Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Численные методы»
на тему
«Нахождение собственных значений и собственных векторов несимметричных разреженных матриц большой размерности. Метод Арнольди»

Студент:	С. М. Власова
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Постановка задачи

Реализовать итерационный метод Арнольди решения задачи поиска собственный векторов и собственных значений разреженных матриц большой размерности.

1 Описание метода решения

1 Подпространство Крылова. Построение базиса

Итерация Арнольди — это алгоритм поиска собственных значений, который был изобретен У. Э. Арнольди в 1951 г и является важным примером итерационных методов в линейной алгебре. Арнольди находит приближение к собственным значениям и собственным векторам общих (возможно, не эрмитовых) матриц, строя ортонормированный $n-$мерный базис подпространства Крылова, что делает его особенно полезным при работе с *большими разреженными* матрицами. На практике часто берется $n = 3$.

Метод Арнольди принадлежит к классу алгоритмов линейной алгебры, которые дают *частичный результат* после небольшого числа итераций, в отличие от прямых методов, которые должны завершиться только после получения любых полезных результатов. Частичным результатом в данном случае являются первые несколько векторов *нового базиса*, который строит алгоритм.

Интуитивно понятный метод поиска наибольшего (по модулю) *собственного значения* данной матрицы m на m A — это степенная итерация: начиная с произвольного начального вектора b, вычисляем $A \cdot b$, $\;A^2 \cdot b$, $\;A^3 \cdot b, \ldots$, нормализуя результат после каждого применения матрицы A. Эта последовательность сходится к собственному вектору $A^{n-1} \cdot b$, соответствующему собственному значению с наибольшим абсолютным значением λ_1. Так, мы формируем так называемую **матрицу Крылова:**

$$K_n = \begin{bmatrix} b & A \cdot b & A^2 \cdot b & \ldots & A^{n-1} b \end{bmatrix}$$

Столбцы этой матрицы, как правило, не ортогональны, но мы можем извлечь ортогональный базис с помощью такого метода, как *ортогонализация Грама–Шмидта*. Таким образом, полученный набор векторов является ортогональным базисом подпространства Крылова K_n.

Вероятно, что векторы этого базиса будут охватывать хорошие аппроксимации собственных векторов, соответствующих n наибольшим собственным значениям матрицы, по той же причине, что и $A^{n-1}b$ аппроксимирует доминантный собственный вектор.

2 Свойства итерации Арнольди

Верхняя матрица Хессенберга – это квадратная матрица, у которой все элементы ниже первой поддиагонали равны нулю. Матрицы Хессенберга и трёхдиагональные матрицы являются исходными точками многих алгоритмов вычисления собственных значений, поскольку нулевые значения уменьшают сложность задачи. Итерация Арнольди сводит исходную матрицу к хессенберговской форме.

Пусть Q_n – матрица размером m на n, образованная первыми n векторами Арнольди $q_1, q_1, \ldots, q_n$, и пусть H_n – верхняя матрица Хессенберга, вычисленная согласно алгоритму:

$$H_n = Q_n^{\star} \cdot A \cdot Q_n.$$

Метод ортогонализации должен быть выбран таким образом, чтобы нижние компоненты Арнольди удалялись из высших векторов Крылова.

Таким образом, имеем

$$H_n = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \ldots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \ldots & h_{2n} \\ 0 & h_{32} & h_{33} & \ldots & h_{3n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & h_{nn-1} & h_{nn} \end{bmatrix}$$

Матрицу H_n можно рассматривать, как A в подпространстве Крылова K_nс векторами Арнольди в качестве ортогонального базиса. A ортогонально проецируется на K_n.

Связь между $Q-$матрицами в последующих итерациях определяется выражением

$$A \cdot Q_n = Q_{n+1} \cdot \tilde{H}_n,$$

где $\tilde{H}_n$ – матрица размером $n+1$ на n, образованная добавлением дополнительной строки к H_n.

$$\tilde{H}_n = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \ldots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \ldots & h_{2n} \\ 0 & h_{32} & h_{33} & \ldots & h_{3n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & h_{nn-1} & h_{nn} \\ 0 & \ldots & \ldots & 0 & h_{n+1n} \end{bmatrix}$$

3 Нахождение собственных значений с помощью итерации Арнольди

Идея итерации Арнольди как алгоритма собственных значений заключается в вычислении собственных значений в подпространстве Крылова. Собственные значения H_n называются **собственными значениями Ритца**. Поскольку H_n представляет собой матрицу Хессенберга небольшого размера, ее собственные значения можно эффективно вычислить, например, с помощью алгоритма QR или, в некоторой степени, алгоритма Фрэнсиса.
На практике часто наблюдается, что некоторые из собственных значений Ритца сходятся к собственным значениям A. Поскольку H_n имеет размерность n на n, у него не более n собственных значений, и не все собственные значения A могут быть аппроксимированы. Как правило, *собственные значения Ритца* сходятся к *наибольшим* собственным значениям A. Чтобы получить наименьшие собственные значения A, вместо этого следует использовать обратную матрицу A.
Однако, детали еще не до конца изучены. Это контрастирует со случаем, когда A симметрична. В этой ситуации итерация Арнольди становится итерацией Ланцоша, для которой теория является более полной.

2 Моя задача

Итак, будем искать собственные значения Ритца, аппроксимирующие наибольшие собственные значения матрицы A. Для этого необходимо построить базис подпространства Крылова размерности $n = 3$ и ортогонально проецировать A на K_n. Далее, полученную матрицу Хессенберга необходимо разложить с помощью алгоритма QR на две составляющие – ортогональную матрицу Q и верхнюю треугольную матрицу R. Т.к. исходная матрица уже преобразована к хессенберговской форме, алгоритм будет работать быстрее.

Пример.

Возьмем матрицу A размерности 4 на 4

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 6 \\ 4 & 1 & 0 & 7 \end{bmatrix}$$

Необходимо найти собственные векторы и собственные значения этой матрицы с помощью метода Арнольди. Будем искать собственные значения Ритца, которые являются приближениями наибольших собственных значений матрицы A.

1. Построение ортонормированного базиса подпространства Крылова K_3 и матрицы Хессенберга – проекции матрицы A на это подпространство с помощью итераций Арнольди.

   (a) Начнем с произвольного вектора с нормой 1.
   Пусть *первый вектор Арнольди*

   $$q_1 = \begin{bmatrix} 0.78289 \\ 0.62216 \\ 0 \\ 0 \end{bmatrix}$$

   Тогда базис равен

   $$Q = \begin{bmatrix} 0.78289 & 0 & 0 & 0 \\ 0.62216 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) k = 1:

$$v = A \cdot q_1 = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 6 \\ 4 & 1 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} 0.78289 \\ 0.62216 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.02721 \\ 1.86648 \\ 0.78289 \\ 3.75372 \end{bmatrix}$$

i. j = 1:

$$h_{11} = q_1 \cdot v = \begin{bmatrix} 0.78289 & 0.62216 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2.02721 \\ 1.86648 \\ 0.78289 \\ 3.75372 \end{bmatrix} = 2.74833$$

$$v = v - h_{11} \cdot q_1 = \begin{bmatrix} 2.02721 \\ 1.86648 \\ 0.78289 \\ 3.75372 \end{bmatrix} - 2.74833 \cdot \begin{bmatrix} 0.78289 \\ 0.62216 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.124431 \\ 0.156578 \\ 0.78289 \\ 3.75372 \end{bmatrix}$$

$$h_{21} = \sqrt{((-0.124431)^2 + 0.156578^2 + 0.78289^2 + 3.75372^2)} = 3.8397$$

Имеем

$$H = \begin{bmatrix} 2.74833 & 0 & 0 \\ 3.8397 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$h_{21} = 3.8397 > 1e - 12 = eps \ \ ? \ \ yes:$

$$q_2 = v/h_{21} = \begin{bmatrix} -0.0324065 \\ 0.0407787 \\ 0.203893 \\ 0.977606 \end{bmatrix}; \quad Q = \begin{bmatrix} 0.78289 & -0.0324065 & 0 & 0 \\ 0.62216 & 0.0407787 & 0 & 0 \\ 0 & 0.203893 & 0 & 0 \\ 0 & 0.977606 & 0 & 0 \end{bmatrix};$$

k = 1 < 3 = n ? yes → continue.

(c) k = 2:

$$v = A \cdot q_2 = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 6 \\ 4 & 1 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} -0.0324065 \\ 0.0407787 \\ 0.203893 \\ 0.977606 \end{bmatrix} = \begin{bmatrix} 0.0491508 \\ 1.09994 \\ 5.83323 \\ 6.7544 \end{bmatrix}$$

i. j = 1:

$$h_{12} = q_1 \cdot v = \begin{bmatrix} 0.78289 & 0.62216 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.0491508 \\ 1.09994 \\ 5.83323 \\ 6.7544 \end{bmatrix} = 0.72282$$

$$v = v - h_{12} \cdot q_1 = \begin{bmatrix} 0.0491508 \\ 1.09994 \\ 5.83323 \\ 6.7544 \end{bmatrix} - 0.72282 \cdot \begin{bmatrix} 0.78289 \\ 0.62216 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.516738 \\ 0.650233 \\ 5.83323 \\ 6.7544 \end{bmatrix}$$

ii. j = 2:

$$h_{22} = q_2 \cdot v = \begin{bmatrix} -0.0324065 & 0.0407787 & 0.203893 & 0.977606 \end{bmatrix} \cdot \begin{bmatrix} -0.516738 \\ 0.650233 \\ 5.83323 \\ 6.7544 \end{bmatrix} = 7.83576$$

$$v = v - h_{22} \cdot q_2 = \begin{bmatrix} -0.516738 \\ 0.650233 \\ 5.83323 \\ 6.7544 \end{bmatrix} - 7.83576 \cdot \begin{bmatrix} -0.0324065 \\ 0.0407787 \\ 0.203893 \\ 0.977606 \end{bmatrix} = \begin{bmatrix} -0.262808 \\ 0.330701 \\ 4.23557 \\ -0.905893 \end{bmatrix}$$

$$h_{32} = \sqrt{((-0.262808)^2 + 0.330701^2 + 4.23557^2 + (-0.905893)^2)} = 4.35191$$

Имеем

$$H = \begin{bmatrix} 2.74833 & 0.72282 & 0 \\ 3.8397 & 7.83576 & 0 \\ 0 & 4.35191 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$h_{32} = 4.35191 > 1e - 12 = eps \;\; and \;\; k = 2 \leq 3 = n \;\; ? \;\; yes :$

$$q_3 = v/h_{32} = \begin{bmatrix} -0.0603891 \\ 0.0759898 \\ 0.973267 \\ -0.20816 \end{bmatrix} ; \;\; Q = \begin{bmatrix} 0.78289 & -0.0324065 & -0.0603891 & 0 \\ 0.62216 & 0.0407787 & 0.0759898 & 0 \\ 0 & 0.203893 & 0.973267 & 0 \\ 0 & 0.977606 & -0.20816 & 0 \end{bmatrix}$$

k = 2 < 3 = n ? yes → continue.

(d) k = 3:

$$v = A \cdot q_3 = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 6 \\ 4 & 1 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} -0.0603891 \\ 0.0759898 \\ 0.973267 \\ -0.20816 \end{bmatrix} = \begin{bmatrix} 0.0915905 \\ 0.0198096 \\ -1.30935 \\ -1.62268 \end{bmatrix}$$

i. j = 1:

$$h_{13} = q_1 \cdot v = \begin{bmatrix} 0.78289 & 0.62216 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.0915905 \\ 0.0198096 \\ -1.30935 \\ -1.62268 \end{bmatrix} = 0.08403$$

$$v = v - h_{13} \cdot q_1 = \begin{bmatrix} 0.0915905 \\ 0.0198096 \\ -1.30935 \\ -1.62268 \end{bmatrix} - 0.08403 \cdot \begin{bmatrix} 0.78289 \\ 0.62216 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.0258042 \\ -0.0324705 \\ -1.30935 \\ -1.62268 \end{bmatrix}$$

ii. j = 2:

$$h_{23} = q_2 \cdot v = \begin{bmatrix} -0.0324065 & 0.0407787 & 0.203893 & 0.977606 \end{bmatrix} \cdot \begin{bmatrix} 0.0258042 \\ -0.0324705 \\ -1.30935 \\ -1.62268 \end{bmatrix} = -1.85547$$

$$v = v - h_{23} \cdot q_2 = \begin{bmatrix} 0.0258042 \\ -0.0324705 \\ -1.30935 \\ -1.62268 \end{bmatrix} + 1.85547 \cdot \begin{bmatrix} -0.0324065 \\ 0.0407787 \\ 0.203893 \\ 0.977606 \end{bmatrix} = \begin{bmatrix} -0.0343252 \\ 0.0431933 \\ -0.931029 \\ 0.191239 \end{bmatrix}$$

iii. j = 3:

$$h_{33} = q_3 \cdot v = \begin{bmatrix} -0.0603891 & 0.0759898 & 0.973267 & -0.20816 \end{bmatrix} \cdot \begin{bmatrix} -0.0343252 \\ 0.0431933 \\ -0.931029 \\ 0.191239 \end{bmatrix} = -0.940593$$

$$v = v - h_{33} \cdot q_3 = \begin{bmatrix} -0.0343252 \\ 0.0431933 \\ -0.931029 \\ 0.191239 \end{bmatrix} + 0.940593 \cdot \begin{bmatrix} -0.0603891 \\ 0.0759898 \\ 0.973267 \\ -0.20816 \end{bmatrix} = \begin{bmatrix} -0.0911267 \\ 0.114669 \\ -0.0155812 \\ -0.00455421 \end{bmatrix}$$

$$h_{43} = \sqrt{((-0.0911267)^2 + 0.114669^2 + (-0.0155812)^2 + (-0.00455421)^2)} = 0.147365$$

Имеем

$$H = \begin{bmatrix} 2.74833 & 0.72282 & 0.08403 \\ 3.8397 & 7.83576 & -1.85547 \\ 0 & 4.35191 & -0.940593 \\ 0 & 0 & 0.147365 \end{bmatrix}$$

$h_{43} = 0.147365 > 1e - 12 = eps \ \ ? \ \ yes:$

$$q_4 = v/h_{43} = \begin{bmatrix} -0.618373 \\ 0.778126 \\ -0.105732 \\ -0.0309042 \end{bmatrix}; \ Q = \begin{bmatrix} 0.78289 & -0.0324065 & -0.0603891 & -0.618373 \\ 0.62216 & 0.0407787 & 0.0759898 & 0.778126 \\ 0 & 0.203893 & 0.973267 & -0.105732 \\ 0 & 0.977606 & -0.20816 & -0.0309042 \end{bmatrix}$$

k = 3 < 3 = n ? no → exit.

2. Итак, мы построили базис K_3 и матрицу A в этом подпространстве, которая имеет форму Хессенберга. Получаем, что

$$H = \begin{bmatrix} 2.74833 & 0.72282 & 0.08403 \\ 3.8397 & 7.83576 & -1.85547 \\ 0 & 4.35191 & -0.940593 \\ 0 & 0 & 0.147365 \end{bmatrix}; \quad Q = \begin{bmatrix} 0.78289 & -0.0324065 & -0.0603891 & -0.618373 \\ 0.62216 & 0.0407787 & 0.0759898 & 0.778126 \\ 0 & 0.203893 & 0.973267 & -0.105732 \\ 0 & 0.977606 & -0.20816 & -0.0309042 \end{bmatrix}$$

3. Далее, чтобы найти собственные значения, необходимо в матрице H удалить последнюю строку и подать полученную матрицу Хессенберга размерностью 3 на 3 QR-алгоритму.
Матрица H после QR-алгоритма имеет вид

$$\begin{bmatrix} 7.49823 & -6.12183 & 2.27266 \\ 9.04112e-07 & 1.67002 & 2.36895 \\ -7.69889e-17 & -4.77942e-07 & 0.475249 \end{bmatrix}$$

Полученные собственные значения Ритца:

(a) $\lambda_1 = 7.49823$;

(b) $\lambda_2 = 1.67002$;

(c) $\lambda_3 = 0.47525$.

Чтобы проверить правильность собственных значений, будем вычислять след матрицы, он должен равняться сумме собственных значений.

$$trH = \sum_{i=1}^{n} \lambda_i$$

Имеем $trH = 2.74833 + 7.83576 - 0.940593 = 9.643497$, $\sum\limits_{i=1}^{3} \lambda_i = 7.49823 + 1.67002 + 0.47525 = 9.6435$. След матрицы совпадает с суммой собственных значений с заданной точностью. Они действительно аппроксимируют наибольшие собственные значения матрицы A.

Пусть X_m – собственные векторы матрицы H_m, тогда приближениями собственных векторов исходной матрицы будут векторы $Y_m = Q_m \cdot X_m$, где Q_m – базис подпространства Крылова.

Соответствующие им собственные векторы:

(a) $[-0.78289 \quad -0.62216 \quad -1.10383e-07 \quad -5.29254e-07]^T$

(b) $[0.032407 \quad -0.0407784 \quad -0.203894 \quad -0.977606]^T$

(c) $[-0.0603891 \quad 0.0759897 \quad 0.973267 \quad -0.208161]^T$

Сложность итерации Арнольди. Для небольших k и преимущественно разреженных матриц стоимость итераций Арнольди составляет $O(n)$.

Сложность QR-алгоритма становится $6n^2 + O(n)$.
Т.к. алгоритм работает с разреженными большими матрицами, я подумала, что будет лучше использовать *разреженный строчный* формат хранения матриц. Матрица хранится в виде трех массивов чисел – первый массив хранит все ненулевые элементы матрицы, второй массив – номера столбцов ненулевых элементов матрицы в порядке их построчного считывания, третий массив хранит данные о том, сколько ненулевых элементов в каждой строке. Так, этот способ хранения был бы эффективным, если бы я строила базис Крылова размерности больше 3. Однако, QR-алгоритм работает очень медленно с матрицами, размерность которых больше 3, так что, данный способ хранения оказался избыточным. Так же, если бы я распараллелила QR-алгоритм, он бы так же пригодился.

3 Протокол

1. Тест 1.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$ cat test1.t
3
1 0 0
0 0 2
3 4 5
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$ ./kurs < test1.t
Введите размерность матрицы: Введите матрицу:
Матрица Хессенберга:
       0.612917        1.28686      -0.359968
        4.86177        4.67151        1.69649
              0        1.99719       0.715571
              0              0    1.50197e-07
Базис Крылова:
        0.78289      0.0623319       -0.61903       -0.78289
        0.62216     -0.0784349       0.778951       -0.62216
              0       0.994969       0.100186    2.89203e-08
Верхняя треугольная матрица A в подпространстве Крылова:
        6.27492             -2        2.85833
   -2.59549e-22       -1.27492       0.911033
   -2.34306e-16    8.20793e-07              1

Собственные значения матрицы A:
   lambda_0 = 6.27492
   lambda_1 = -1.27492
   lambda_2 = 1
Собственные векторы:
       -0.78289     -0.0623324       -0.61903
       -0.62216      0.0784355       0.778951
   -5.24689e-23      -0.994969       0.100187
Проверка:
След матрицы H: +0.612917 +4.67151 +0.715571  = +6
Сумма собственных значений : +6.27492+0 -1.27492+0 +1+0  = +6
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$
```

Полученные собственные значения :

(a) $\lambda_1 = 6.27492$;

(b) $\lambda_2 = -1.27492$;

(c) $\lambda_3 = 1$.

Проверка: $trH = \sum_{i=1}^{3} \lambda_i = 6$

2. Тест 2.

$$A = \begin{bmatrix} 1 & 0 & 10 & 5 \\ 0 & 3 & 0 & 21 \\ 1 & 30 & 0 & 6 \\ 4 & 15 & 0 & 71 \end{bmatrix}$$

```
4
1 0 10 5
0 3 0 21
1 30 0 6
4 15 0 71
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$ ./kurs < test2.t
Введите размерность матрицы: Введите матрицу:
Матрица Хессенберга:
        1.77417         15.7806         10.0761
        23.1195         24.4632         37.8509
              0         30.3065         47.7913
              0               0         11.4635
Базис Крылова:
        0.78289      -0.0262154       -0.020853       -0.621258
        0.62216       0.0329879       0.0262404        0.781754
              0        0.841181       -0.540476       -0.017354
              0         0.53911        0.840692      -0.0509676
Верхняя треугольная матрица А в подпространстве Крылова:
        75.0535        -6.64593        -8.20927
    1.66693e-24        -12.2594         10.0373
    1.54382e-17     9.96829e-07         11.2346

Собственные значения матрицы А:
   lambda_0 = 75.0535
   lambda_1 = -12.2594
   lambda_2 = 11.2346
Собственные векторы:
       -0.78289       0.0262154       -0.020853
       -0.62216      -0.0329879       0.0262404
    2.03868e-26       -0.841181       -0.540475
    1.30658e-26        -0.53911        0.840692
Проверка:
След матрицы Н: +1.77417 +24.4632 +47.7913  = +74.0287
Сумма собственных значений : +75.0535+0 -12.2594+0 +11.2346+0  = +74.0287
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$
```

Полученные собственные значения :

(a) $\lambda_1 = 75.0535$;

(b) $\lambda_2 = -12.2594$;

(c) $\lambda_3 = 11.2346$.

Проверка: $trH = \sum\limits_{i=1}^{4} \lambda_i = 74.0287$

3. Тест 3.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2$ cat test3.t
10
0 0 1 0 6 7 0 1 10 8
4 5 0 0 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 3
1 2 0 0 1 1 5 6 9 0
11 34 21 6 0 0 0 0 47 0
0 0 90 32 6 55 0 0 11 10
0 0 0 0 0 0 0 0 0 0
10 37 89 0 0 1 1 3 0 1
0 0 0 0 0 0 0 0 0 0
2 15 0 0 11 1 1 1 16 1
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2$
```

```
Матрица Хессенберга:
        3.88375        6.30921         6.7468
        44.5464        7.19845       -2.87412
              0        12.7082        34.3646
              0              0        44.5367
Базис Крылова:
        0.78289     -0.0682557       0.173961     -0.0749495
        0.62216       0.085889      -0.218902       0.094312
              0              0      0.0577537    -0.00386564
              0      0.0455078       0.361907      -0.274215
              0       0.668185      -0.186292       0.138705
              0              0       0.622579       0.775764
              0              0              0              0
              0        0.69251      -0.013179      0.0541302
              0              0              0              0
              0       0.244648       0.604173       -0.53507
Верхняя треугольная матрица A в подпространстве Крылова:
        37.7808        6.66705       -2.11078
    -7.9272e-09        15.6674       -41.2493
   -2.44486e-15    6.50089e-07       -8.00146

Собственные значения матрицы A:
   lambda_0 = 37.7808
   lambda_1 = 15.6674
   lambda_2 = -8.00146
Собственные векторы:
       -0.78289      0.0682557       0.173961
       -0.62216      -0.085889      -0.218902
              0   -4.69227e-09      0.0577537
    2.30254e-11     -0.0455079       0.361907
    3.38079e-10      -0.668185      -0.186292
              0   -5.05823e-08       0.622579
              0              0              0
    3.50387e-10       -0.69251      -0.013179
              0              0              0
    1.23784e-10      -0.244648       0.604173
Проверка:
След матрицы H: +3.88375 +7.19845 +34.3646  = +45.4468
Сумма собственных значений : +37.7808+0 +15.6674+0 -8.00146+0  = +45.4468
```

Полученные собственные значения :

(a) $\lambda_1 = 37.7808$;

(b) $\lambda_2 = 15.6674$;

(c) $\lambda_3 = -8.00146$.

Проверка: $trH = \sum\limits_{i=1}^{10} \lambda_i = 45.4468$

4. Тест 4.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2$ cat test4.t
20
32 32 54 12 52 56 8 30 44 94 44 39 65 19 51 91 1 5 89 34
25 58 20 51 38 65 30 7 20 10 51 18 43 71 97 61 26 5 57 70
65 0 75 29 86 93 87 87 64 75 88 89 100 7 40 37 38 36 44 24
46 95 43 89 32 5 15 58 77 72 95 8 38 69 37 24 27 90 77 92
31 30 80 30 37 86 33 76 21 77 100 68 37 8 22 69 81 38 94 57
76 54 65 14 89 69 4 16 24 47 7 21 78 53 17 81 39 50 22 60
93 89 94 30 97 16 65 43 20 24 67 62 78 98 42 67 32 46 49 57
60 56 44 37 75 62 17 13 11 40 40 4 95 100 0 57 82 31 0 1
56 67 30 100 64 72 66 63 18 81 19 44 2 63 81 78 91 64 91 2
70 97 73 64 97 39 21 78 70 21 46 25 54 76 92 84 47 57 46 31
38 31 75 40 61 21 84 51 86 41 19 21 37 58 86 100 97 73 44 67
60 90 58 13 31 49 63 44 73 76 76 77 73 16 83 100 4 67 51 56
7 36 77 10 95 28 10 57 0 54 23 60 9 48 39 40 97 69 84 35
44 25 11 83 8 61 83 12 27 100 34 0 35 10 10 96 39 87 53 5
40 42 66 15 90 71 55 87 39 5 88 49 97 100 32 4 60 81 83 53
80 16 53 14 94 29 77 99 16 29 3 22 71 35 4 61 6 25 13 11
30 0 27 94 66 25 64 92 5 47 44 85 29 63 65 89 59 41 87 41
36 57 29 7 92 33 34 64 59 47 76 55 13 2 48 46 27 12 37 99
25 48 83 20 77 13 9 35 55 62 76 57 18 72 64 10 4 64 74 63
77 15 18 91 84 32 36 77 10 39 75 35 87 23 22 30 37 31 65 58
```

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/СР/2/2$ ./kurs < test4.t
Введите размерность матрицы: Введите матрицу:
Матрица Хессенберга:
        69.8279         229.531         89.6703
        320.679         761.683         334.029
              0         349.984          101.06
              0               0          151.71
Базис Крылова:
        0.78289      -0.0302668         0.04303      -0.0475523
        0.62216       0.0380861      -0.0541465        0.059837
              0        0.158688        0.361251      -0.0959707
              0        0.296615      -0.0647383      -0.0280654
              0        0.133886         0.33447       0.0398652
              0         0.29031       -0.207322       0.0665354
              0        0.399717       -0.300661        0.123977
              0        0.255128       -0.183076        0.266888
              0        0.266704       0.0884987       -0.270129
              0        0.359087       -0.191087        0.110854
              0        0.152915        0.310598        0.145389
              0        0.321093      -0.0344461       -0.330199
              0       0.0869342        0.278134        0.396517
              0        0.155923         0.19444        -0.49664
              0         0.17914        0.207232        0.368469
              0         0.22635      -0.0756181       -0.244329
              0       0.0732405        0.490756      -0.0736696
              0        0.198476       0.0637224      -0.0695412
              0         0.15416        0.173777        0.256666
              0        0.217086       0.0286956         0.02515
Верхняя треугольная матрица А в подпространстве Крылова:
        986.428         -20.447        -49.8852
    8.09152e-07        -56.7657        -117.033
   -3.94158e-14     1.29046e-09         2.90833
```

```
Собственные значения матрицы A:
   lambda_0 = 986.428
   lambda_1 = -56.7657
   lambda_2 = 2.90833
Собственные векторы:
       -0.78289      0.0302668        0.04303
       -0.62216     -0.0380861     -0.0541465
    2.26197e-09      -0.158688       0.361251
    4.22802e-09      -0.296615     -0.0647383
    1.90844e-09      -0.133886        0.33447
    4.13814e-09       -0.29031      -0.207322
    5.69767e-09      -0.399717      -0.300661
    3.63666e-09      -0.255128      -0.183076
    3.80167e-09      -0.266704      0.0884987
    5.11852e-09      -0.359087      -0.191087
    2.17969e-09      -0.152915       0.310598
    4.57693e-09      -0.321093     -0.0344461
    1.23918e-09     -0.0869342       0.278134
    2.22256e-09      -0.155923        0.19444
     2.5535e-09       -0.17914       0.207232
    3.22645e-09       -0.22635     -0.0756181
    1.04399e-09     -0.0732405       0.490756
    2.82913e-09      -0.198476      0.0637224
    2.19743e-09       -0.15416       0.173777
    3.09439e-09      -0.217086      0.0286956
Проверка:
След матрицы H: +69.8279 +761.683 +101.06  = +932.571
Сумма собственных значений : +986.428+0 -56.7657+0 +2.90833+0  = +932.571
(base) vlasochka@vlasochka-VPCSB11FX:~/Documents/ЧМ/CP/2/2$
```

Полученные собственные значения :

(a) $\lambda_1 = 986.428$;

(b) $\lambda_2 = -56.7657$;

(c) $\lambda_3 = 2.90833$.

Проверка: $trH = \sum\limits_{i=1}^{20} \lambda_i = 932.571$

На этих тестах я проверила правильность выполнения программы. Собственные значения Ритца хорошо аппроксимируют наибольшие собственные значения исходной матрицы A.

4 Исходный код

Листинг 1: matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H
#include <iomanip>

typedef struct
{
    int m;
    int n;
    std::vector<double> values;
    std::vector<int> columns;
    std::vector<int> points;
} Matrix;

typedef struct Complex{
    double real;
    double imag;
} Complex;

void InitHousholder(std::vector<double>& v, Matrix* matrix){

    int n = v.size();
    matrix-> m = n;
    matrix-> n = n;
    int points = 0;
    double kaf = 0;
    double element;
    matrix-> points.push_back(points);

    for( int i = 0; i < n; i++)
        kaf += v[i] * v[i];

    for( int i = 0; i < n; i++)
    {
        for( int j = 0; j < n; j++)
        {
            if( i == j)
                element = 1 - 2 * v[i]*v[j]/kaf;
            else
                element = -2 * v[i]*v[j]/kaf;
```

```
            if( element != 0)
            {
                matrix-> values.push_back(element);
                matrix-> columns.push_back(j);
                points++;
            }
        }
        matrix-> points.push_back(points);
    }
}

void InitE( Matrix* E, int n)
{
    E-> m = n;
    E-> n = n;
    for( int i = 0; i < n; i++)
    {
        E-> values.push_back(1);
        E-> columns.push_back(i);
        E-> points.push_back(i);
    }
    E-> points.push_back(n);
}

int Sign( double value)
{
    return (value > 0) ? 1 : -1;
}

void ClearMatrix( Matrix* matrix)
{
    matrix-> values.clear();
    matrix-> columns.clear();
    matrix-> points.clear();
}

void InitMatrix( Matrix* matrix)
{
    int value;
    int count = 0;
    int m;
    std::cout << "Введите размерностьматрицы: ";
    std::cin >> m;
```

```
    matrix-> points.push_back(0);
    matrix-> m = m;
    matrix-> n = m;
    std::cout << "Введите матрицу: " << std::endl;
    for( int i = 0; i < m; i++)
    {
        for( int j = 0; j < m; j++)
        {
            std::cin >> value;
            if( value != 0)
            {
                matrix-> values.push_back(value);
                matrix-> columns.push_back(j);
                count++;
            }
        }
        matrix-> points.push_back(count);
    }
}

void InitMatrixWithMatrix( Matrix* m1, Matrix* m2)
{
    m1-> m = m2-> m;
    m1-> n = m2-> n;
    m1-> values = m2-> values;
    m1-> columns = m2-> columns;
    m1-> points = m2-> points;
}

void TransMatrix( Matrix* m)
{
    int points = 0;
    int n = m-> n;
    Matrix trans;
    trans.points.push_back(points);
    trans.m = m-> n;
    trans.n = m-> m;
    for( int i = 0; i < n; i++)
    {
        int str = 0;
        for( int j = 0; j < m-> points.size() - 1; j++)
        {
            for( int k = m-> points[j]; k < m-> points[j + 1]; k++)
            {
```

```
                if( m-> columns[k] == i)
                {
                    trans.values.push_back(m-> values[k]);
                    trans.columns.push_back(str);
                    points++;
                }
            }
            str++;
        }
        trans.points.push_back(points);
    }
    ClearMatrix(m);
    m-> m = trans.m;
    m-> n = trans.n;
    m-> values = trans.values;
    m-> columns = trans.columns;
    m-> points = trans.points;
}

void TransMatrix( Matrix* m, Matrix* trans)
{
    int points = 0;
    int n = m-> n;
    trans-> points.push_back(points);
    trans-> m = m-> n;
    trans-> n = m-> m;
    for( int i = 0; i < n; i++)
    {
        int str = 0;
        for( int j = 0; j < m-> points.size() - 1; j++)
        {
            for( int k = m-> points[j]; k < m-> points[j + 1]; k++)
            {
                if( m-> columns[k] == i)
                {
                    trans-> values.push_back(m-> values[k]);
                    trans-> columns.push_back(str);
                    points++;
                }
            }
            str++;
        }
        trans-> points.push_back(points);
    }
```

```
}

void PrintMatrix( Matrix* matrix)
{
    int id = 0;
    for( int i = 0; i < matrix-> points.size()-1; i++)
    {
        for( int j = matrix-> points[i]; j < matrix-> points[i+1]; j++)
        {
            for( int k = id; k < matrix-> columns[j]; k++)
                std::cout << std::setw(15) << std::right << "0";
            std::cout << std::setw(15) << std::right << matrix-> values[j];
            id = matrix-> columns[j] + 1;
        }

        for( int j = id; j < matrix-> n; j++)
            std::cout << std::setw(15) << std::right << "0";
        id = 0;
        std::cout << std::endl;
    }
}

void MultMatVec( Matrix* m, std::vector<double>& v, std::vector<double>& res)
{
    double element = 0;
    int points = 0;
    for( int i = 0; i < m-> points.size() - 1; i++)
    {
        for( int k = m-> points[i]; k < m-> points[i + 1]; k++)
                element += m-> values[k] * v[m-> columns[k]];
        res.push_back(element);
        element = 0;
    }
}

void MultMatrix( Matrix* m1, Matrix* m2, Matrix* res)
{
    double element = 0;
    int points = 0;
    Matrix trans;
    TransMatrix(m2, &trans);
    res-> points.push_back(points);
    res-> m = m1-> m;
    res-> n = trans.m;
```

```
    for( int i = 0; i < m1-> points.size() - 1; i++)
    {
        for( int j = 0; j < trans.points.size() - 1; j++)
        {
            for( int k = m1-> points[i]; k < m1-> points[i + 1]; k++)
            {
                for( int l = trans.points[j]; l < trans.points[j + 1]; l++)
                {
                    if( m1-> columns[k] == trans.columns[l])
                        element += m1-> values[k] * trans.values[l];
                }
            }
            if( element != 0)
            {
                res-> values.push_back(element);
                res-> columns.push_back(j);
                points++;
                element = 0;
            }
        }
        res-> points.push_back(points);
    }
}

void MultMatrix( Matrix* m1, Matrix* m2)
{
    double element = 0;
    int points = 0;
    Matrix trans;
    TransMatrix(m2, &trans);
    ClearMatrix(m2);
    m2-> m = m1-> m;
    m2-> points.push_back(points);
    for( int i = 0; i < m1-> points.size() - 1; i++)
    {
        for( int j = 0; j < trans.points.size() - 1; j++)
        {
            for( int k = m1-> points[i]; k < m1-> points[i + 1]; k++)
            {
                for( int l = trans.points[j]; l < trans.points[j + 1]; l++)
                {
                    if( m1-> columns[k] == trans.columns[l])
                        element += m1-> values[k] * trans.values[l];
                }
```

```
            }
            if( element != 0)
            {
                m2-> values.push_back(element);
                m2-> columns.push_back(j);
                points++;
                element = 0;
            }
        }
        m2-> points.push_back(points);
    }
}

void SetDim(Matrix* matrix, int m, int n)
{
    matrix-> m = m;
    matrix-> n = n;
    matrix-> points.push_back(0);
}

void RemRow( Matrix* matrix)
{
    matrix-> m = matrix-> m - 1;
    int shift = matrix-> points[matrix-> points.size() - 1] - matrix-> points[matrix-> points.size() - 2];
    matrix-> points.pop_back();
    while( shift > 0)
    {
        matrix-> values.pop_back();
        matrix-> columns.pop_back();
        shift--;
    }
}

void SetRow( Matrix* matrix, std::vector<double>& v)
{
    int n = v.size();
    int points = matrix-> points[matrix-> points.size() - 1];
    for( int i = 0; i < n; i++)
    {
        if(v[i] != 0)
        {
            matrix-> values.push_back(v[i]);
            matrix-> columns.push_back(i);
```

```
                points++;
            }
        }
        matrix-> points.push_back(points);
}

double Norma( std::vector<double>& v)
{
        double norma = 0;
        for( int i = 0; i < v.size(); i++)
            norma += pow(v[i], 2);
        return pow(norma, 0.5);
}

void Normalization( std::vector<double>& from, double norma, std::vector<double>& to)
{
        for(int i = 0; i < from.size(); i++)
            to[i] = from[i] / norma;
}

void GetComplexSV( Matrix* A, std::vector<Complex>& Lyambda, int i)
{
        double Disc = pow(A-> values[A-> points[i + 1] + i + 1] + A-> values[A-> points[i] + i], 2) - 4*(A-> values[A-> points[i] + i] * A-> values[A-> points[i + 1] + i + 1] - A-> values[A-> points[i] + i + 1] * A-> values[A-> points[i + 1] + i]);
        Lyambda[i].real = (A-> values[A-> points[i + 1] + i + 1] + A-> values[A-> points[i] + i ])/2;
        Lyambda[i].imag = pow(fabs(Disc), 0.5)/2;
        Lyambda[i+1].real = Lyambda[i].real;
        Lyambda[i+1].imag = -Lyambda[i].imag;
}

#endif
```

Листинг 2: arnoldi.cpp

```
#include <iostream>
#include <math.h>
#include <vector>
#include <iterator>
#include "matrix.h"

void Housholder( Matrix* R, Matrix* Q)
```

```
{
    int n = R-> n;
    int m = R-> m;
    ClearMatrix(Q);
    InitE(Q, n);
    for( int i = 0; i < m - 1; i++)
    {
        std::vector<double> v(n, 0);
        double sum = 0;
        for(int j = R-> points[i]; j < R-> points[i + 1]; j++)
        {
            if( R-> columns[j] == i)
            {
                sum += pow( R-> values[j], 2);
                v[i] = R-> values[j];
            }
            else if( R-> columns[j] == i + 1)
            {
                sum += pow( R-> values[j], 2);
                v[i] += Sign(R-> values[j-1])*pow(sum, 0.5);
                v[i + 1] = R-> values[j];
            }
        }
        Matrix Housholder;
        InitHousholder(v, &Housholder);
        TransMatrix(R);
        MultMatrix(&Housholder, R);
        MultMatrix(Q, &Housholder);
        InitMatrixWithMatrix(Q, &Housholder);
        if( i == m - 2)
        {
            Matrix A_new;
            MultMatrix(R, Q, &A_new);
            InitMatrixWithMatrix(R, &A_new);
        }
        TransMatrix(R);
    }
}

Matrix QR_algorithm( Matrix* A)
{
    Matrix Q;
    int it_count = 0;
    double kr = 1;
```

```
double eps = 1e-10;
int n = A-> m;
std::vector<Complex> Lyambda(n, {0, 0});
TransMatrix(A);
while(kr > eps or it_count < 500)
{
    Housholder(A, &Q);
    for( int i = 0; i < n; i++ )
    {
        kr = 0;
        for(int j = A-> points[i]; j < A-> points[i + 1]; j++)
            if( A-> columns[j] > i)
                kr += pow(A-> values[j], 2);
        kr = pow(kr, 0.5);

        if(kr <= eps)
        {
            Lyambda[i].real = 0;
            Lyambda[i].imag = 0;

            for( int k = A-> points[i]; k < A-> points[i + 1]; k++)
                if( A-> columns[k] == i )
                {
                    Lyambda[i].real = A-> values[k];
                    break;
                }
            if( i == 1)
                it_count += 500;
        }
        else if( i == 0)
            break;
        else if(i != n - 1)
        {
            double real = Lyambda[i].real;
            double img = Lyambda[i].imag;
            GetComplexSV(A, Lyambda, i);
            kr = pow(pow(real - Lyambda[i].real, 2) + pow(img - Lyambda[i].imag, 2), 0.5);
            i++;
            if(kr > eps)
                break;
        }
    }
    it_count ++;
```

```
        }
        if( it_count > 501)
            it_count -= 500;
        TransMatrix(A);
        std::cout << "Верхняя треугольнаяматрицаA вподпространствеКрылова: " << std::endl;
        PrintMatrix(A);
        std::cout << std::endl;
        std::cout << "Собственные значенияматрицыA:" << std::endl;
        for(int i = 0; i < n; i++)
            if(Lyambda[i].imag == 0)
                std::cout << std::setw(10) << std::right << "lambda_" << i << " = " <<
                    Lyambda[i].real << std::endl;
            else
            {
                std::cout << std::setw(10) << std::right << "lambda_" << i << " = " <<
                    Lyambda[i].real;
                std::cout<< std::showpos << Lyambda[i].imag << "i" << std::endl;
            }
        return Q;
}

void ArnoldiIteration(Matrix* A)
{
        int dim = 3, points = 0;
        double eps = 1e-10;
        Matrix Q, H;
        std::vector<double> q(A-> m, 0);
        q[0] = 0.78289;
        q[1] = 0.62216;
        SetDim(&Q, dim + 1, A-> n);
        SetDim(&H, dim, dim + 1);
        SetRow(&Q, q);
        for( int i = 0; i < dim; i++)
        {
            std::vector<double> v;
            MultMatVec(A, q, v);
            for( int j = 0; j < i + 1; j++)
            {
                double element = 0;
                for( int k = Q.points[j]; k < Q.points[j + 1]; k++)
                    element += Q.values[k] * v[Q.columns[k]];
                if(element != 0){
                    H.values.push_back(element);
                    H.columns.push_back(j);
```

```
                points++;
            }
            for( int k = Q.points[j]; k < Q.points[j + 1]; k++)
                v[Q.columns[k]] -= element * Q.values[k];
        }
        double norma = Norma(v);
        if( norma != 0)
        {
            H.values.push_back(norma);
            H.columns.push_back(i+1);
            points++;
        }
        H.points.push_back(points);
        if( norma > eps)
        {
            Normalization(v, norma, q);
            SetRow(&Q, q);
        }
        else break;
    }
    Matrix TQ;
    TransMatrix(&Q, &TQ);
    RemRow(&TQ);
    TransMatrix(&TQ);
    TransMatrix(&Q); //Basis m(n+1)
    TransMatrix(&H); //Hessenberg (n+1)n
    std::cout << "Матрица Хессенберга: " << std::endl;
    PrintMatrix(&H);
    std::cout << "Базис Крылова: " << std::endl;
    PrintMatrix(&Q);
    RemRow(&H);
    Matrix SV = QR_algorithm(&H); //Nortmal Hessenberg matrix with removed last row

    MultMatrix(&Q, &SV);
    MultMatrix(&SV, &TQ);
    std::cout << "Собственные векторыпослепреобразованияподобия: " << std::endl;
    PrintMatrix(&TQ);
}

int main(void)
{
    Matrix matrix;
    InitMatrix(&matrix);
    ArnoldiIteration(&matrix);
```

```
    ClearMatrix(&matrix);
    return 0;
}
```

5 Выводы

При выполнении данной курсовой работы я познакомилась с подпространством Крылова. На построении ортонормированного базиса этого подпространства основан метод Арнольди, позволяющий аппроксимировать наибольшие собственные значения матрицы большой размерности. Метод строит матрицу Хессенберга, которая является проекцией исходной матрицы на подпространство Крылова. Ее размерность может быть задана, таким образом, будут аппроксимированы только некоторые собственные значения. В случае задания небольшой размерности подпространства и разреженной входной матрицы, итерация Арнольди будет иметь сложность $O(n)$. Полученные собственные значения Ритца совпадают с собственным значениями исходной матрицы. Они вычисляются с помощью $QR-$алгоритма, на вход котором подается матрица Хессенберага, что несколько ускоряет сходимость алгоритма, т.к. рассматриваются ненулевые значения диагонали и под диагонали. Сложность QR-алгоритма становится $6n^2 + O(n)$ В дальнейшем, чтобы аппроксимировать больше собственных значений, стоит распараллелить $QR-$ алгоритм. Так же, хочется заметить, что метод Арнольди все еще находится на стадии изучения и теория по нему не является полной.