Реализация решения СЛАУ методом итераций (Градиентные методы. Метод наискорейшего спуска. С применением первой трансформации Гаусса)

Задача:1.2.4(а)(1)

Представим, что требуется решить СЛАУ $Ax = f$, где $A -$ положительно определенная симметричная матрица размером n на n.

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

Но может быть такой случай, что матрица A – несимметрична. Чтобы решить эту проблему можно применить первую трансформацию Гаусса. Суть этой трансформации заключается в том, что если дана несимметричная положительно определенная матрица A, то достаточно умножить в $Ax = f$ левую и правые части на A^T, таким образом получается выражение $A^T Ax = A^T f$, где B = $A^T A$ и B – положительно определенная симметричная матрица., а g = $A^T f$, такая, что $Bx = g$. Полученное решение будет удовлетворять решению исходного уравнения. Таким образом, после данного преобразования применяется метод наискорейшего спуска.

Описание метода наискорейшего спуска. Данный метод заключается в том, что поиск решения $Ax = f$ сводится к решению задачи нахождения минимума некоторой функции $H(x) = (Ax, x) - (f, x)$. Суть заключается в том, что градиент данной функции по заданному направлению равна $-2(Ax - f)$, что означает, что искомое решение $Ax = f$ совпадает с минимумом данной функции. Далее необходимо определиться с начальным вектором приближения, который задаст начало всему итерационному процессу. Для удобства можно взять нулевой вектор и назвать его x_0 . Далее определим направление спуска, которое будет являться вектором невязки на текущей итерации, и выберем шаг, на который будем спускаться на каждой итерации. Шаг определяется как $\alpha = \frac{(r,r)}{(Ar,r)}$. Он выбирается из условия минимизации заданной функции H(x) вдоль направления спуска.

$$H(x') = H(x) - 2\alpha(r, r) + \alpha(Ar, r)$$

$$\min_{\alpha} H(x') \Rightarrow \frac{d}{d\alpha} H(x') = 0 \Rightarrow \alpha = \frac{(r,r)}{(Ar,r)}.$$

Следовательно, если переобозначить индексы x' и x можно вывести итерационную формулу искомого решения.

$$x_{k+1} = x_k + \alpha_k r_k$$

Чтобы любой итерационный метод работал корректно, необходимо определить условия остановки вычислений следующей итерации.

Критерии остановки итерационного процесса следующие:

1) По числу итераций: Пользователь сам выбирает такое число k, по достижению которого процесс вычислений прекращается.
2) По близости к решению. Пользователь задает какое-то малое число, с которым сравнивается на каждой итерации норма разности текущего и нового приближения. И если эта норма меньше заданного числа, то вычисления прекращаются
3) По малости вязки. Пользователь задает какое-то мало число, с которым сравнивается на каждой итерации норма вектора невязки

$$r_k = f + Ax_k$$

Программная реализация:

Мною было реализовано 2 функции. Обе функции очень похожи, однако обладают разными критериями прекращения итерационного процесса:

Первая ищет решение СЛАУ и ищет приближения по критерию остановки – число итераций.

```
fun1[A_, f_, k_] := Module[
  {n = Length[A[[1]]], l, x1, x2},
  x1 = ConstantArray[0, {n, 1}];
  Do[ l = Transpose[f - A.x1].(f - A.x1) / Transpose[(A.(f - A.x1))].(f - A.x1) // Flatten // First;
   x2 = x1 + l*(f - A.x1);
   x1 = x2 , {i, 1, k}];
  x2
 ]
```

Вторая ищет решение СЛАУ и ищет приближения по критерию остановки – малость невязки.

```
fun2[A_, f_, e_] := Module[
  {n = Length@f, x1, x2, r1 = e + 1, k = 1},
  x1 = ConstantArray[0, {n, 1}];
  While[Norm[r1] >= e, r1 = f - A.x1;
   x2 = x1 + First[Flatten[Transpose[r1].r1 / Transpose[(A.r1)].r1]] * r1;
   x1 = x2;
   k++;];
  {x1, k}
 ]
```

Реализация написана с помощью среды $Wolfram\ Mathematica$. Необходимо узнать, насколько точно решают данные функции заданное уравнение. Для этого найдем вектор невязки – разность левой и правой части с подставленным найденным с помощью реализации решением – и определим его норму — точность вычисленного решения. Также следует сравнить решение со встроенными функциями среды $Wolfram\ Mathematica$, которые также решает поставленную задачу.

Входные данные:

$$A = \begin{pmatrix} 1.00 & 0.17 & -0.25 & -0.54 \\ 0.47 & 1.00 & 0.67 & -0.32 \\ -0.1 & 0.35 & 1.00 & -0.74 \\ 0.55 & 0.43 & 0.36 & 1.00 \end{pmatrix},\ f = \begin{pmatrix} 0.3 \\ 0.5 \\ 0.7 \\ 0.9 \end{pmatrix}$$

Выходные данные, полученные собственной реализацией 1ой функцией при k max = 200:

Поскольку матрица A несимметрична, применим 1 трансформацию Гаусса.

Решим полученное уравнение

```
Row[{Transpose[a2].a2 // MatrixForm , {{x1}, {x2}, {x3}, {x4}} // MatrixForm,
   Transpose[a2].f // MatrixForm}]
```

$$\begin{pmatrix} 1.5355 & 0.838 & 0.1529 & -0.059 \\ 0.838 & 1.3363 & 1.1323 & -0.2408 \\ 0.1529 & 1.1323 & 1.641 & -0.4594 \\ -0.059 & -0.2408 & -0.4594 & 1.9416 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} \begin{pmatrix} 0.953 \\ 1.183 \\ 1.284 \\ 0.06 \end{pmatrix}$$

```
fun1[a, g, 200]
```

```
{{0.851771}, {-0.625394}, {1.20887}, {0.265252}}
```

В данном случае норма невязки очень мала, что означает, что полученное решение довольно точное.

```
Norm[g - a.fun1[a, g, 200]]
```

7.76012×10^{-8}

Стоит отметить, что если взять kmax малым, то решение будет менее точным. Например при kmax = 50. Норма невязки больше, чем при kmax = 200. Это довольно логично, учитывая тот факт, что каждая итерация приближает решение к искомому.

```
Norm[g - a.fun1[a, g, 100]]
```

```
0.000134038
```

При kmax = 50 норма вектора невязки еще больше. Что также очевидно.

```
Norm[g - a.fun1[a, g, 50]]
```

```
0.0055707
```

Выходные данные, полученные собственной реализацией 2ой функцией при e = 0.0001:

```
fun2[a1, f, 0.0001]
```

```
{{{-1.25768}, {0.0434849}, {1.03907}, {1.48226}}, 29}
```

Число 29 означает, сколько понадобилось итераций, чтобы малость нормы вектора невязки была меньше заданного e. Соответственно, очевидно, что при увеличении e, число итераций будет уменьшаться и норма вектора невязки также будет увеличиваться, что приведет к потери точности вычислений.

```
Norm[f - a1.fun2[a1, f, 0.0001][[1]]]
```

```
0.000117091
```

При, например,e = 0.1, норма вектора невязки значительно увеличилась по описанным выше причинам, а при e = 0.0000001 уменьшилась

```
Norm[g - a.fun2[a, g, 0.1][[1]]]
```

```
0.0993955
```

```
Norm[g - a.fun2[a, g, 0.0000001][[1]]]
```

9.45966×10^{-8}

Выходные данные, полученные с помощью встроенное функции-1:

$$\text{Transpose}[\{x1, x2, x3, x4\}/.\ \text{Solve}[\text{Thread}[a.\{x1, x2, x3, x4\} = \\ = \{0.3,0.5,0.7,0.9\}], \{x1, x2, x3, x4\}]]//\text{MatrixForm}$$

```
{{0.851771}, {-0.625394}, {1.20887}, {0.265253}}
```

Выходные данные, полученные с помощью встроенной функции-2:

$$\text{LinearSolve}[a, f]//\text{MatrixForm}$$

```
{{0.851771}, {-0.625394}, {1.20887}, {0.265253}}
```

Несложно убедиться, что 1ая трансформация Гаусса никак не повлияла на решение, поскольку встроенные функции искали решение по исходным данным (несимметричной матрице A).

Посчитаем нормы векторов невязки встроенных функций:

Встроенная функция-1:

```
Norm[f - a2.Transpose[{x1, x2, x3, x4} /. Solve[Thread[a2.{x1, x2, x3, x4} == f], {x1, x2, x3, x4}]]]
```

4.04127×10^{-16}

Встроенная функция-2:

```
Norm[f - a2.LinearSolve[a2, f]]
```

1.57009×10^{-16}

Несложно заметить по моим функциям, что при довольно маленьком значении e во 2 функции увеличивается число итераций – повышается точность. Точно также по такому же принципу в 1ой функции чем больше задается пользователем значение K тем точнее конечный результат.

При довольно больших K и малых E нормы векторов невязки практически не отличаются от норм встроенных функций, а в некоторых случаях могут даже быть меньше них.