**Московский авиационный институт**

**(национальный исследовательский университет)**

Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

**Курсовая работа по**

**численным методам по теме**

**«Нахождение собственных значений и собственных векторов несимметричных разреженных матриц большой размерности. Метод Арнольди»**

Студент Аксенов А.Е.

Группа: М8О-406Б-19

Руководитель: Ревизников Д.Л.

Оценка:

Дата: 25.11.2022

*Москва*

*2022 г*

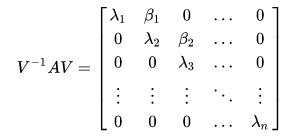
**Собственные значения и собственные векторы**

Если задана *n* × *n* квадратная матрицаА над вещественными или комплексными числами, собственное значение λ и соответствующий ему корневой вектор **v** — это пара, удовлетворяющая равенству , где **v** ненулевой *n* × *1* вектор-столбец, *E* является *n* × *n* единичной матрицей, *k* — положительным целым, а λ и **v** могут быть комплексными, даже если *A* вещественна. Если *k* = 1, вектор просто называется собственным вектором. В этом случае *A***v** = λ v. Любое собственное значение λ матрицы *A* имеет простой собственный вектор, соответствующий ему так, что если *k* — наименьшее целое, при котором для корневого вектора **v**, то будет простым собственным вектором. Значение *k* всегда можно взять меньше либо равным *n*. В частности, для всех корневых векторов **v** соответствующих λ. Для любого собственного значения λ матрицы *A* ядро состоит из всех собственных векторов, соответствующих λ (вместе с 0) и называется *собственным подпространством* числа λ, а векторное подпространство состоит из всех корневых векторов (дополненное нулевым вектором) и называется *корневым подпространством*. *Геометрическая кратность* значения λ является размерностью его собственного подпространства. *Алгебраическая кратность* значения λ является размерностью его корневого подпространства. Дальнейшие термины связаны с равенством

Здесь det — определитель, λ - все различные собственные значения матрицы *A*, а — соответствующие алгебраические кратности. Функция — это *характеристический многочлен* матрицы *A*. Таким образом, алгебраическая кратность является кратностью собственных значений как корней характеристического многочлена. Поскольку любой собственный вектор является корневым вектором, геометрическая кратность меньше либо равна алгебраической кратности. Сумма алгебраических кратностей равна n степени характеристического многочлена. Уравнение называется *характеристическим уравнением*, поскольку его корни являются в точности собственными значениями матрицы *A*. По теореме Гамильтона — Кэли сама матрица *A* удовлетворяет тому же самому уравнению:. Как следствие, столбцы матрицы должны быть либо 0, либо корневыми векторами, соответствующими собственному значению , поскольку они уничтожаются матрицей . Любой набор корневых векторов различных собственных значений линейно независим, так что базис для всего можно выбрать из набора корневых векторов. Точнее этот базис можно выбрать и выстроить так, что:

* Eсли **v***i* и **v***j* имеют одно и то же собственное значение, то тоже будет верно для любого **v***k* при *k* между *i* и *j*;
* Eсли **v***i* не является простым собственным вектором и если λ*i* его собственное значение, то (в частности **v**1 должен быть простым собственным вектором).

Если эти базисные вектора расположить как столбцы матрицы то *V* можно использовать для преобразования матрицы *A* в её нормальную жорданову форму:



где λ*i* — собственные значения, *βi* = 1 если (*A* - λ*i*+1)**v***i*+1 = **v***i* и *βi* = 0 в других случаях. Если *W* является обратимой матрицей и λ — собственное значение матрицы *A* с соответствующим корневым вектором **v**, то (*W* -1*AW* - λ*E* )*k* *W* -*k***v** = 0. Таким образом, λ является собственным значением матрицы *W* -1*AW* с соответствующим корневым вектором *W* -*k***v**. Таким образом, подобные матрицы имеют те же самые собственные значения.

**Итерация Арнольди**

В численной линейной алгебре **итерация Арнольди** является алгоритмом вычисления собственных значений. Арнольди находит приближение собственных значений и собственных векторов матриц общего вида (возможно не эрмитовой) с помощью построения ортонормированного базиса подпространства Крылова.

Метод Арнольди относится к алгоритмам линейной алгебры, которые позволяют получить частичное решение после небольшого количества итераций, в отличие от так называемых прямых методов, которые должны полностью завершиться для получения каких-либо удовлетворительных результатов (например отражения Хаусхолдера).

Если алгоритм применяется на эрмитовых матрицах, то он сводится к алгоритму Ланцоша. Итерация Арнольди была придумана Уолтером Эдвином Арнольди в 1951 г.

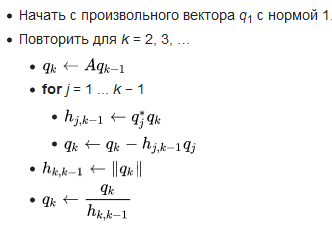
**Подпространство Крылова и степенной метод**

Интуитивным методом нахождения наибольшего (по модулю) собственного значения данной матрицы A {\displaystyle A} mxn размерами является степенной метод: начать с произвольного начального вектора b вычислять , нормируя результат после каждого вычисления. Эта последовательность сходится к собственному вектору соответствующего собственного значения с максимальным модулем. Это наводит на мысль, что много вычислений тратится впустую, т.к. в итоге используется лишь конечный результат . Тогда давайте вместо этого составим так называемую матрицу Крылова:

A b , A 2 b , A 3 b , … {\displaystyle Ab,A^{2}b,A^{3}b,\dots } m × m {\displaystyle m\times m}. Столбцы этой матрицы в общем случае не являются ортогональными, но мы можем получить из них ортогональный базис с помощью ортогонализации Грама-Шмидта. Полученное множество векторов будет являться ортогональным базисом подпространства Крылова **K***n*. Можно ожидать, что вектора этого базиса будут хорошим приближением к векторам, соответствующим n наибольшим по модулю собственным значениям.

**Итерация Арнольди**

Итерация Арнольди использует стабилизированный процесс Грама-Шмидта для получения последовательности ортонормированных векторов , называемых *векторами Арнольди*, таких, что для каждого n векторы являются базисом подпространства Крылова **K***n*. Алгоритм выглядит следующим образом:



Цикл по *j {\displaystyle j} j* проецирует компоненту на . Это обеспечивает ортогональность всех построенных векторов. Алгоритм останавливается, когда является нулевым вектором. Это происходит, когда минимальный многочлен матрицы A будет степени *k*. Каждый шаг цикла по *k* производит одно умножение матрицы на вектор и около 4*mk* операций с дробными числами.

**Код программы**

import numpy as np

import matplotlib.pyplot as plt

from scipy.linalg import circulant

import numpy as np

def arnoldi\_iteration(A, b, n):

    m = A.shape[0]

    h = np.zeros((n + 1, n), dtype=np.complex)

    Q = np.zeros((m, n + 1), dtype=np.complex)

    q = b / np.linalg.norm(b)  # Нормализация входного вектора

    Q[:, 0] = q  # Использование его как первого вектора Крылова

    for k in range(n):

        v = A.dot(q)  # Генерация нового вектора

        for j in range(k + 1):

            h[j, k] = np.dot(Q[:, j], v)

            v = v - h[j, k] \* Q[:, j]

        h[k + 1, k] = np.linalg.norm(v)

        eps = 1e-12

        if h[k + 1, k] > eps:  # Добавление полученного вектора в список, если только

            q = v / h[k + 1, k]  # получается нулевой вектор.

            Q[:, k + 1] = q

        else:  # Если это произойдет. Прекратить повторять

            return Q, h

    return Q, h

# Построение матрицы А

N = 2\*\*4

I = np.eye(N)

k = np.fft.fftfreq(N, 1.0 / N) + 0.5

alpha = np.linspace(0.1, 1.0, N)\*2e2

c = np.fft.fft(alpha) / N

C = circulant(c)

A = np.einsum("i, ij, j->ij", k, C, k)

# Показываем, что A является эрмитовым

print(np.allclose(A, A.conj().T))

# Произвольный (случайный) начальный вектор

np.random.seed(0)

v = np.random.rand(N)

# Выполнение итерации Арнольди с комплексным А

\_, h = arnoldi\_iteration(A, v, N)

# Выполнение итерации Арнольди с вещественным A

\_, h2 = arnoldi\_iteration(np.real(A), v, N)

print("Arnoldi iteration with complex A = " , h)

print("Arnoldi iteration with real A = " , h2)

# Построение результатов.

plt.subplot(121)

plt.imshow(np.abs(h))

plt.title("Complex A")

plt.subplot(122)

plt.imshow(np.abs(h2))

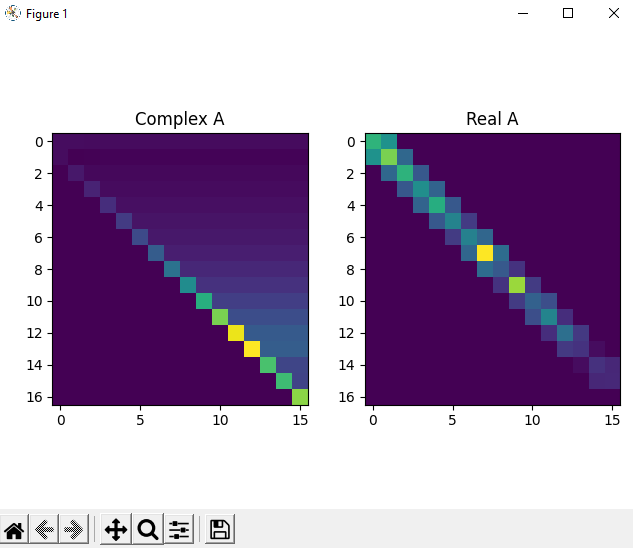
plt.title("Real A")

plt.tight\_layout()

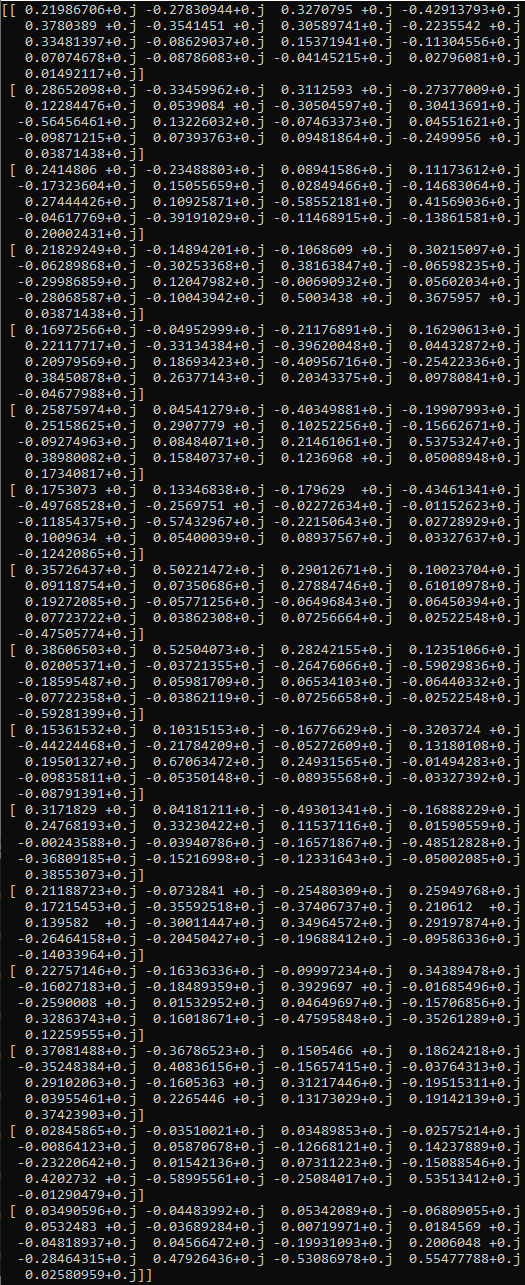
plt.show()

**Результат работа программы**

Результатом работы программы является визуализация верхней матрицы Хейссенберга. Чем значение ближе к среднему тем более ярко оно отображается.



А так же результат работы программы: множество векторов которое является ортогональным базисом подспространства Крылова. Вектора этого базиса являются хорошим приближением к векторам, соответствующим n большим по модулю собственным значениям:



import numpy as np

import matplotlib.pyplot as plt

from scipy.linalg import circulant

import numpy as np

def arnoldi\_iteration(A, b, n):

m = A.shape[0] # получение размерности (размеров в виде (n, m)) матрицы А

h = np.zeros((n + 1, n), dtype=np.complex) # создание нулевой матрицы размеров n+1 на n с комплексными типом данных

Q = np.zeros((m, n + 1), dtype=np.complex) # создание нулевой матрицы размеров m на n+1 с комплексным типом данных

q = b / np.linalg.norm(b) # Нормализация входного вектора

Q[:, 0] = q # Использование его как первого вектора Крылова

for k in range(n):

v = A.dot(q) # Генерация нового вектора

for j in range(k + 1):

h[j, k] = np.dot(Q[:, j], v)

v = v - h[j, k] \* Q[:, j]

h[k + 1, k] = np.linalg.norm(v) #вычисление нормы матрицы

eps = 1e-12

if h[k + 1, k] > eps: # Добавление полученного вектора в список, если только

q = v / h[k + 1, k] # получается нулевой вектор.

Q[:, k + 1] = q

else: # Если это произойдет. Прекратить повторять

return Q, h

return Q, h

# Построение матрицы А

N = 2\*\*4 # задаём размерность матрицы А

I = np.eye(N) # создаём единичную матрицу размерности N

k = np.fft.fftfreq(N, 1.0 / N) + 0.5 # возвращаем плавающий массив центров частотных бинов в циклах за единицу времени выборки (длина окна 16, интервал выборки 1/16) плюс (0.5?)

alpha = np.linspace(0.1, 1.0, N)\*2e2 # создаём одномерный массив из N элементов равномерно распределённых на интервале [0.1, 1.0], умноженный на (200?)

c = np.fft.fft(alpha) / N # производим Быстрое Преобразование Фурье над массивом alpha, результат делим на размерность матрицы А

C = circulant(c) # строим циркулянтную матрицу из элементов массива c

A = np.einsum("i, ij, j->ij", k, C, k) # получаем произведение (в скобочках означают: i{j} - одномерные массивы из строк{столбцов}, ij - двумерный массив)

# Показываем, что A является эрмитовым

print(np.allclose(A, A.conj().T))

# Произвольный (случайный) начальный вектор

np.random.seed(0)

v = np.random.rand(N) # заполнение случайными числами из равномерного распределения [0, 1) матрицы размерности N

# Выполнение итерации Арнольди с комплексным А

\_, h = arnoldi\_iteration(A, v, N)

# Выполнение итерации Арнольди с вещественным A

\_, h2 = arnoldi\_iteration(np.real(A), v, N)

print(\_)

print("Arnoldi iteration with complex A =", h)

print("Arnoldi iteration with real A = ", h2)

# Построение результатов.

plt.subplot(121)

plt.imshow(np.abs(h))

plt.title("Complex A")

plt.subplot(122)

plt.imshow(np.abs(h2))

plt.title("Real A")

plt.tight\_layout()

plt.show()

Индус:

1) Начинаем с матрицы А

2) Умножаем матрицу А на произвольный вектор b (получаем вектор c = A\*b)

3) Снова умножаем вектор c (A\*b) на матрицу А (получаем A\*b^2)

4) Снова умножаем на матрицу А спереди (получаем A\*b^3)

5) Повторяем процедуру умножения, пока не получим A\*b^n, которое будет сходиться к наибольшему собственному вектору матрицы А

6) Для стабильности метода нам нужно гарантировать, что каждый из векторов от A\*b до A\*b^n ортогонален всем предыдущим векторам (нам это известно как "операция Арнольди")

Непосредственно смысл опреции Арнольди:

7) используем ортогонализацию Грамма-Шмидта (гарантирующую нам ортогональность векторов)

8) берем произвольный вектор q\_1 с нормой 1 (||q\_1|| = 1) для наличия ортонормированного базиса

9) q\_(k) = A\*q\_(k-1)

10) нам нужно найти всех j принадлежащих [1, k-1] ...

11) ... найти вектор h\_(j,k-1), равный скалярному произведению q\_j^T и q\_k (т.е. q\_j^T \* q\_k), где q\_k - скаляр, определяющий длину проекции на вектор j

12) тогда q\_k станет равным q\_k - q\_j^T \* q\_k \* q\_j (мы вычитаем длину в направлении j). Другими словами: q\_k = q\_k - h\_(j,k-1) \* q\_j

13) затем мы нормализуем наш q\_k по завершении цикла (т.е. когда достигнем h\_(k, k-1)). Другими словами: h\_(k, k-1) = ||q\_k||

14) наконец мы можем масштабировать наш q\_k, чтобы он был нормальным, приняв q\_k = q\_k / h\_(k, k-1)