Реализация решения СЛАУ итерационным методом (Градиентные методы) (1.2.4(а) – метод наискорейшего спуска)

Группа: ПМ-2001

Студент: Иксанов Марат Васильевич

1. Суть метода и алгоритм решения:

Для решения уравнения $Ax = f$ методом наискорейшего спуска необходимо, чтобы матрица A была симметрична и положительно определена.

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, f = \begin{pmatrix} f_{11} \\ \vdots \\ f_{n1} \end{pmatrix}, x = \begin{pmatrix} x_{11} \\ \vdots \\ x_{n1} \end{pmatrix}$$

Пусть x^* искомое решение СЛАУ. Введем функцию $H(x) = (Ax, x) - (f, x)$. Так как градиент функции $H(x)$ по некоторому направлению z имеет вид $-2(Ax - f)$, то это значит, что решение СЛАУ совпадает с точкой минимума самой функции $H(x)$. С этого момента задачу поиска решения x^* можно заменить равносильной задачей поиска минимума функции $H(x)$. Определим начальное приближение,например, $x^0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$, от которого будем опираться при нахождении дальнейших x_k приближений. Вектор невязки $r_k = f - Ax_k$ будем считать направлением спуска, а l_k выбирается из условия минимизации $H(x)$ вдоль направления спуска:

$H(x_{k+1}) = \cdots = H(x_k) - 2l_k(r_k, r_k) + l^2(Ar_k, r_k)$,где $r_{k+1} = f - Ax_{k+1} = r - l_k Ar_k$.

$$\min_{l_k} H(x_k)_1 \Rightarrow H(x_k)'_{l_k} = 0 \Rightarrow l_k = \frac{(r_k, r_k)}{(Ar_k, r_k)}$$

Таким образом, искомые приближения строятся по формуле: $x_{k+1} = x_k + l_k r_k$.

На данный момент не учтен один главный момент: мы не установили границы, на котором итерации заканчиваются.

Существуют 3 основных критерия прекращения итерационного процесса:

1) По числу итераций. $k < K_{max}$, где K_{max} задается пользователем
2) По близости к решению нормы разности $x_{k+1} - x_k$, которая должна быть меньше заданного δ
3) По малости невязки. Норма вектора невязки должна быть меньше некоторого ε

Также стоит отметить, что поскольку данный метод работает только для симметричных положительно определенных матриц, следовательно, данный метод не работает для несимметричных матриц. Для решения этой проблемы, чтобы использовать метод на любой матрице можно применить, например, 1 метод преобразования Гаусса, путем умножения левой и правой части на транспонированную матрицу A

2. Программная реализация:

Входные данные: матрица A и матрица правой части f.

Выходные данные: матрица x.

Метод наискорейшего спуска реализован с помощью Wolfram Mathematica:

Мною были созданы 5 функций, которые выполняют данную задачу, однако в каждой из них используются, либо отличные от других методы, либо различные критерии прекращения, описанные выше. Рассмотрим каждую из них.

```
stDescentVer1[A_, f_, k_] := Module[
  {n = Length@f, x0, x1, x2, r1, l1},
  x0 = ConstantArray[0, {n, 1}];
  x1 = x0;
  Do[
   r1 = f - A.x1;
   l1 = (Transpose[r1].r1)/(Transpose[(A.r1)].r1)[[1, 1]];
   x2 = x1 + l1*r1;
   x1 = x2
   , {i, 1, k}];
  x2
 ]

stDescentVer2[A_, f_, k_] := Nest[# + (Transpose[(f - A.#)].(f - A.#))/(Transpose[(A.(f - A.#))].(f - A.#))[[1, 1]]*(f - A.#) &, ConstantArray[0, {Length@f, 1}], k]

stDescentVer3[A_, f_, e_] := NestWhile[# + (Transpose[(f - A.#)].(f - A.#))/(Transpose[(A.(f - A.#))].(f - A.#))[[1, 1]]*(f - A.#) &, ConstantArray[0, {Length@f, 1}], Norm[f - A.#] >= e &]

stDescentVer4[A_, f_, e_] := Module[
  {n = Length@f, x0, x1, x2, r1 = 10, l1, k = 1},
  x0 = ConstantArray[0, {n, 1}];
  x1 = x0;
  While[Norm[r1] >= e,
   r1 = f - A.x1;
   l1 = (Transpose[r1].r1)/(Transpose[(A.r1)].r1)[[1, 1]];
   x2 = x1 + l1*r1;
   x1 = x2;
   k++;
   ];
  {x1, k}
 ]

stDescentVer5[A_, f_, e_] := Module[
  {n = Length@f, x0, x1, x2, r1 = 10, l1, delta = 10, k = 1},
  x0 = ConstantArray[0, {n, 1}];
  x1 = x0;
  While[Norm[delta] >= e,
   r1 = f - A.x1;
   l1 = (Transpose[r1].r1)/(Transpose[(A.r1)].r1)[[1, 1]];
   x2 = x1 + l1*r1;
   delta = Norm[x2 - x1];
   x1 = x2;
   k++;
   ];
  {x1, k}
 ]
```

3. Результат вычисления на предоставленных входных данных и оценка точности решения:

Для оценки точности решения рассмотрим входные данные, тип данных входных матриц – числа с плавающей точкой.

Входные данные: $A1 = \begin{pmatrix} 1.00 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1.00 & 0.32 & 0.44 \\ 0.54 & 0.32 & 1.00 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1.00 \end{pmatrix}$

$$A3 = \begin{pmatrix} 4.33 & -1.12 & -1.08 & 1.14 \\ -1.12 & 4.33 & 0.24 & -1.22 \\ -1.08 & 0.24 & 7.21 & -3.22 \\ 1.14 & -1.22 & -3.22 & 5.43 \end{pmatrix}, f = \begin{pmatrix} 0.3 \\ 0.5 \\ 0.7 \\ 0.9 \end{pmatrix}$$

Все 5 функций решают примерно с одинаковой нормой невязки и получают довольно точный ответ. Один из результатов, вычисленный с помощью собственной функции, например функции 2 и для большей точности выберем количество итераций $k = 100$:

```
stDescentVer1[a1, f, 100]
```

```
{{-1.25779}, {0.0434873}, {1.03917}, {1.48239}}
```

Решение с помощью встроенной функции $LinearSolve$:

```
LinearSolve[a1, f]
```

```
{{-1.25779}, {0.0434873}, {1.03917}, {1.48239}}
```

```
Norm[f - a1.stDescentVer1[a1, f, 100]]
```

5.16164×10^{-15}

```
Norm[f - a1.LinearSolve[a1, f]]
```

5.55112×10^{-17}

Нормы векторов невязки практически не отличаются.

Стоит отметить, что остальные функции работают также хорошо, поэтому предлагаю не сравнивать все остальные функции с встроенной, а сравнить их точность между собой в зависимости от выбранных данных для прекращения итерационного процесса.

Для начала следует описать особенности каждой из функций:

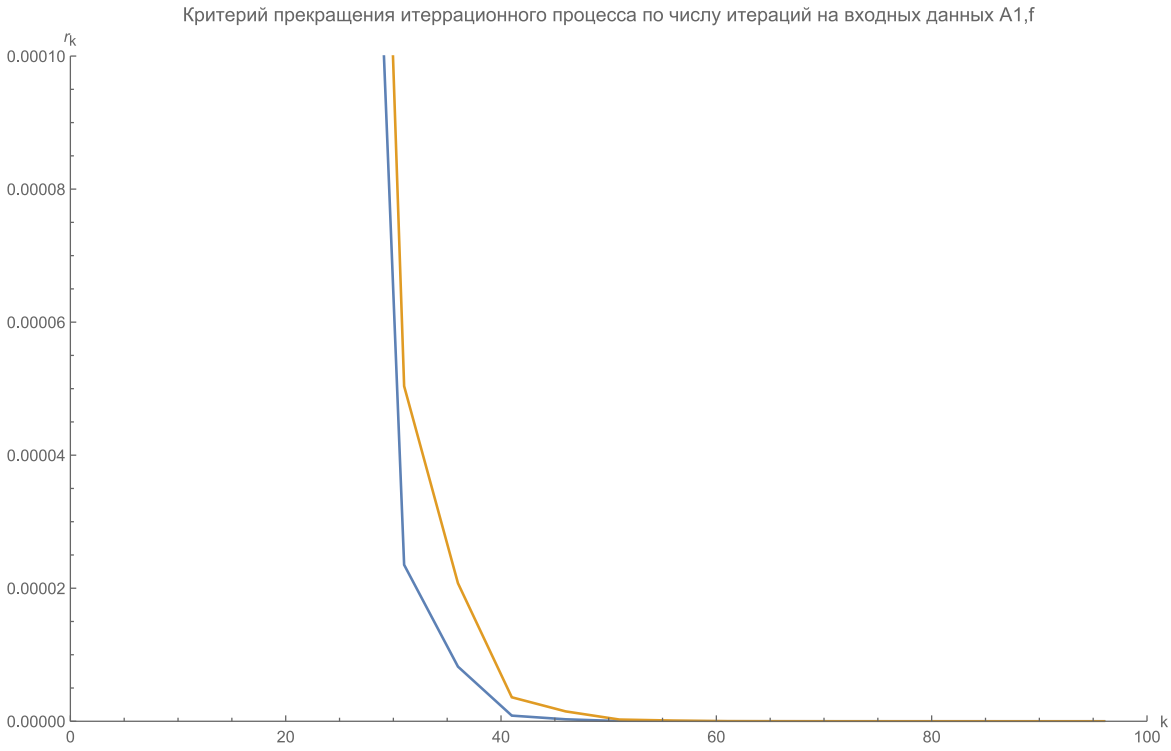
1, 2) Критерий прекращения K_{max}

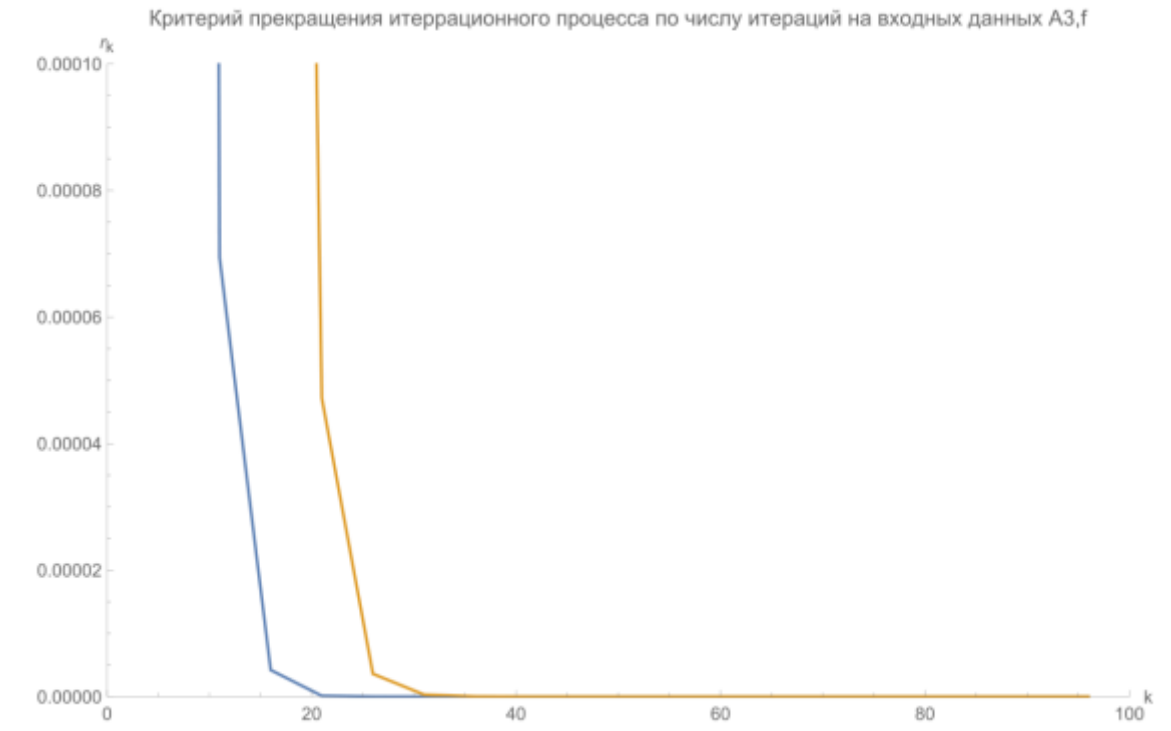
3, 4) Критерий прекращения малость невязки

5)Критерий прекращения близость к решению

Предлагается далее сравнить схожие между собой функции на всех входных данных:

1) 1 – синяя и 2 – желтая функции:
   Рассматривается промежуток $0 < k < 100$ на оси X и $0 < r < 0.0001$ на оси У

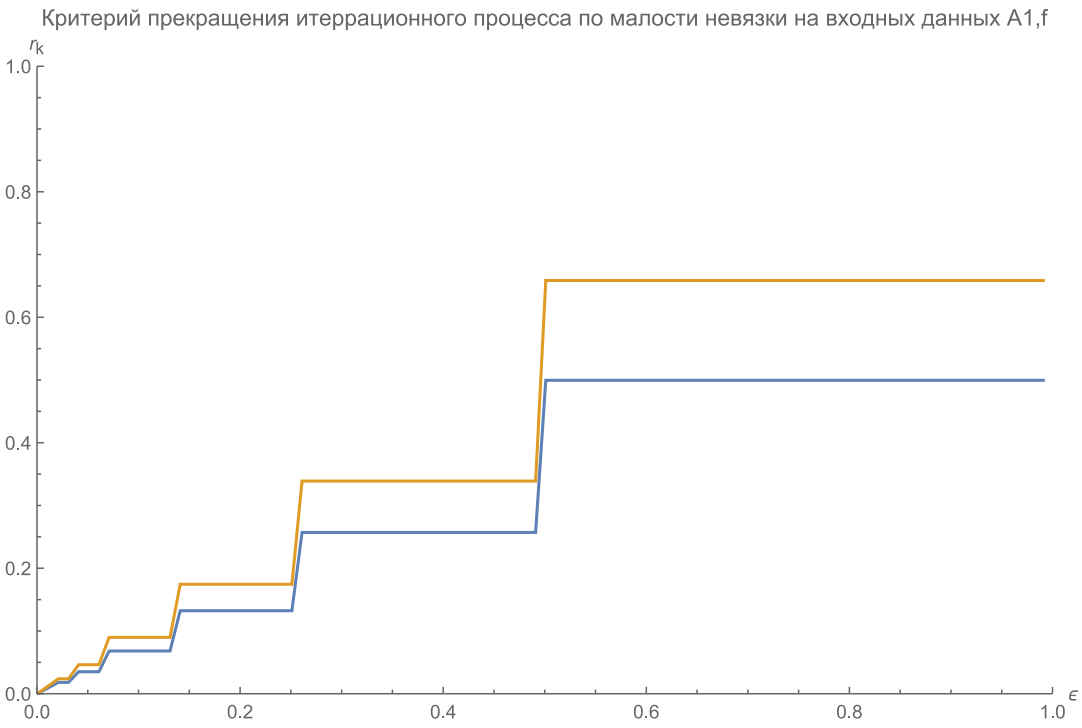


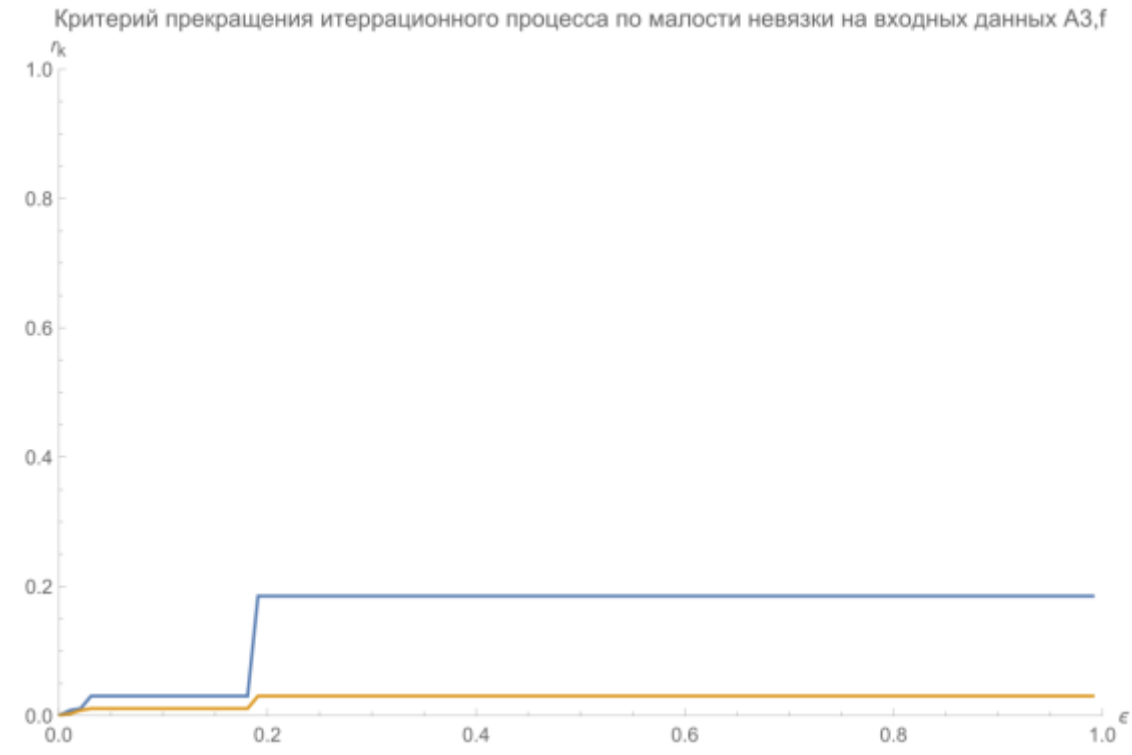


Стоит подметить, что обе функции показывают себя по разному при разных входных данных, однако в общем случае при $k > 50$ обе функции вычисляют достаточно точно и их норма вектора невязки практически равна нулю.

2) 3 - синяя и 4 - желтая функции:

Рассматривается промежуток $0 < \varepsilon < 1$ на оси X и $0 < r < 0.0001$ на оси У





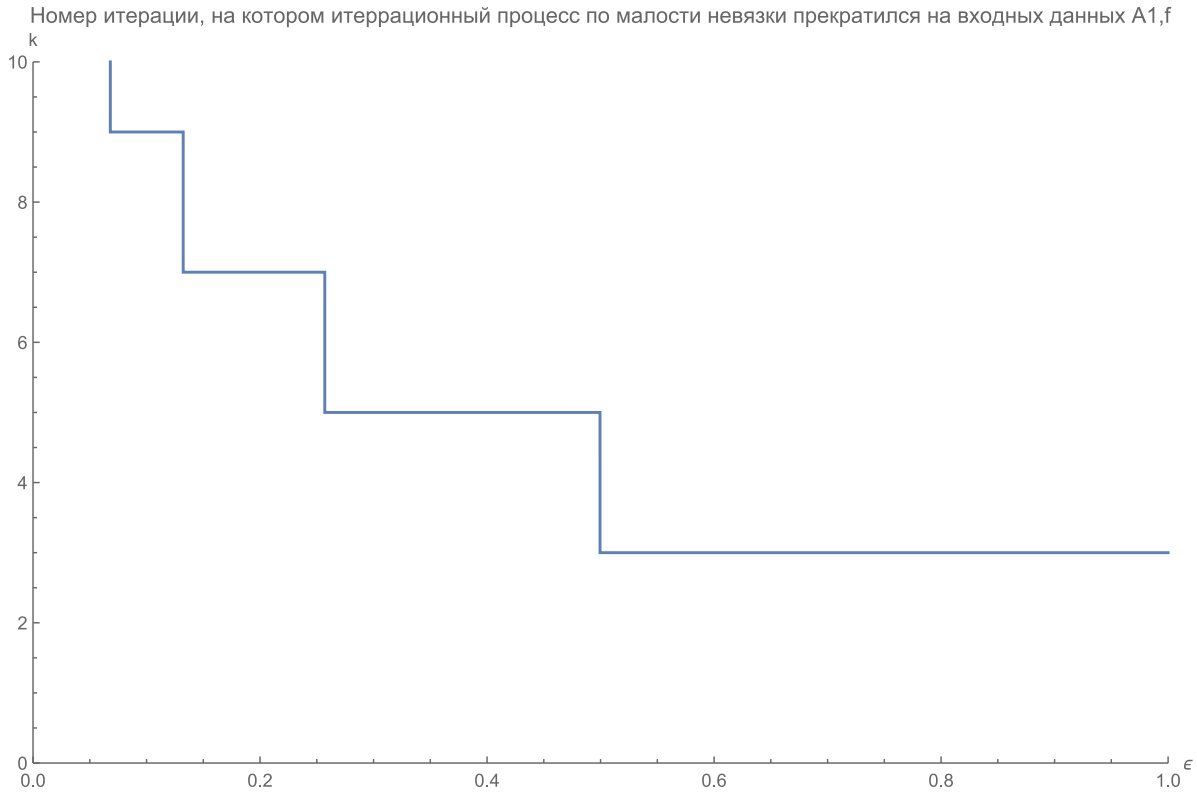
По графикам видно, что при больших значениях ε нормы векторов невязки обоих функций могут отличаться, однако при малых значениях ε нормы векторов обоих стремятся к нулю, что и логично и доказывает правильность итерационного метода.

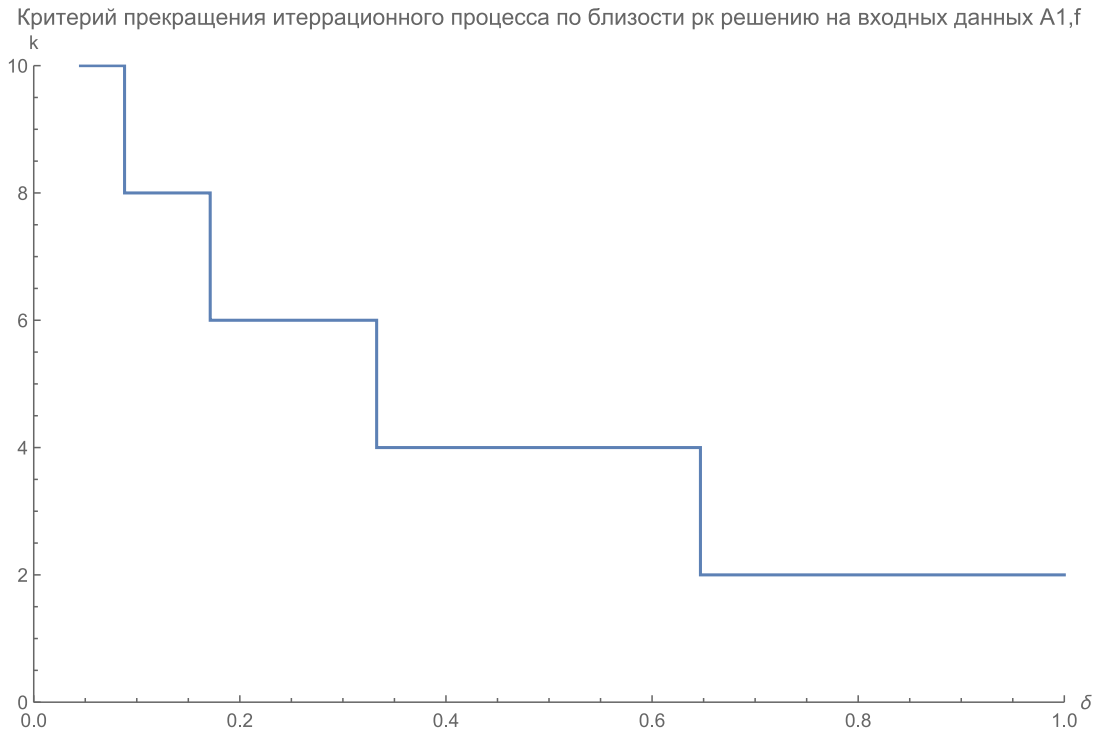
3) Стоит также рассмотреть отдельно 4 функцию, ведь она помимо вычисленного решения запоминают число итераций k, после которой норма невязки стала меньше заданного числа

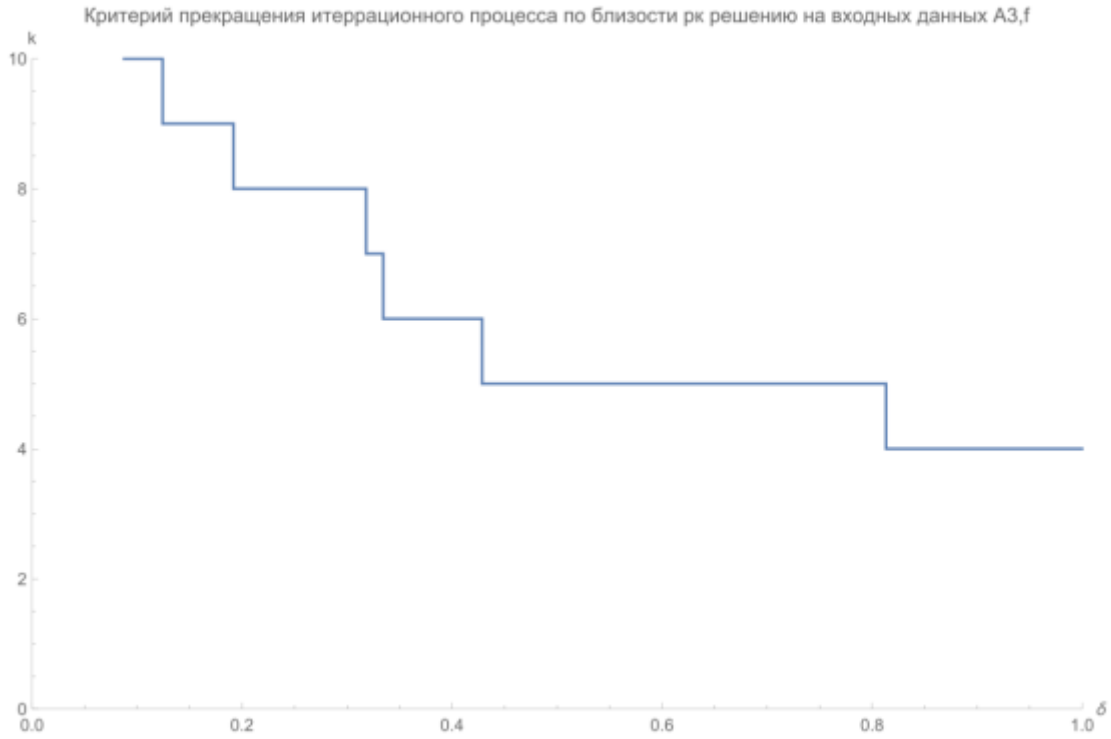
Для входных данных видно, что при уменьшении заданного числа в среднем при k = 9 норма вектора невязки практически равна 0, поскольку меньше очень малого числа.

4) Последняя функция, которую стоит рассмотреть – функция 5:

Рассматривается промежуток $0 < \delta < 1$ на оси X и $0 < k < 10$ на оси У







По данным графикам видно, что при увеличении значения k, число, с которым сравнивается норма соседних приближений уменьшается и стремится к 0, следовательно, это означает, что функция также работает корректно. Аналогично с графиком, где выбирается число, с которым сравнивается норма вектора невязки.