Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовой проект

по курсу «Численные методы»

Студент: Сыроежкин К. Г.
Группа: 80-304б
Преподаватель: Гидаспов В.Ю.
Оценка:

Москва, 2021

1. Постановка задачи:

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

2. Теория:

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 размером m x m является степенной метод: начать с произвольного начального вектора b вычислить $Ab, A^2b, A^3b, \ldots$, нормируют результат после каждого вычисления.

Эта последовательность сходится к собственному вектору соответствующего собственного значения λ_1 с максимальным модулем. Это наводит на мысль, что много вычислений тратится впустую, т.к. в итоге используется лишь конечный результат $A^{n-1}b$.

Тогда давайте вместо этого составим так называемую *матрицу Крылова*:

$$K_n = [b \quad Ab \quad A^2b \quad \cdots \quad A^{n-1}b].$$

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* Можно ожидать, что вектора этого базиса будут хорошим приближением к векторам, соответствующим n наибольшим по модулю K_n ственным значениям.

Итерация Арнольди использует стабилизированный процесс Грама-Шмидта для получения последовательности ортонормированных векторов называем $q_1, q_2, q_3, \ldots$ *ими Арнольди*, таких, что для каждого n векторы являются базисом $q_1, \ldots, q_n$ транства Крылова. Алгоритм выглядит следующим образом:

- Начать с произвольного вектора q_1 с нормой 1.
- Повторить для $k = 2, 3, \ldots$
  - $q_k \leftarrow A q_{k-1}$
  - **for** $j = 1 \ldots k-1$
    - $h_{j,k-1} \leftarrow q_j^* q_k$
    - $q_k \leftarrow q_k - h_{j,k-1} q_j$
  - $h_{k,k-1} \leftarrow \|q_k\|$
  - $q_k \leftarrow \frac{q_k}{h_{k,k-1}}$

Цикл по j проецирует компоненту q_k на $q_1, \ldots, q_{k-1}$ Это обеспечивает ортогональность всех построенных векторов. Алгоритм останавливается, когда q_k 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 4mk операций с дробными числами.

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.

3. Код программы

```
def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[:-1]))
            mat.append(row)
    return np.array(mat)


def arnoldi_iteration(A, b, n: int):
    m = A.shape[0]
    h = np.zeros((n + 1, n))
    Q = np.zeros((m, n + 1))
    q = b / np.linalg.norm(b)
    Q[:, 0] = q
    for k in range(n):
        v = A.dot(q)
        for j in range(k + 1):
            h[j, k] = np.dot(Q[:, j].conj(), v)
            v = v - h[j, k] * Q[:, j]

        h[k + 1, k] = np.linalg.norm(v)
        eps = 0.001
        if h[k + 1, k] > eps:
            q = v / h[k + 1, k]
            Q[:, k + 1] = q
        else:
            return Q[:,0:n], h[0:n,:]
    return Q[:,0:n], h[0:n,:]


def get_mat(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(int, line.split(',')[:-1]))
            mat.append(row)
    return np.array(mat)

def get_v(column, size, k):
    v = np.zeros(size)
    v[k] = column[k]+np.sign(column[k])*np.sqrt(np.sum(column[k:]*column[k:]))
    for i in range(k+1, size):
        v[i] = column[i]
    return v


def get_H(column, size, k):
    v = get_v(column,size,k)[:, np.newaxis]
```

```
    return np.eye(size) - 2*np.dot(v,v.T)/np.dot(v.T,v)


def get_A(H,A0):
    return np.dot(H,A0)

def get_QR(A):
    size = A[:,1].size
    Q = np.eye(size)
    for i in range(size-1):
        H = get_H(A[:,i],A[:,i].size,i)
        A = get_A(H,A)
        Q = np.dot(Q,H)
    return Q,A

def solve_equation(A, i):
    size = A[:,1].size
    if i+1 < size:
        a12 = A[i][i + 1]
        a21 = A[i + 1][i]
        a22 = A[i + 1][i + 1]
    else:
        a12 = 0
        a21 = 0
        a22 = 0
    a11 = A[i][i]
    a = 1
    b = -a11 - a22
    c = a11 * a22 - a12 * a21
    D = b*b-4*a*c
    if D < 0:
        l1 = (-b+np.sqrt(complex(D)))/2
        l2 = (-b-np.sqrt(complex(D)))/2
    else:
        l1 = (-b+np.sqrt(D))/2
        l2 = (-b-np.sqrt(D))/2
    return np.array([l1,l2])


def get_eigvectors(A,k):
    A_k = A
    Q_k = np.eye(A.shape[1])
    for k in range(k):
        Q, R = qr(A_k)
        Q_k = Q_k.dot(Q)
        A_k = R.dot(Q)
    return Q_k

def have_complex_lambda(A, eps, i):
        Q, R = get_QR(A)
        A1 = np.dot(R,Q)
```

```
        lambda1 = solve_equation(A, i)
        lambda2 = solve_equation(A1, i)
        if np.all(abs(lambda1 - lambda2) <= eps) and isinstance(lambda1[0],
complex) and isinstance(lambda2[0], complex):
            return True
        else:
            return False

def get_eig(A,eps,i,k):
    while True:
        Q, R = get_QR(A)
        A = np.dot(R,Q)
        k+=1
        if np.sqrt(np.sum(A[i + 1:, i]*A[i + 1:, i])) <= eps:
            return A[i,i], False, A, k
        elif np.sqrt(np.sum(A[i + 2:, i]*A[i + 2:, i])) <= eps and
have_complex_lambda(A, eps, i):
            return solve_equation(A,i), True, A, k

def QR(A, eps):
    eigs = []
    i = 0
    k = 0
    c = 0
    size = A[:,1].size
    while i < size:
        eig = get_eig(A,eps,i,k)
        k = eig[3]
        if eig[1]:
            eigs.append(eig[0])
            i+=2
        else:
            eigs.append(eig[0])
            i+=1
        A = eig[2]
    return eigs,k



def QR_arnoldi(A, eps):
    eigs = []
    i = 0
    k = 0
    c = 0
    size = A[:,1].size
    _, A = arnoldi_iteration(A,np.ones(len(A)),len(A))
    while i < size:
        eig = get_eig(A,eps,i,k)
        k = eig[3]
        if eig[1]:
            eigs.append(eig[0])
```

```
            i+=2
        else:
            eigs.append(eig[0])
            i+=1
        A = eig[2]
    return eigs,k

A = 10*np.random.random((7,7))+5
B = A.copy()
print("A:\n", A)
eigs,k = QR(A,0.1)
print("Количество итераций:\n", k)
print("Собственные значения:\n", eigs)
print("Наибольший собственные вектор:\n", get_eigvectors(A,k)[:,0])

print("A:\n", B)
eigs,k = QR_arnoldi(A,0.1)
print("Количество итераций:\n", k)
print("Собственные значения:\n", eigs)
print("Наибольший собственные вектор:\n", get_eigvectors(A,k)[:,0])
A = 10*np.random.random((50,50))+5
B = A.copy()
print("A:\n", A)
eigs,k = QR(A,0.1)
print("Количество итераций:\n", k)
print("Собственные значения:\n", eigs)
print("Наибольший собственные вектор:\n", get_eigvectors(A,k)[:,0])
print("A:\n", B)
eigs,k = QR_arnoldi(A,0.1)
print("Количество итераций:\n", k)
print("Собственные значения:\n", eigs)
print("Наибольший собственные вектор:\n", get_eigvectors(A,k)[:,0])
```

4. Результат

Для матрицы размерностью 7 на 7 количество итераций без Арнольди составляет 25 и занимает 16ms . Если же использовать метод Арнольди, то количество итераций составляет 19 и занимает 13 ms

```
Простой QR
A:
 [[13.35706205  7.53886042 14.28980853  7.12985633  9.70496521  6.79385841
  10.73783776]
 [ 5.54727917  9.40253279  8.13685633 13.79883253 12.6217821   8.29121793
  11.95487998]
 [14.4799896   8.40168138  9.89108886  7.09544624 13.14510152 11.32873348
   5.78792737]
 [ 5.5907856   9.61656094  7.49728912  5.84605454 13.20307864 13.66872108
```

```
  11.83067892]
 [ 9.96335357  7.26022014  5.91745379  5.00084691  8.5460817  10.71353009
  13.80229811]
 [ 8.57402786  6.56114798 14.11421971 12.26448314 12.90039806  6.25406176
  13.17045603]
 [14.86943708  9.84276166  9.72761468  6.09257472 10.32087455  8.79892079
   6.53662349]]
Количество итераций:
 25
Собственные значения:
 [68.07352327732126, -10.216312329186234, 6.771643641895924, -3.977043535322688,
array([0.60190431+2.85207658j, 0.60190431-2.85207658j]), -2.034302803041191]
Наибольший собственные вектор:
 [-0.38554532 -0.38311707 -0.38889821 -0.37176961 -0.33944084 -0.40584638
 -0.36768231]
Wall time: 16 ms
Метод Арнольда
A:
 [[13.35706205  7.53886042 14.28980853  7.12985633  9.70496521  6.79385841
  10.73783776]
 [ 5.54727917  9.40253279  8.13685633 13.79883253 12.6217821   8.29121793
  11.95487998]
 [14.4799896   8.40168138  9.89108886  7.09544624 13.14510152 11.32873348
   5.78792737]
 [ 5.5907856   9.61656094  7.49728912  5.84605454 13.20307864 13.66872108
  11.83067892]
 [ 9.96335357  7.26022014  5.91745379  5.00084691  8.5460817  10.71353009
  13.80229811]
 [ 8.57402786  6.56114798 14.11421971 12.26448314 12.90039806  6.25406176
  13.17045603]
 [14.86943708  9.84276166  9.72761468  6.09257472 10.32087455  8.79892079
   6.53662349]]
Количество итераций:
 19
Собственные значения:
 [68.07399575433672, -10.191026997457477, 6.788550300464315, -
3.9721666538358473, array([0.59275397+2.85232198j, 0.59275397-2.85232198j]), -
2.037890430152606]
Наибольший собственные вектор:
 [-0.38554532 -0.38311707 -0.38889821 -0.37176961 -0.33944084 -0.40584638
 -0.36768231]
Wall time: 13 ms
```

Для матрицы размерностью 50 на 50 количество итераций без Арнольди составляет 10990 и занимает 1m 13s. Если же использовать метод Арнольди, то количество итераций составляет 1513 и занимает 16.6s

```
Простой QR
```

```
A:
 [[10.72359075 11.13393155  5.45011535 ...  5.6824868  11.99036914
  14.87507716]
 [ 5.90310659 11.80042802 11.16954808 ...  7.40854133  5.68148828
  10.83780643]
 [11.39337208 12.24684578 14.25807987 ...  5.36864195 14.512801
  14.63870507]
 ...
 [ 5.22989387 10.02020103 14.24603303 ... 10.03507778  5.27834598
   8.61464961]
 [13.21367351  8.10398975 12.7823375  ...  9.29827281  6.62171877
   9.45570276]
 [10.82619811  5.80370336 10.74201628 ... 12.60582502 10.94879985
  10.22128505]]
Количество итераций:
 10990
Собственные значения:
 [498.72786609345746, array([7.63767745+17.71374686j, 7.63767745-17.71374686j]),
array([-17.89997941+6.86955109j, -17.89997941-6.86955109j]), array([-
18.62898328+4.30689305j, -18.62898328-4.30689305j]), array([-
1.2200838+19.00424042j, -1.2200838-19.00424042j]), array([-
6.95378503+17.68582648j, -6.95378503-17.68582648j]),
array([17.95479879+4.48567692j, 17.95479879-4.48567692j]), -17.9183383697109,
array([13.86795686+11.1270776j, 13.86795686-11.1270776j]),
array([0.42240579+17.33736326j, 0.42240579-17.33736326j]), 17.180663327975427,
array([-10.85484281+12.99746055j, -10.85484281-12.99746055j]), array([-
8.41457279+13.75855606j, -8.41457279-13.75855606j]),
array([9.19424116+11.74162278j, 9.19424116-11.74162278j]),
array([12.43212093+8.06107556j, 12.43212093-8.06107556j]), array([-
12.57586358+6.56262268j, -12.57586358-6.56262268j]),
array([4.34269269+12.55365036j, 4.34269269-12.55365036j]), 12.94990960564017,
array([12.15463776+2.88606493j, 12.15463776-2.88606493j]),
array([8.86679185+8.31166108j, 8.86679185-8.31166108j]), array([-
6.51801337+9.22297741j, -6.51801337-9.22297741j]), array([-
7.77771128+3.67937107j, -7.77771128-3.67937107j]),
array([0.95000471+8.49466019j, 0.95000471-8.49466019j]), array([-
6.11314747+2.02250404j, -6.11314747-2.02250404j]),
array([1.58857344+6.11007248j, 1.58857344-6.11007248j]), array([-
1.9606185+5.29883553j, -1.9606185-5.29883553j]), 5.532123587865671,
0.20242656693990546]
Наибольший собственные вектор:
 [-0.14385398 -0.14598671 -0.13536595 -0.14545322 -0.14624958 -0.15421271
 -0.14362198 -0.15336259 -0.13826873 -0.14005858 -0.13995884 -0.13565568
 -0.15143168 -0.14036101 -0.14401146 -0.14051339 -0.14038523 -0.13056568
 -0.13832544 -0.12938097 -0.14347639 -0.14069527 -0.14738818 -0.14240815
 -0.13694434 -0.13812655 -0.13505034 -0.14233302 -0.1416992  -0.1395524
 -0.14272178 -0.14193346 -0.14199484 -0.13854898 -0.13738362 -0.13876796
 -0.14852257 -0.14332644 -0.13897433 -0.14635337 -0.14044134 -0.13467176
 -0.1389087  -0.14809361 -0.14347247 -0.13011107 -0.14377554 -0.13712042
 -0.14515587 -0.1413318 ]
Wall time: 1min 13s
```

```
Арнольди:
A:
 [[10.72359075 11.13393155  5.45011535 ...  5.6824868  11.99036914
  14.87507716]
 [ 5.90310659 11.80042802 11.16954808 ...  7.40854133  5.68148828
  10.83780643]
 [11.39337208 12.24684578 14.25807987 ...  5.36864195 14.512801
  14.63870507]
 ...
 [ 5.22989387 10.02020103 14.24603303 ... 10.03507778  5.27834598
   8.61464961]
 [13.21367351  8.10398975 12.7823375  ...  9.29827281  6.62171877
   9.45570276]
 [10.82619811  5.80370336 10.74201628 ... 12.60582502 10.94879985
  10.22128505]]
Количество итераций:
 1513
Собственные значения:
 [498.7280282768505, array([-2.7002897+12.9878644j, -2.7002897-12.9878644j]),
array([-16.12576311+6.55239754j, -16.12576311-6.55239754j]), array([-
1.60766762+18.70783326j, -1.60766762-18.70783326j]), array([-
13.11151793+5.49223882j, -13.11151793-5.49223882j]), array([-
10.87351684+13.19716456j, -10.87351684-13.19716456j]),
array([17.95478922+4.48567637j, 17.95478922-4.48567637j]), -17.93096808189397,
array([13.86165705+11.12448465j, 13.86165705-11.12448465j]),
array([0.42240386+17.33736303j, 0.42240386-17.33736303j]), 17.18066388923096,
array([-10.85484281+12.99746055j, -10.85484281-12.99746055j]), array([-
8.41457279+13.75855606j, -8.41457279-13.75855606j]),
array([9.19456138+11.74164957j, 9.19456138-11.74164957j]),
array([12.4317678+8.06122101j, 12.4317678-8.06122101j]), array([-
12.57586358+6.56262268j, -12.57586358-6.56262268j]),
array([4.34269269+12.55365036j, 4.34269269-12.55365036j]), 12.949909605640224,
array([12.15463776+2.88606493j, 12.15463776-2.88606493j]),
array([8.86679185+8.31166108j, 8.86679185-8.31166108j]), array([-
6.51801337+9.22297741j, -6.51801337-9.22297741j]), array([-
7.77784946+3.67948634j, -7.77784946-3.67948634j]),
array([0.95000661+8.49448582j, 0.95000661-8.49448582j]), array([-
6.11314747+2.02250404j, -6.11314747-2.02250404j]),
array([1.58857344+6.11007248j, 1.58857344-6.11007248j]), array([-
1.9606185+5.29883553j, -1.9606185-5.29883553j]), 5.5321235878656445,
0.2024265669398999]
Наибольший собственные вектор:
 [-0.14385398 -0.14598671 -0.13536595 -0.14545322 -0.14624958 -0.15421271
 -0.14362198 -0.15336259 -0.13826873 -0.14005858 -0.13995884 -0.13565568
 -0.15143168 -0.14036101 -0.14401146 -0.14051339 -0.14038523 -0.13056568
 -0.13832544 -0.12938097 -0.14347639 -0.14069527 -0.14738818 -0.14240815
 -0.13694434 -0.13812655 -0.13505034 -0.14233302 -0.1416992  -0.1395524
 -0.14272178 -0.14193346 -0.14199484 -0.13854898 -0.13738362 -0.13876796
 -0.14852257 -0.14332644 -0.13897433 -0.14635337 -0.14044134 -0.13467176
 -0.1389087  -0.14809361 -0.14347247 -0.13011107 -0.14377554 -0.13712042
 -0.14515587 -0.1413318 ]
```

```
Wall time: 16.6 s
```

5. Вывод

Разница между простым QR алгоритмом и алгоритмом, который работает с итерациями Арнольди достаточно велика. В нашем примере для матриц размерностью 50x50 алгоритм с применением метода Арнольди работает ~7 раз эффективнее, а для матриц маленькой размерности существенной прибавки нет.