Отчет по лабораторной работе №1: Решение СЛАУ численными методами

Чигарев Дмитрий 381807-1

1. Подготовка системы

Исходная СЛАУ, предлагаемая для решения имеет вид:

$$A = \begin{pmatrix} 0.197 & 0.219 & 0.274 & 3.127 \\ 0.186 & 0.275 & 2.987 & 0.316 \\ 0.329 & 2.796 & 0.179 & 0.278 \\ 2.389 & 0.273 & 0.126 & 0.418 \end{pmatrix} \quad b = \begin{pmatrix} 0.869 \\ 0.529 \\ 0.297 \\ 0.144 \end{pmatrix}$$

Для удобства работы с системой, приведем её к виду, в котором преобладают диагональные элементы матрицы (на диагонали стоят максимальный в строке элемент):

```
# Примечание: коды всех используемых функций вынесены в приложении
>>> [A, b] = transform(A, b)
>>> disp("Преобразованная система с преобладанием диагональных элементов:", [A, b])
  "Преобразованная система с преобладанием диагональных элементов:"

   2.389   0.273   0.126   0.418   0.144
   0.329   2.796   0.179   0.278   0.297
   0.186   0.275   2.987   0.316   0.529
   0.197   0.219   0.274   3.127   0.869
```

2. Устойчивость задачи

Исследуем устойчивость задачи решения данной СЛАУ — оценим зависимость изменения вектора решения от малых колебаний системы.

Изменение решения при:

$$\frac{|\Delta x|}{|x|} \leq C \frac{|\Delta b|}{|b|} - \text{возмущении правой части}$$

$$\frac{|\Delta x|}{|x|} \leq D \frac{|\Delta A|}{|A + \Delta A|} - \text{возмущении левой части}$$

Справедливой оценкой для коэффициентов C и D служит число обусловленности:

$$cond(A) = |A^{-1}||A|$$

Вычислим число обусловленности для нашей системы:

```
>>> cond_n = cond(A)
>>> printf("Число обусловленности равно: %f\n", cond_n)
Число обусловленности равно: 1.667647
```

Число обусловленности является коэффициентом роста относительной погрешности при возмущении системы. Малыми “возмущениями системы” могут стать ошибки округления при формировании СЛАУ или же ошибки измерения данных, представленных в системе.

3, 4. Оценка погрешностей

По условию задания: *абсолютная погрешность всех членов системы равна 0.001.* Оценим, с какой точностью мы можем получить решение.

Для этого определим величину относительной и абсолютной погрешности решения.

$$\frac{|\Delta x|}{|x|} \leq \frac{cond(A)}{1 - cond(A) * \delta_A} * (\delta_b + cond(A) * \delta_A) - \text{относительная погрешность}$$

$$|\Delta x| \leq |A^{-1}||\Delta b| - \text{абсолютная погрешность (при возмущении правой части)}$$

В случае нашей системы оценки примут следующее значение:

```
>>> [rel_err, abs_err] = get_err(A, b, err=0.001)
>>> printf("Оценка относительной погрешности решения: %f\n", rel_err)
Оценка относительной погрешности: 0.002319
>>> printf("Оценка абсолютной погрешности решения: %f\n", abs_err)
Оценка абсолютной погрешности: 0.000455
```

Из числа обусловленности и оценки относительной погрешности, можно сделать вывод о гарантированной точности решения в 2 знака после запятой.

Соответственно, ответом на вопрос: можно ли гарантировано получить решение с точностью *0.001*, для системы, у которой погрешность членов системы также составляет *0.001*, – нет.

5, 6. Метод Гаусса

Решим СЛАУ методом Гаусса. Для этого приведем систему к треугольному виду, а затем последовательно выразим неизвестные переменные.

```
>>> gaus_x = solve_with_gaus(A, b)
>>> disp("Метод Гаусса выдал следующее решение:", gaus_x)
  "Метод Гаусса выдал следующее решение:"

  -0.0009828
   0.0712863
   0.1430468
   0.2604372
```

Посчитаем относительную и абсолютную погрешность полученного решения:

```
>>> ref_x = A^-1 * b  // Точное решение
>>> printf("Абсолютная погрешность: %.e\n", norm(gaus_x - ref_x))
Абсолютная погрешность: 2.2708e-17
>>> printf("Относительная погрешность: %.e\n", norm(gaus_x-ref_x)/norm(ref_x))
Относительная погрешность: 6.9389e-18
```

7. Метод простых итераций

Итерационные методы основаны на последовательном приближении начального решения.

Исходная СЛАУ преобразовывается к эквивалентной ей:

$$x = Qx + c$$

Далее строится ряд:

$$x^{k+1} = Qx^k + c, \quad k = 0, 1, 2, 3, \ldots$$

Можно доказать, что при выполнении некоторых условий, накладываемых на матрицу Q, этот ряд сходится к точному решению x^*:

$$x^* = \lim_{k \to \infty} x^k = (E - Q)^{-1} * c$$

Отсюда можно вывести оценку абсолютной погрешности решения на $k-$ом шаге итерационного процесса:

$$\left|x^* - x^k\right| = \left|\left(Q^k + Q^{k+1} + \cdots\right)c - Q^k x^0\right| \leq \left|Q^k\right| * \left|(E - Q)^{-1}c - x^0\right| \leq \left|Q^k\right| * \left(|x^0| + \frac{|c|}{1 - |Q^k|}\right)$$

Также видно, что, решив следующее неравенство для k, мы вычислим необходимое кол-во итераций для получения решения с заданной точностью:

$$\left|Q^k\right| * \left(|x^0| + \frac{|c|}{1 - |Q^k|}\right) < \varepsilon$$

Ограничением на матрицу Q, является:

$$\max_{p \epsilon SPEC_Q} p < 1, \text{ где } SPEC_Q - \text{множество собственных чисел матрицы } Q$$

Поиск собственных чисел матрицы является трудоемкой задачей, и потому, проверка этого условия не является частью работы МПИ. В реализации метода из приложения это условие проверяется лишь для демонстрационных целей.

8. Метод Якоби

Изложенное выше, является определением метода простых итераций. Метод Якоби задает правила по преобразованию исходной системы к виду, пригодному для использования метода МПИ, а также предлагает способ задания начального приближения x^0.

Матрица Q, вектор c и начальное приближение x^0 строятся следующим правилам:

$$Q_{ij} = \begin{cases} 1 + \frac{A_{ij}}{A_{ii}}, \text{если } i = j \\ -\frac{A_{ij}}{A_{ii}}, \text{если } i \neq j \end{cases}$$

$$c_i = \frac{b_i}{A_{ii}}$$

$$x^0 = \mathrm{c}$$

Решим нашу систему методом Якоби с заданной погрешностью в 0.01 и 0.001:

```
>>> x1 = solve_with_iter(A, b, eps=0.01, verbose=%T)
Запуск итерационного метода на 4 шагов...

Шаг: 1 | Погрешность: 0.09795509 |
Текущее решение:
  -0.009827
   0.0601614
```

```
   0.1341681
   0.2511471

Шаг: 2 | Погрешность: 0.02442234 |
Текущее решение:
   0.0023823
   0.0738191
   0.1456046
   0.2625515

Шаг: 3 | Погрешность: 0.00678556 |
Текущее решение:
  -0.001777
   0.0705163
   0.1423804
   0.2598237

Шаг: 4 | Погрешность: 0.00182003 |
Текущее решение:
  -0.0007523
   0.0714834
   0.1432321
   0.2605995
>>> x2 = solve_with_iter(A, b, eps=0.01, verbose=%T)
Запуск итерационного метода на 6 шагов...

Шаг: 1 | Погрешность: 0.09795509 |
Текущее решение:
  -0.009827
   0.0601614
   0.1341681
   0.2511471

Шаг: 2 | Погрешность: 0.02442234 |
Текущее решение:
   0.0023823
   0.0738191
   0.1456046
   0.2625515

Шаг: 3 | Погрешность: 0.00678556 |
Текущее решение:
  -0.001777
   0.0705163
   0.1423804
   0.2598237

Шаг: 4 | Погрешность: 0.00182003 |
Текущее решение:
  -0.0007523
   0.0714834
   0.1432321
   0.2605995

Шаг: 5 | Погрешность: 0.00049640 |
Текущее решение:
  -0.0010435
   0.0712311
```

```
   0.1429972
   0.2603926

Шаг: 6 | Погрешность: 0.00013446 |
Текущее решение:
  -0.000966
   0.071301
   0.1430604
   0.2604492
```

9. Метод Зейделя

Метод Зейделя основан на методе Якоби. Его отличием является то, что при подсчете $i-$ й компоненты приближенного вектора $k+1$, учитываются $(i-1)$ компоненты, уже полученные на этом шаге:

$$\begin{cases} x_1^{k+1} = q_{11}x_1^k + q_{12}x_2^k + \cdots + q_{1n}x_n^k + c_1 \\ x_2^{k+1} = q_{21}x_1^{k+1} + q_{22}x_2^k + \cdots + q_{2n}x_n^k + c_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n^{k+1} = q_{n1}x_1^{k+1} + q_{n2}x_2^{k+1} + \cdots + q_{nn}x_n^k + c_n \end{cases}$$

В методе Зейделя будем пользоваться следующим критерием останова (вместо заранее рассчитанного кол-ва шагов метода):

$$\left|x^{k+1} - x^k\right| < \varepsilon$$

При выполнении этого условия будем останавливать итерационный процесс.

Решим нашу систему методом Зейделя с заданной погрешностью в 0.001:

```
>>> x3 = solve_with_zeidel(A, b, eps=0.01, verbose=%T)
Шаг: 1 | Погрешность: 0.088608.8 |
Текущее решение:
  -0.009827
   0.0684103
   0.1420147
   0.2612862

Шаг: 2 | Погрешность: 0.009593.8 |
Текущее решение:
  -0.0007482
   0.0712403
   0.1429466
   0.2604344

Шаг: 3 | Погрешность: 0.000250.8 |
Текущее решение:
  -0.0009717
   0.0712917
   0.143046
   0.2604362
```

10. Сравнение исследованных методов

Метод	Решение	Относительная погрешность	Абсолютная погрешность	Сложность
Гаусс	-0.0009828 0.0712863 0.1430468 0.2604372	2.2708e-17	6.9389e-18	$O(n^3)$
Якоби	-0.000966 0.071301 0.1430604 0.2604492	9.4058e-05	2.8741e-05	$O(kn^2)$ где k- кол-во итераций
Зейдель	-0.0009717 0.0712917 0.143046 0.2604362	4.0389e-05	1.2342e-05	$O(kn^2)$ где k- кол-во итераций

Приложения

1. Ссылка на репозиторий с исходным кодом: https://github.com/proxodilka/numerical-analysis-labs/blob/master/lab1_system_linear_equtions/lab1.sce