}
&P_n(x) = \sum_{i=0}^{n} a_i x^{n-i} = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1}x + a_n =\\
&= (x^2 + px + q)(b_0 x^{n-2} + b_1 x^{n-3} + b_2 x^{n-4} + \ldots + b_{n-3}x + b_{n-2}) + b_{n-1}x + b_n =\\
&= b_0 x^n + b_1 x^{n-1} + b_2 x^{n-2} + \ldots + b_{n-3}x^3 + b_{n-2}x^2 +\\
&+ b_0 p x^{n-1} + b_1 p x^{n-2} + b_2 p x^{n-3} + \ldots + + b_{n-3} p x^2 + b_{n-2} p x +\\
&+ b_0 q x^{n-2} + b_1 q x^{n-3} + b_2 q x^{n-4} + \ldots + + b_{n-3} q x + b_{n-2} q + b_{n-1}x + b_n.
\end{aligned}
\tag{4.7}
$$

Отсюда получаем рекуррентные формулы для определения коэффициентов b_i

$$
\begin{aligned}
&b_0 = a_0;\\
&b_1 = a_1 - b_0 p;\\
&b_2 = a_2 - b_1 p - b_0 q;\\
&\ldots\ldots\\
&b_{n-1} = a_{n-1} - b_{n-2} p - b_{n-3} q;\\
&b_n = a_n - b_{n-2} q
\end{aligned}
\tag{4.8}
$$

и алгоритм для их последовательного вычисления

$$
\begin{aligned}
&b_0 = a_0;\\
&b_1 = a_1 - b_0 p;\\
&for\ i = 2, (n-1)\ \{b_i = a_i - b_{i-1} p - b_{i-2} q\};\\
&b_n = a_n - b_{n-2} q.
\end{aligned}
\tag{4.9}
$$

Если $b_{n-1} = b_n = 0$, то полином $P_n(x)$ делится на квадратичный множитель $(x^2 + px + q)$ нацело.

Замечание. При выделении из полинома пары комплексно сопряженных корней $\alpha \pm i\beta$ коэффициенты соответствующего квадратичного множителя равны $p = -2\alpha,\ q = (\alpha^2 + \beta^2)$.

4.2. Локализация корней полинома

Приведем некоторые сведения, касающиеся корней полинома $P_n(x)$ с действительными коэффициентами:

- все комплексные корни полинома попарно комплексно сопряженные;
- полином нечетной степени имеет, по крайней мере, один действительный корень.

Укажем несколько теорем, позволяющих грубо оценивать местоположение корней на комплексной плоскости:

- все корни полинома лежат внутри круга радиуса $R = 1 + \dfrac{A}{|a_0|}$,

где $A = \max\limits_{i=1,n} |a_i|$;

- все корни полинома лежат за пределами круга радиуса $r = \dfrac{1}{1 + \dfrac{B}{|a_n|}}$, где $B = \max\limits_{i=0,n-1} |a_i|$ и $a_n \neq 0$ – иначе полином имеет нулевой корень и его порядок можно понизить;
- число корней полинома внутри замкнутой фигуры, в частном случае, круга, которую пробегает на комплексной плоскости против часовой стрелки аргумент x, равно количеству полных оборотов вокруг начала координат соответствующей замкнутой кривой $P_n(x)$, являющейся отображением этой фигуры.

Опираясь на эти результаты, можно предложить последовательность операций по локализации корней полинома (для выполнения этих операций разработана программа, экранные формы которой представлены на рис. 4.1–4.8).

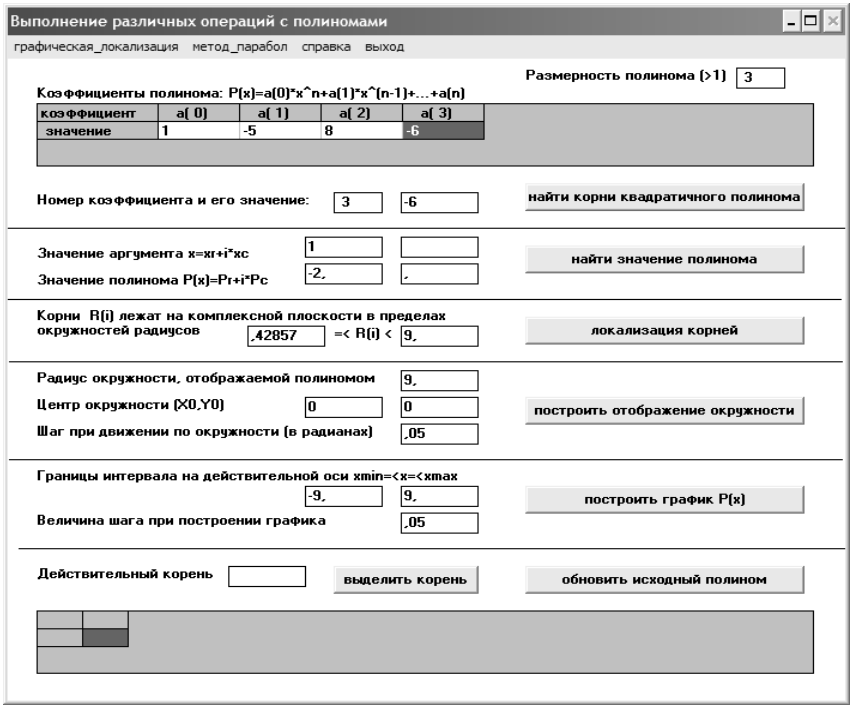


Рис. 4.1. Главная экранная форма программы, предназначенная для выполнения различных операций с заданным полиномом $P_3(x)$ с целью локализации его корней

1. Находим радиусы r, R окружностей, между которыми на комплексной плоскости расположены все корни заданного полинома (управляющая кнопка «локализация корней» на рис. 4.1).

2. Нажатием кнопки «построить отображение окружности», строим отображение полиномом $P_3(x)$ окружности (радиус – 9, расположение центра окружности – 0,0). Отображение показано на рис. 4.2. Видно, что кривая три раза оборачивается вокруг центра координат. Это подтверждает правильность грубой оценки (см. рис. 4.1.).

Подобным образом, меняя центр и радиус окружности, можно протестировать любую область комплексной плоскости на наличие в ней корней полинома.

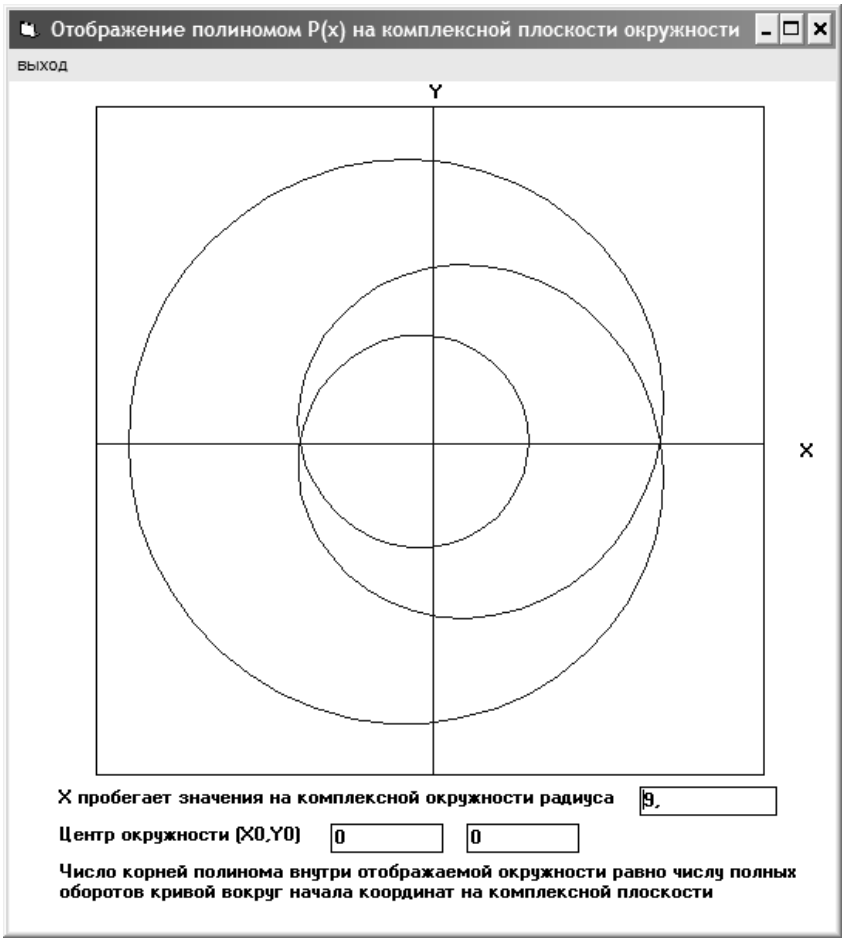


Рис. 4.2. Экранная форма с отображением окружности, построенным согласно заданному полиному $P_3(x)$. Три полных оборота вокруг начала координат, которое делает построенное отображение, указывают на наличие трех корней внутри области комплексной плоскости, ограниченной центральной окружностью радиуса = 9

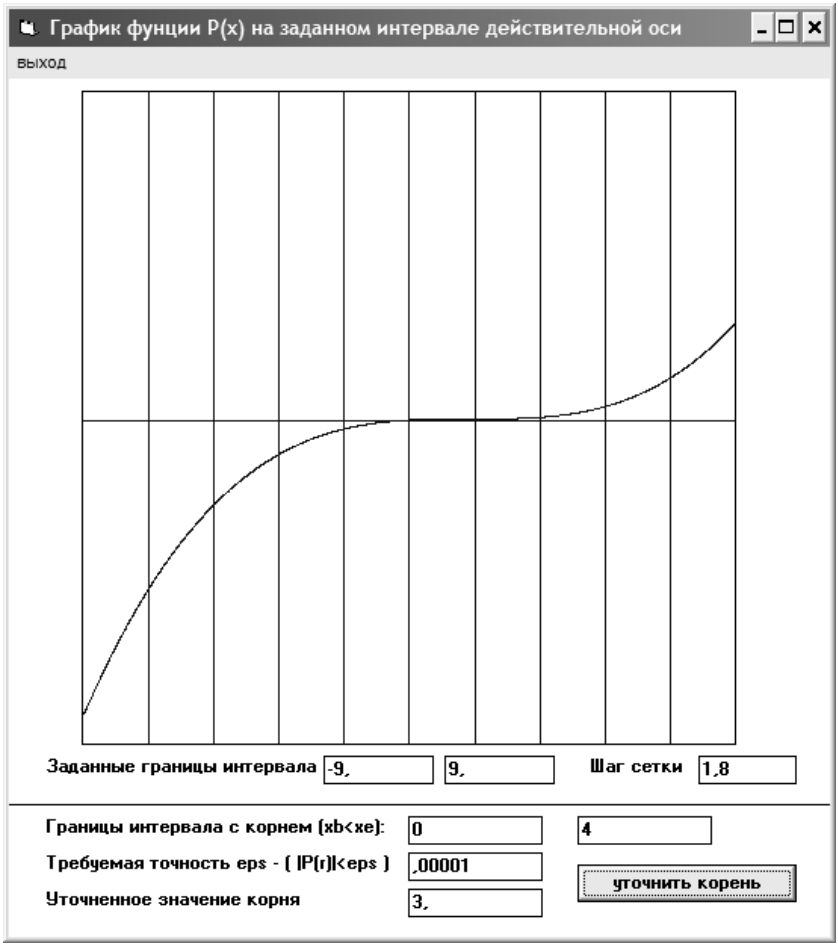


Рис. 4.3. График изменения $P_3(x)$ для заданного интервала. Здесь же приведено значение действительного корня, уточненное методом дихотомии

3. Локализуем положение действительного корня полинома, строя кривую $P_3(x)$ для аргумента, изменяющегося в интервале $-9 \le x \le 9$ (кнопка «построить график $P_n(x)$). Построенная кривая приведена на рис. 4.3. Здесь же приведено значение действительного корня, уточненное методом дихотомии, алгоритм

которого записывается следующим образом (здесь *xb*, *xe* – левая и правая границы интервала, содержащего один корень, *eps* – точность, *root* – найденное значение корня):

$$
\begin{aligned}
&pb = P(xb),\ pe = P(xe),\ pm = 1000\\
&do\ while\ |pm| > eps\\
&\quad xm = \frac{xb + xe}{2},\ pm = P(xm)\\
&\quad if\ sign(pb) = sign(pm)\ then\\
&\qquad pb = pm,\ xb = xm\\
&\quad else\\
&\qquad pe = pm,\ xe = xm\\
&\quad end\ if\\
&loop\\
&root = xm
\end{aligned}
$$

В данном алгоритме отрезок последовательно делится пополам и за новый отрезок выбирается та его половина, на концах которой значение полином имеет разные знаки. Выход из процесса деления происходит, когда значение полинома в средней точке текущего отрезка оказывается меньше заданной точности.

4. После нахождения уточненного значения корня, выделяем его из текущего полинома и тем самым понижаем его порядок. Коэффициенты нового полинома, полученного после выделения действительного корня из предыдущего полинома, приведены в нижней таблице рис. 4.4. Используя кнопку «обновить исходный полином», коэффициенты нового полинома ставим в верхнюю таблицу на место исходного полинома (см. рис. 4.5).

5. Новый полином имеет второй порядок, поэтому его корни определяются по известным формулам – кнопка «найти корни квадратичного полинома». Экранная форма со значениями этих корней приведена на рис. 4.6.

6. С помощью кнопки «*найти значение полинома*» – можно рассчитать значение полинома для найденного корня и убедится в правильности полученных его действительных и мнимых значений.

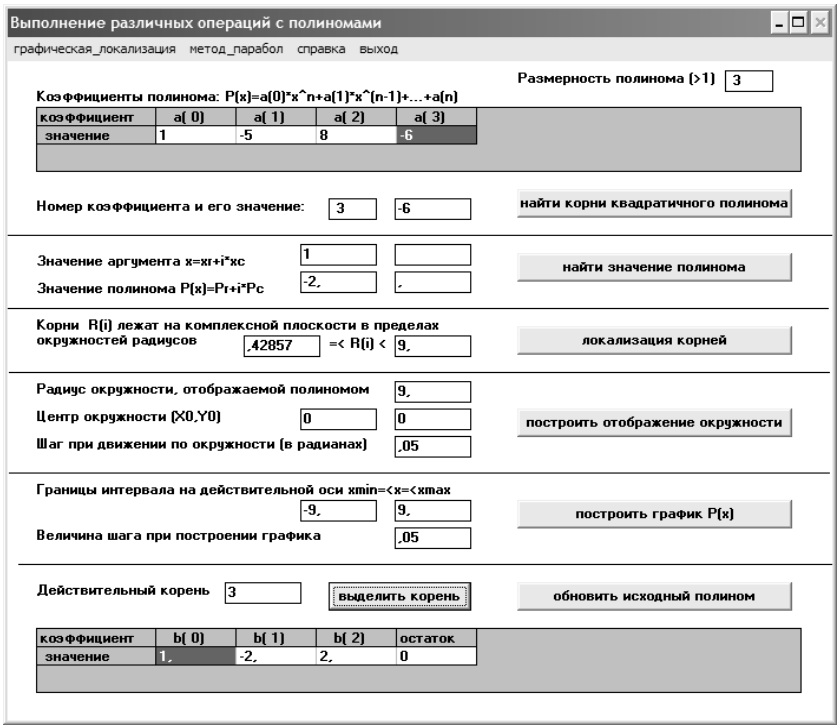


Рис. 4.4. В нижней таблице приведены коэффициенты нового полинома, полученного после выделения действительного корня из предыдущего полинома

7. Пункт меню «*графическая локализация*», расположенный на главной форме, загружает форму, приведенную на рис. 4.7. В рамках этой формы можно методом перебора найти местоположение точек на комплексной плоскости, в которых значение модуля полинома меньше заданного предельного значения: $\left|P(x+iy)\right| = \left|P1+iP2\right| = \sqrt{(P1\cdot P1 + P2\cdot P2)} < eps$. И таким образом, произвести графическую локализацию местоположения корней полинома на комплексной плоскости. Затем значение каждого локализованного корня можно уточнить, минимизирую выпуклый квадратичный функционал $F(x,y) = \left|P(x+iy)\right| = \left|P1+iP2\right| = \sqrt{(P1\cdot P1 + P2\cdot P2)}$ методами наискорейшего или покоординатного спуска.

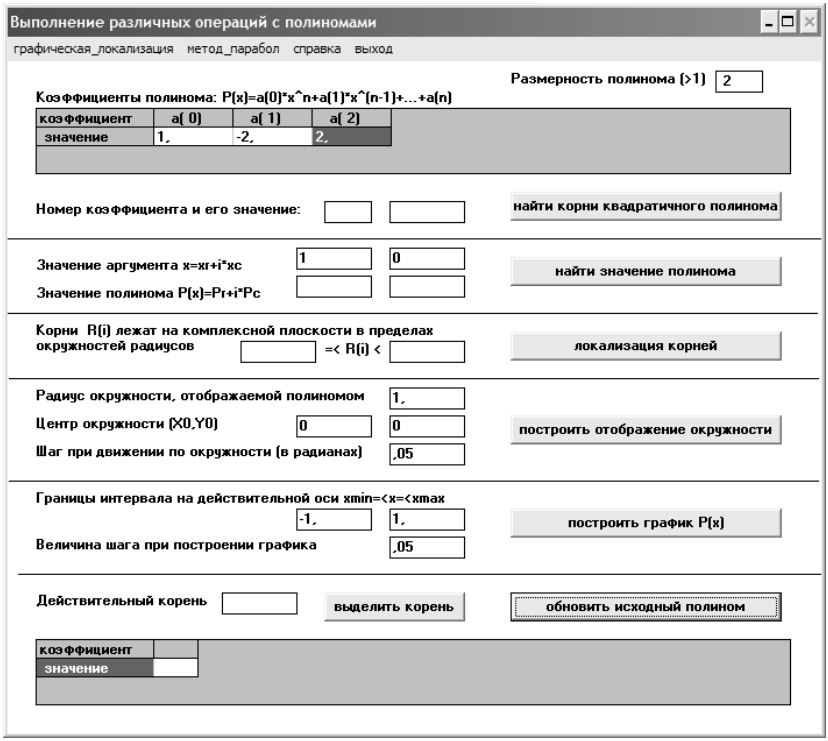


Рис. 4.5. Произведен перенос коэффициентов нового полинома на место исходного. Порядок анализируемого полинома понижен на единицу. Получена оценка местоположения его корней

Согласно первому методу спуск к нулевому значению функционала осуществляется против направления, заданного ортами вектора градиента функционала в текущей начальной точке комплексной плоскости. В программе используются два алгоритма наискорейшего спуска. В первом алгоритме одномерный спуск вдоль выбранного направления аналогичен затухающим колебаниям шарика при скатывании на дно «лунки», во втором – при переходе через минимум производится отступ к предыдущей точке, а затем выполняется новый уменьшенный в три раза шаг. При покоординатном спуске в пределах одной итерации производится одномерный спуск вначале по координате x, а затем y. В этом случае стратегия одномерного спуска аналогична затухающим колебаниям шарика при скатывании на дно «лунки».

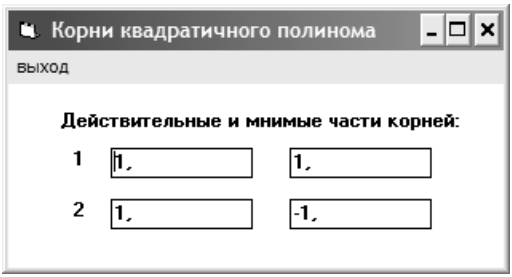


Рис. 4.6. Экранная форма с комплексными корнями квадратичного полинома

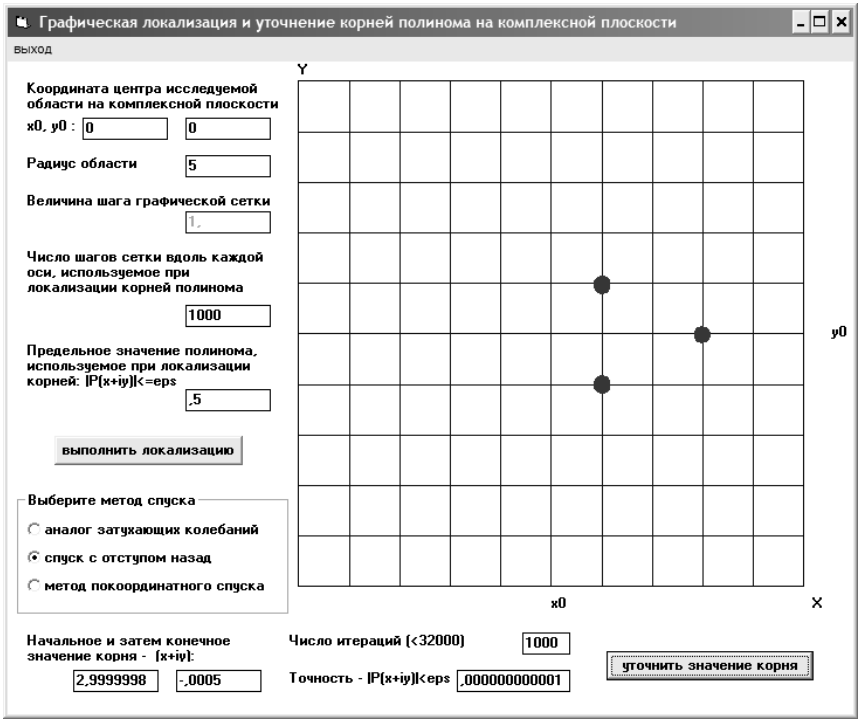


Рис. 4.7. Форма программы для локализации корней на комплексной плоскости и последующего уточнения локализованных корней. Здесь показан результат графической локализация методом перебора всех корней исходного полинома. Затем методом наискорейшего спуска произведено уточнение действительного корня (его точное значение равно 3). При этом использован алгоритм одномерного спуска с отступом назад

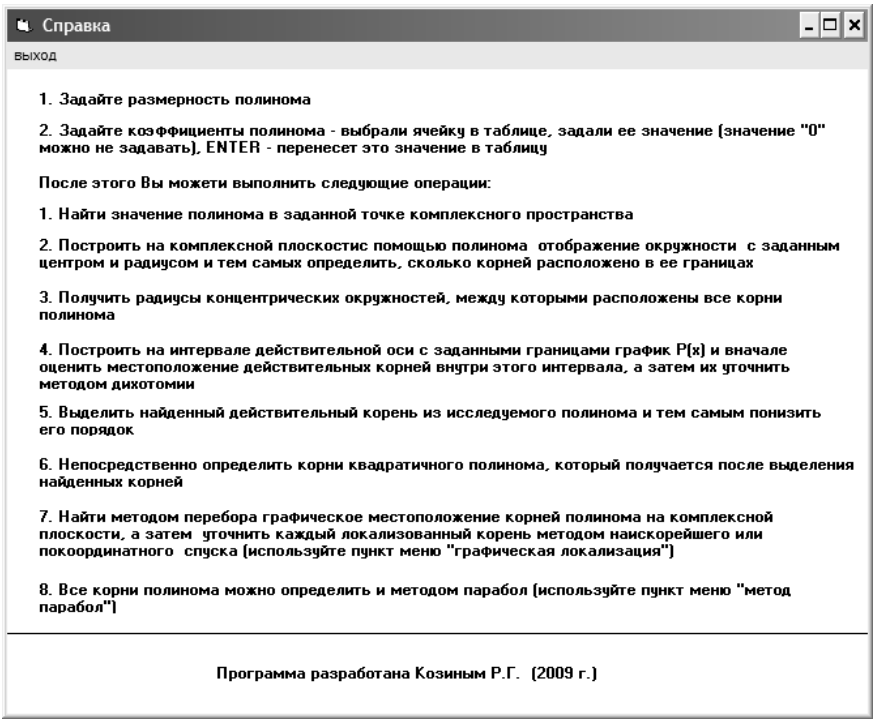


Рис. 4.8. Экранная форма с описанием порядка работы с программой.

Ниже приведены соответствующие алгоритмы:

1. Алгоритмы метода наискорейшего спуска для двух стратегий одномерного спуска

Алгоритм 1

$it = 0$ – *число использованных итераций*
do while $it < it\max$
$F\min = F(rx, ry)$
$h = 0,001$
$x = rx + h$
$y = ry$
$Fx = F(x, y)$
$x = rx$
$y = ry + h$
$Fy = F(x, y)$

$$\frac{dF}{dx} = \frac{Fx - F\min}{h}$$

$$\frac{dF}{dy} = \frac{Fy - F\min}{h}$$

$$g \equiv \left|grad(F)\right| = \sqrt{(\frac{dF}{dx})^2 + (\frac{dF}{dy})^2}$$

$$dx = \frac{dF}{dx} / g, \quad dy = \frac{dF}{dy} / g$$

$h = 0,01$

do while $h > 0,00000001$

$rx = rx - h * dx$

$ry = ry - h * dy$

$Ft = F(rx, ry)$

if $Ft < F\min$ *then* $F\min = Ft$ *else* $h = -h / 3$

loop

if $F\min <$ *eps then exit do*

$it = it + 1$

loop

Алгоритм 2

$it = 0$ – *число использованных итераций*

do while $it <$ *it* max

$F\min = F(rx, ry)$

$h = 0,001$

$x = rx + h$

$y = ry$

$Fx = F(x, y)$

$x = rx$

$y = ry + h$

$Fy = F(x, y)$

$$\frac{dF}{dx} = \frac{Fx - F\min}{h}$$

$$\frac{dF}{dy} = \frac{Fy - F\min}{h}$$

$$g \equiv \left|grad(F)\right| = \sqrt{(\frac{dF}{dx})^2 + (\frac{dF}{dy})^2}$$

$dx = \frac{dF}{dx} / g,\ \ dy = \frac{dF}{dy} / g$
$h = 0,01$
do while $h > 0,00000001$
$x = rx - h * dx$
$y = ry - h * dy$
$Ft = F(x, y)$
if $Ft < F\min$ *then* $rx = x, ry = y, F\min = Ft$ *else* $h = h / 3$
loop
if $F\min < eps$ *then exit do*
$it = it + 1$
loop

2. Алгоритм метода покоординатного спуска

$it = 0$ – *число использованных итераций*
do while $it < it\max$
for $i = 0$ *to* 1
'*определение направления спуска для данной координаты*
$h = 0,001$
$s = 1$
$F\min = F(rx, ry)$
a:
$x = rx + h * s * (1 - i)$
$y = ry + h * s * i$
$Ft = F(x, y)$
if $Ft < F\min$ *then goto b*
if $s = -1$ *then goto c else* $s = -1$, *goto a*
b:
$h = 0,01 * s$
'*одномерный спуск вдоль данной координаты*
do while $h > 0,00000001$
$rx = rx + h * (1 - i)$
$ry = ry + h * i$
$Ft = F(rx, ry)$
if $Ft < F\min$ *then* $F\min = Ft$ *else* $h = -h / 3$
loop
c:
if $F\min < eps$ *then exit do*
$it = it + 1$
loop

Здесь $it\max$ – максимальное число допустимых итераций (спусков), h – параметр спуска, *eps* – требуемая точность.

4.3. Метод парабол для нахождения всех корней полинома

Метод парабол является наиболее надежным алгоритмом поиска всех корней полинома. Метод не требует задания приближенных значений корней и определяет их последовательно один за другим. Сходимость метода теоретически не доказана, но не известно ни одного случая, когда бы метод не сходился.

Для запуска метода произвольно выбираются три начальные точки на комплексной плоскости $z = x + iy \equiv z(x, y)$, обычно $z_0 = (-1,0)$, $z_1 = (1,0)$, $z_2 = (0,0)$. По значениям полинома в этих точках $P_n(z_0), P_n(z_1), P_n(z_2)$ (для их расчета используется схема (4.3)) строится интерполяционный многочлен Лагранжа второго порядка.

$$L(z) = P_n(z_0)\frac{(z-z_1)(z-z_2)}{(z_0-z_1)(z_0-z_2)} +$$

$$+P_n(z_1)\frac{(z-z_0)(z-z_2)}{(z_1-z_0)(z_1-z_2)} + P_n(z_2)\frac{(z-z_0)(z-z_1)}{(z_2-z_0)(z_2-z_1)} = \qquad (4.10)$$

$$= L_0(z-z_1)(z-z_2) + L_1(z-z_0)(z-z_2) + L_2(z-z_0)(z-z_1) =$$

$$= Az^2 - Bz + C,$$

где

$$A = L_0 + L_1 + L_2, \quad B = L_0(z_1+z_2) + L_1(z_0+z_2) + L_2(z_0+z_1),$$

$$C = L_0 z_1 z_2 + L_1 z_0 z_2 + L_2 z_0 z_1, \quad L_0 = \frac{P_n(z_0)}{(z_0-z_1)(z_0-z_2)},$$

$$L_1 = \frac{P_n(z_1)}{(z_1-z_0)(z_1-z_2)}, \; L_2 = \frac{P_n(z_2)}{(z_2-z_0)(z_2-z_1)}.$$

Находятся два корня этого многочлена

$$z_{1,2} = \frac{B \pm \sqrt{B^2 - 4AC}}{2A}.$$

и за новую точку z_2 берется тот из них, который расположен ближе к предыдущей точке z_2. При этом первая точка предыдущей тройки отбрасывается. По новым точкам тем же порядком строится очередная новая точка и т.д. Построенная таким образом последовательность точек всегда сходится к некоторому корню.

Процесс повторяется до тех пор, пока значение полинома для новой точки z_2 (или расстояние между старой и новой точками z_2) не будет меньше заданной малой величины $P(z_2) < eps$.

После того, как корень найден, порядок полинома понижается с использованием схем (4.6) или (4.9) соответственно для действительного корня (мнимая часть корня равна нулю) или для случая комплексно сопряженных корней (мнимая часть корня отлична от нуля). Затем поиск корня продолжается уже для полинома меньшего порядка.

Это повторяется, пока текущая размерность полинома больше двух. В противном случае корни полинома второго или первого порядка находятся просто по известным формулам.

Замечание. Все переменные (кроме коэффициентов полинома) в алгоритме должны быть описаны как комплексные. Соответственно, и все операции над ними должны учитывать их природу.

На рис. 4.9. приведены корни полинома, вычисленные методом парабол в рамках программы, описанной в предыдущем параграфе.

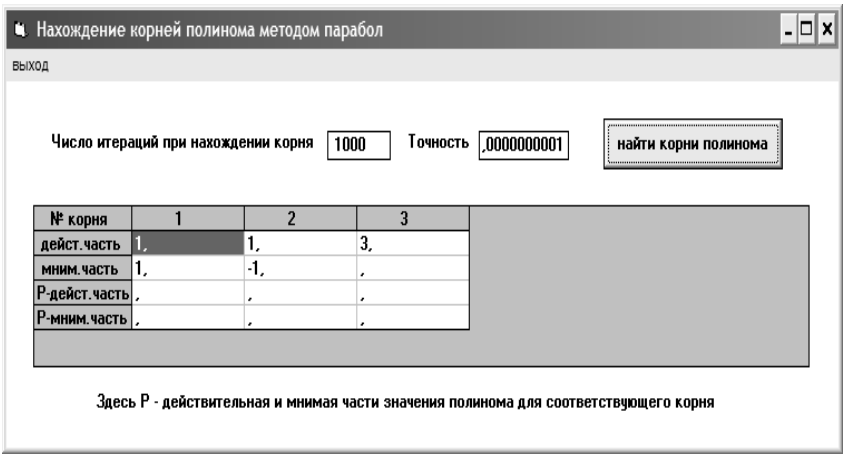


Рис. 4.9. Корни полинома $x^3 - 5x^2 + 8x - 6$, определенные методом парабол. Форма вызывается пунктом меню «метод парабол» на главной форме рис. 4.1

При реализации алгоритма метода парабол для комплексных чисел был определен новый тип переменной – комплексное число, состоящее из действительной и комплексной частей. Все операции с комплексными числами, используемые в алгоритме

$$(a+ib)\pm(c+id)=(a\pm c)+i(b\pm d);$$
$$(a+ib)(c+id)=(ac-bd)+i(ad+cb);$$
$$\frac{(a+ib)}{(c+id)}=\frac{(ac+bd)+i(cb-ad)}{(c^2+d^2)};$$
$$\sqrt{a\pm ib}=\pm[\sqrt{\frac{r+a}{2}}\pm i\sqrt{\frac{r-a}{2}}],\ r=\sqrt{a^2+b^2},$$

были оформлены в виде соответствующих подпрограмм. Листинги на Visual Basic этих программ и прицедуры, реализующей метод парабол, приведены ниже:

```
'складывает два комплексных числа
Sub addc (a As complex, b As complex, c As complex)
  a.r = b.r + c.r
  a.c = b.c + c.c
End Sub

'вычитание для двух комплексных чисел
Sub subc (a As complex, b As complex, c As complex)
  a.r = b.r - c.r
  a.c = b.c - c.c
End Sub

'умножение двух комплексных числа
Sub mulc (aa As complex, b As complex, c As complex)
 aa.r = b.r * c.r - b.c * c.c
 aa.c = b.r * c.c + b.c * c.r
End Sub

'деление для двух комплексных чисел
Sub divc (a As complex, b As complex, c As complex)
 Dim d#
```

```
 d = c.r * c.r + c.c * c.c
 If d = 0# Then
   a.r = 0
   a.c = 0
   Exit Sub
 End If
 a.r = (b.r * c.r + b.c * c.c) / d
 a.c = (c.r * b.c - b.r * c.c) / d
End Sub

'корень квадратный из комплексного числа
Sub sqrtc (a As complex, b As complex)
  Dim r#
  r = Sqr(b.r * b.r + b.c * b.c)
  a.r = Sqr((r + b.r) / 2)
  a.c = Sqr((r - b.r) / 2)
  If b.c < 0 Then
    a.c = -a.c
  End If
End Sub

Sub muller ()
 ReDim zn(3)
 ReDim l(3)
 Dim dz#, dz1#, dz2#, normp#
 Dim it%, txt$
 Dim z0 As complex
 Dim z1 As complex
 Dim z2 As complex
 Dim z3 As complex
 Dim z4 As complex
 Dim z5 As complex
 Dim aa As complex
 Dim bb As complex
 Dim cc As complex
 Dim dd As complex
```

```
Dim l0 As complex
Dim l1 As complex
Dim l2 As complex
nr = 0        'число найденных корней
Do While n > 2
   ‘наличие нулевого корня
   If Abs(a(n)) < .0001 Then
     root(nr).r = 0
     root(nr).c = 0
     nr = nr + 1
     n = n - 1
     GoTo Lab
   End If
   '
 zn(0).r = -1
 zn(0).c = 0
 zn(1).r = 1
 zn(1).c = 0
 zn(2).r = 0          '
 zn(2).c = 1          '0
 For i = 0 To 2       'значения полинома в трех точках
   Call polinomc(zn(i))
   l(i).r = p.r
   l(i).c = p.c
 Next i
 normp = Sqr(l(2).r * l(2).r + l(2).c * l(2).c)
 it = 0
 Do While normp > eps
   '
   Call subc(z0, zn(0), zn(2))     'zn(0)-zn(2) -> z0
   Call subc(z1, zn(0), zn(1))     'zn(0)-zn(1) -> z1
   Call mulc(z3, z0, z1)           'z0*z1 -> z3
   Call divc(l0, l(0), z3)         'l(0)/z3 -> l0   L0

   Call subc(z0, zn(1), zn(0))     'zn(1)-zn(0) -> z0
   Call subc(z1, zn(1), zn(2))     'zn(1)-zn(2) -> z1
```

```
Call mulc(z3, z0, z1)          'z0*z1 -> z3
Call divc(l1, l(1), z3)        'l(1)/z3 -> l1   L1

Call subc(z0, zn(2), zn(1))    'zn(2)-zn(1) -> z0
Call subc(z1, zn(2), zn(0))    'zn(2)-zn(0) -> z1
Call mulc(z3, z0, z1)          'z0*z1 -> z3
Call divc(l2, l(2), z3)        'l(2)/z3 -> l2   L2
'
Call addc(z0, l0, l1)
Call addc(aa, z0, l2)          'A   aa=l0+l1+l2
'
Call addc(z3, zn(2), zn(1))    'zn(2)+zn(1) -> z3
Call mulc(z0, l0, z3)          'l0*z3 -> z0     L0*(z(2)+z(1))
Call addc(z3, zn(0), zn(2))    'zn(0)+zn(2) -> z3
Call mulc(z1, l1, z3)          'l1*z3 -> z1     L1*(z(0)+z(2))
Call addc(z3, zn(1), zn(0))    'zn(1)+zn(0) -> z3
Call mulc(z2, l2, z3)          'l2*z3 -> z2     L2*(z(1)+z(0))
Call addc(z3, z0, z1)
Call addc(bb, z3, z2)          'B   bb=z0+z1+z2
'
Call mulc(z3, zn(2), zn(1))    'zn(2)*zn(1) -> z3
Call mulc(z0, l0, z3)          'L0*z3 -> z0
Call mulc(z3, zn(2), zn(0))    'zn(2)*zn(0) -> z3
Call mulc(z1, l1, z3)          'L1*z3 -> z1
Call mulc(z3, zn(0), zn(1))    'zn(0)*zn(1) -> z3
Call mulc(z2, l2, z3)          'L2*z3 -> z2
Call addc(z3, z0, z1)
Call addc(cc, z3, z2)          'C   cc=z0+z1+z2
'
Call mulc(z3, bb, bb)          'B*B -> z3
Call mulc(z4, aa, cc)          'A*C -> z4
z4.r = 4 * z4.r
z4.c = 4 * z4.c                '4*A*C -> z4
Call subc(dd, z3, z4)          'D   dd=z3-z4
'
Call sqrtc(z3, dd)             'sqr(dd) -> z3
```

```
        aa.r = 2 * aa.r
        aa.c = 2 * aa.c                 '2A
        Call addc(z4, bb, z3)
        Call subc(z5, bb, z3)
        Call divc(z0, z4, aa)           'z0     два корня
        Call divc(z1, z5, aa)           'z1

        'находим ближайший корень
        Call subc(z3, zn(2), z0)        'zn(2)-z0 -> z3
        Call subc(z4, zn(2), z1)        'zn(2)-z1 -> z4
        dz1 = Sqr(z3.r * z3.r + z3.c * z3.c)
        dz2 = Sqr(z4.r * z4.r + z4.c * z4.c)
        zn(0) = zn(1)
        zn(1) = zn(2)
        l(0) = l(1)
        l(1) = l(2)
        If dz1 < dz2 Then
          dz = dz1
          zn(2) = z0
        Else
          dz = dz2
          zn(2) = z1
        End If
        it = it + 1
        If it > k Then
          MsgBox "При поиске корня исчерпаны все итерации !", 48,
"Сообщение"
          Exit Do     '? do or sub
        End If
        Call polinomc(zn(2))
        l(2).r = p.r       'значение полинома в третей точке
        l(2).c = p.c
        normp = Sqr(l(2).r * l(2).r + l(2).c * l(2).c)
      Loop
      root(nr) = zn(2)
      nr = nr + 1
```

```
    '
      If Abs(zn(2).c) < .0001 Then
        Call extr_root1(zn(2).r)
      Else
        Call extr_root2(zn(2))
        root(nr) = zn(2)                    'сохраняем второй комплексно-
сопряженный корень
        root(nr).c = -root(nr).c
        nr = nr + 1
      End If
    Lab:
     Loop
      'корни линейного или квадратичного полинома
     If n = 1 Then
       root(nr).r = -a(1) / a(0)
       root(nr).c = 0#
       nr = nr + 1
     Else
       Call fine_root2
       root(nr).r = rr1
       root(nr).c = rc1
       nr = nr + 1
       root(nr).r = rr2
       root(nr).c = rc2
       nr = nr + 1
     End If
    End Sub

    'вычисление значения полинома в точке x
    Sub polinomc (x As complex)
      Dim i%, r1 As Double
      p.r = a(0)
      p.c = 0
      For i = 1 To n
        r1 = p.r
        p.r = x.r * p.r - x.c * p.c + a(i)
```

```
    p.c = x.r * p.c + x.c * r1
  Next i
End Sub
```

Здесь используются переменными типа – comlex, введенного следующим оператором

```
Type complex
  r As Double
  c As Double
End Type
```

Вопросы и задания для самоконтроля

1. Почему при вычислении значения полинома схема Горнера является предпочтительной по сравнению с другими способами?
2. В каких случаях используются схемы деления полинома на линейный и квадратичный множители?
3. Каким образом можно грубо оценить местоположение корней полинома на комплексной плоскости?
4. Опишите, каким образом можно с помощью приведенной выше программы последовательно найти все корни полинома.
5. В чем отличие методов наискорейшего и покоординатного спусков?
6. Составьте алгоритм, реализующий метод парабол.
7. Попробуйте найти все корни полинома: $x^3 - 5x^2 + 8x - 6$.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторный практикум выполняется с использованием программ, которые упоминались в данном пособии.

Краткое описание возможностей программ

1. *Нормы* – приложение позволяет рассчитать три нормы произвольной матрицы A, введенной пользователем. Поскольку программа в процессе своей работы определяет все собственные числа вспомогательной матрицы A^TA, то с ее помощью можно в ручную найти меру обусловленности исходной матрицы.

2. *ITER_MET* – в приложении реализованы три итерационных метода, с помощью которых можно найти решения произвольной системы уравнений. Исходную систему можно автоматически преобразовать в равносильную систему с положительно определенной симметричной матрицей. В приложении исходная система может быть сформирована случайным образом с помощью датчика равномерно распределенных псевдослучайных чисел.

3. *CONVERT* – приложение позволяет находить для произвольной матрицы обратную и значение определителя.

4. *ORTGAUSS* – приложение позволяет найти решение произвольной системы уравнений методом оптимального исключения или первым методом ортогонализации.

5. *KV_ROOT* – приложение позволяет найти решение произвольной системы уравнений с симметричной положительно определенной матрицей методом квадратного корня или вторым методом ортогонализации. В приложении можно преобразовать обычную систему в равносильную с симметричной положительно определенной матрицей.

6. *REGUL* – приложение строит решение для произвольной системы и ее регуляризованного прототипа. Может быть использована для поиска нормального решения для плохо обусловленных систем.

7. *LMDMAX* – в приложении реализована итерационная процедура нахождения максимального по модулю собственного

числа, которую в рамках данного приложения можно использовать для нахождения минимального по модулю собственного числа, а также для определения границ отрезка, на котором располагаются все действительные собственные числа симметричной матрицы.

8. *M_DANIL* – в приложении для нахождения характеристического полинома произвольной матрицы используется метод Данилевского. Предусмотрено решение этой задачи и в случае, когда полином распадается на несколько полиномов. Сами собственные значения матрицы находятся методом парабол.

9. *KRILOV* – приложение позволяет определить собственные значения произвольной матрицы через ее характеристический полином, коэффициенты которого находятся методом Крылова.

10. *ВРАЩЕНИЕ* – в приложении методом вращения решается полная проблема собственных значений произвольной симметричной матрицы. В приложении исходная матрица может быть сформирована случайным образом с помощью датчика равномерно распределенных псевдослучайных чисел.

11. *МПАРАБОЛ* – приложение позволяет: локализовать местоположение корней полинома на комплексной плоскости; рассчитать значение полинома в произвольной точке; выделить из него действительный корень, а также найти методом парабол все его корни.

Задания

1. Исследовать скорость сходимости итерационных методов для конкретной системы линейных уравнений.

2. Исследовать точность различных прямых методов при решении конкретной системы линейных уравнений.

3. Найти все собственные числа конкретной матрицы, используя все возможные методы. Сравнить результаты.

4. Найти графически местоположение на комплексной плоскости корней конкретного полинома и определить их точное значение.

Список литературы

1. Турчак Л.И. Основы численных методов. М.: Наука, 1987.

2. Крылов В.И., Бобков В.В., Монастырский П.И. Вычислительные методы. Т.1. М.: Физматгиз, 1976.

3. Калиткин Н.Н. Численные методы. М.: Наука, 1978.

4. Волков Е.А. Численные методы. М.: Наука, 1987.

5. Демидович Б.П., Марон И.А. Основы вычислительной математики. М.: Наука, 1966.

6. Бахвалов Н.С. Численные методы. М.: Наука, 1973.

7. Воеводин В.В. Вычислительные основы линейной алгебры. М.: Наука, 1977.

8. Фаддеев Д.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Наука, 1963.

9. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. М.: Бином, 2008.

Редактор *Е.Н. Кочубей*

Подписано в печать 21.10.2010. Формат 60×84 1/16
Объем 8,0 п.л. Уч. изд. л. 8,0. Тираж 50 экз.
Изд. № 078-1. Заказ № 330.
Национальный исследовательский ядерный университет «МИФИ».
115409, Москва, Каширское шоссе, 31.