



HUMBOLDT UNIVERSITÄT ZU BERLIN

SEMINAR PAPER

Numerical Methods for solving Eigenvalue-Problems

Thomas Siskos (580726)

NUMERICAL INTRODUCTORY COURSE

Supervised by:

Prof. Dr. Brenda López Cabrera

June 21, 2018

Contents

1	Motivation	3
2	Similarity Transformations	4
2.1	Householder-Reflections	5
2.2	Givens-Rotations	6
3	Algorithms	7
3.1	Jacobi Method	7
3.2	QR-Method	8
3.2.1	Hessenberg Variant	9
3.2.2	Accelerated Variant	9
4	Analysis	10
4.1	Accuracy	10
4.2	Efficiency	10
5	Conclusion	10
6	Appendix	11
6.1	Eigenvalue Routines	11
6.2	Analysis: Figures	20
6.3	Analysis: Unit tests	26

List of Tables

List of Figures

1	Progress Jacobi-Method 	7
2	Progress basic QR-Method 	8
3	Progress Hessenberg-QR-Method 	9
4	Progress Accelerated QR-Method 	10
5	Unit-tests: Iterations 	11

List of Algorithms

1	jacobi	7
2	QRM1	8
3	QRM2	9
4	QRM3	9

1 Motivation

Diese Dokumentation enth"alt eine sortierte Liste der wichtigsten L^AT_EX-Befehle. Die einzelnen Listeneintr"age sind untereinander durch viele Querverweise verkettet, die ein Auffinden inhaltlich zusammengeh"origer Informationen erheblich erleichtern.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

If A is an $n \times n$ matrix, v is a non-zero vector and λ is a scalar, such that

$$Av = \lambda v \tag{1}$$

then v is called an *eigenvector* and λ is called an *eigenvalue* of the matrix A . An eigenvalue of A is a root of the characteristic equation,

$$\det(A - \lambda I) = 0 \tag{2}$$

2 Similarity Transformations

Two $n \times n$ matrices A and B are called *similar* if there exists an invertible matrix P such that

$$A = P^{-1}BP. \tag{3}$$

This transformation defined in 3 is also called a *similarity transformation*. It is obvious that the similarity relationship is commutative as well as transitive. If A and B are similar, it holds that

$$\begin{aligned} B - \lambda I &= P^{-1}BP - \lambda P^{-1}IP \\ &= A - \lambda I. \end{aligned}$$

Hence A and B have the same eigenvalues. This fact also follows immediately from the transitivity of the similarity relationship and the fact that a matrix is

similar to the diagonal matrix formed from its eigenvalues, as stated in the spectral-decomposition. Important types of similarity transformations are based around orthogonal matrices. If Q is orthogonal and

$$A = Q' B Q,$$

A and B are called *orthogonally similar*.

2.1 Householder-Reflections

Let u and v be orthonormal vectors and let x be a vector in the space spanned by u and v , such that

$$x = c_1 u + c_2 v$$

for some scalars c_1 and c_2 . The vector

$$\tilde{x} = -c_1 u + c_2 v$$

is a *reflection* of x through the line defined by the vector u . Now consider the matrix

$$Q = I - 2uu'. \tag{4}$$

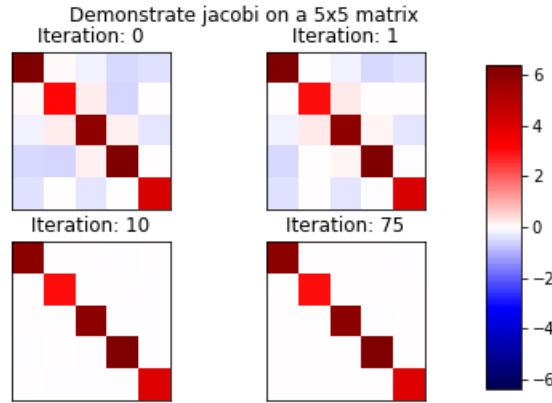
$$\begin{aligned}
Qx &= c_1u + c_2v - 2c_1uuu' - 2c_2vuu' \\
&= c_1u + c_2v - 2c_1u'uu - 2c_2u'vu \\
&= -c_1u + c_2v \\
&= \tilde{x}
\end{aligned}$$

2.2 Givens-Rotations

Using orthogonal transformations we can also rotate a vector in such a way that a specified element becomes 0 and only one other element in the vector is changed.

$$V_{pq}(\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \cos \theta & \sin \theta & & \\ & & & & \ddots & & \\ & & & -\sin \theta & \cos \theta & & \\ & & & & & 1 & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \quad (5)$$

where $\cos \theta = \frac{x_p}{||x||}$ and $\sin \theta = \frac{x_q}{||x||}$

Figure 1: Progress Jacobi-Method 

3 Algorithms

3.1 Jacobi Method

Algorithm 1 jacobi

Require: symmetric matrix A

Ensure: $0 < precision < 1$

initialize: $L \leftarrow A; U \leftarrow I; L_{max} \leftarrow 1$

while $L_{max} > precision$ **do**

 Find indices i, j of largest value in lower triangle of $abs(L)$

$L_{max} \leftarrow L_{i,j}$

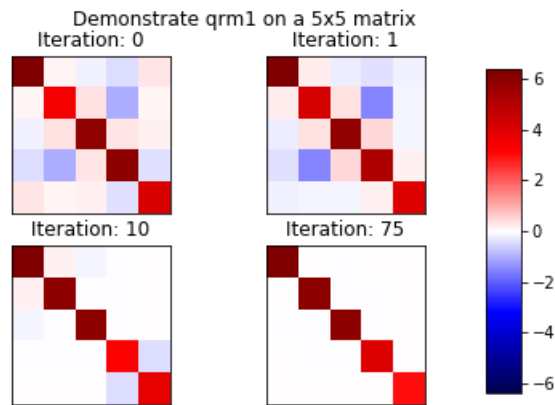
$\alpha \leftarrow \frac{1}{2} \cdot \arctan\left(\frac{2A_{i,j}}{A_{i,i} - A_{j,j}}\right)$

$V \leftarrow I$

$V_{i,i}, V_{j,j} \leftarrow \cos \alpha; V_{i,j}, V_{j,i} \leftarrow -\sin \alpha, \sin \alpha$

$A \leftarrow V'AV; U \leftarrow UV$

return $diag(A), U$

Figure 2: Progress basic QR-Method 

3.2 QR-Method

Algorithm 2 QRM1

Require: square matrix A

initialize: $conv \leftarrow False$

while not $conv$ **do**

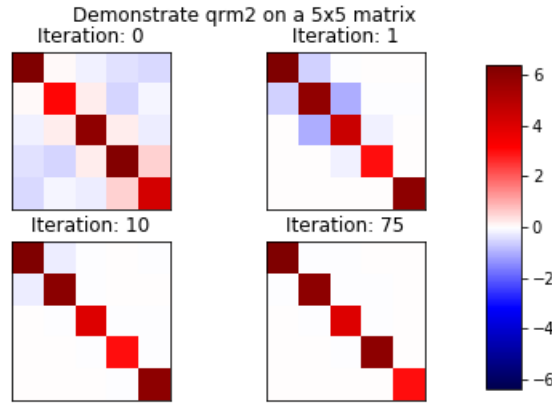
$Q, R \leftarrow$ QR-Factorization of A

$A \leftarrow RQ$

if A is diagonal **then**

$conv \leftarrow True$

return $diag(A), Q$

Figure 3: Progress Hessenberg-QR-Method 

3.2.1 Hessenberg Variant

Algorithm 3 QRM2

Require: square matrix A

$A \leftarrow \text{hessenberg}(A)$

continue with: QRM1(A)

3.2.2 Accelerated Variant

Algorithm 4 QRM3

Require: square matrix $A \in \mathbb{R}^{p \times p}$

$T \leftarrow \text{hessenberg}(A)$, $\text{conv} \leftarrow \text{False}$

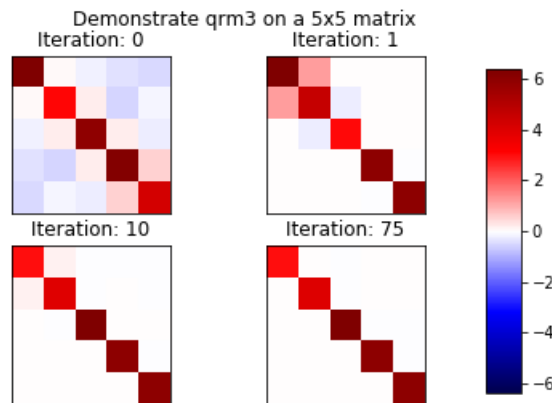
while not conv **do**

$Q, R \leftarrow \text{QR-Factorization of } T - t_{p-1,p-1}I$

$T \leftarrow RQ + t_{p-1,p-1}I$

if T is diagonal **then**

$\text{conv} \leftarrow \text{True}$
return $\text{diag}(T), Q$

Figure 4: Progress Accelerated QR-Method 

4 Analysis

4.1 Accuracy

4.2 Efficiency

5 Conclusion

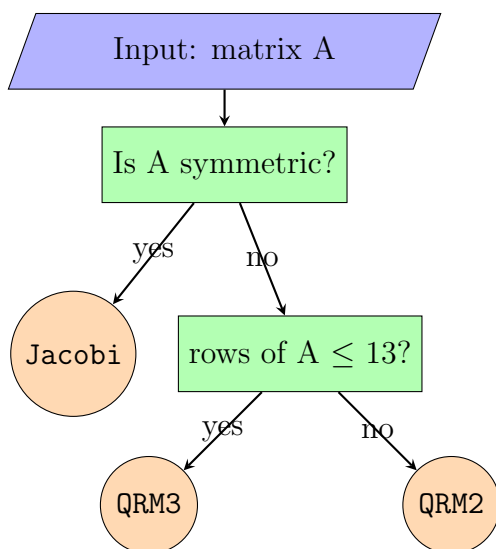
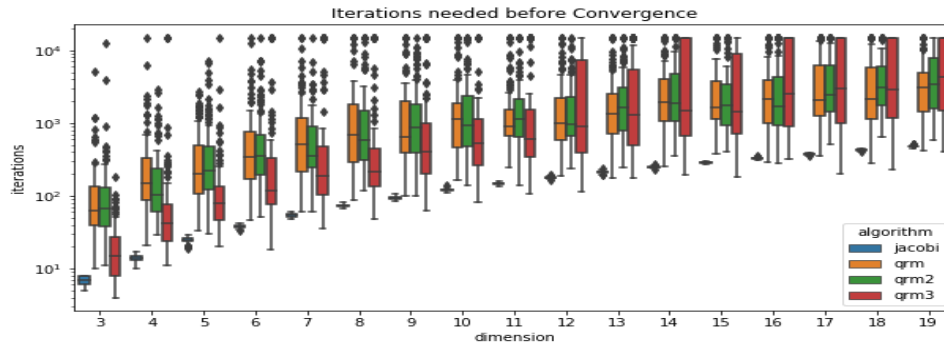


Figure 5: Unit-tests: Iterations 

6 Appendix

6.1 Eigenvalue Routines

```

1 import numpy as np
2 import copy
3
4
5 def hreflect1D(x):
6     """
7     Calculate Householder reflection:  $Q = I - 2*uu'$ .
8
9     Parameters:
10         X: numpy array.
11
12     Returns:
13         Qx: reflected vector.
14         Q: Reflector (matrix).
15     """

```

```

16     # Construct v:
17     v = copy.deepcopy(x)
18     v[0] += np.linalg.norm(x)
19
20     # Construct u: normalize v.
21     vnorm = np.linalg.norm(v)
22     if vnorm:
23         u = v / np.linalg.norm(v)
24     else:
25         u = v
26
27     # Construct Q:
28     Q = np.eye(len(x)) - 2 * np.outer(u, u)
29     Qx = np.dot(Q, x)
30
31     return Qx, Q
32
33
34 def qr_factorize(X, offset=0):
35     """
36     Compute QR factorization of X s.t. QR = X.
37
38     Parameters:
39         - X: square numpy ndarray.
40         - offset: (int) either 0 or 1. If offset is unity: compute
41             Hessenberg-
42             matrix.
43
44     Returns:
45         Q: square numpy ndarray, same shape as X. Rotation matrix.

```

```

45         R: square numpy ndarray, same shape as X. Upper triangular
         matrix if
46         offset is 0, Hessenberg-matrix if offset is 1.
47         """
48         assert offset in [0, 1]
49         assert type(X) == np.ndarray
50         assert X.shape[0] == X.shape[1]
51
52         R = copy.deepcopy(X)
53         Q = np.eye(X.shape[0])
54
55         for i in range(X.shape[0]-offset):
56             Pi = np.eye(R.shape[0])
57             _, Qi = hreflect1D(R[i+offset:, i])
58             Pi[i+offset:, i+offset:] = Qi
59
60             Q = Pi.dot(Q)
61             R = Pi.dot(R)
62
63         return Q.T, R

```

```

1     """
2     Algorithms for solving eigenvalue problems.
3
4     1. Compute diagonalization of 2x2 matrices via jacobi iteration.
5     2. Generalize Jacobi iteration for symmetric matrices.
6     """
7     import numpy as np
8     import copy
9     import warnings

```

```

10 from scipy import linalg as lin
11 from algorithms import helpers
12
13
14 def jacobi2x2(A):
15     """
16     Diagonalize a 2x2 matrix through jacobi step.
17
18     Solve:  $U^T A U = E$  s.t.  $E$  is a diagonal matrix.
19
20     Parameters:
21         A - 2x2 numpy array.
22
23     Returns:
24         A - 2x2 diagonal numpy array
25     """
26     assert type(A) == np.ndarray
27     assert A.shape == (2, 2)
28     assert A[1, 0] == A[0, 1]
29
30     alpha = 0.5 * np.arctan(2*A[0, 1]/(A[1, 1] - A[0, 0]))
31     U = np.array([[np.cos(alpha), np.sin(alpha)],
32                  [-np.sin(alpha), np.cos(alpha)]])
33     E = np.matmul(U.T, np.matmul(A, U))
34     return E
35
36 def jacobi(X, precision=1e-6, debug=False):
37     """
38     Compute Eigenvalues and Eigenvectors for symmetric matrices.
39

```

```

40  Parameters:
41      X - 2D numpy ndarray which represents a symmetric matrix
42      precision - float in (0, 1). Convergence criterion.
43
44  Returns:
45      A - 1D numpy array with eigenvalues sorted by absolute
46      value
47      U - 2D numpy array with associated eigenvectors (column).
48  """
49  assert 0 < precision < 1.
50  assert type(X) == np.ndarray
51  n, m = X.shape
52  assert n == m
53  assert all(np.isclose(X - X.T, np.zeros(n)).flatten())
54  A = copy.deepcopy(X)
55  U = np.eye(A.shape[0])
56  L = np.array([1])
57  iterations = 0
58
59  while L.max() > precision:
60      L = np.abs(np.tril(A, k=0) - np.diag(A.diagonal()))
61      i, j = np.unravel_index(L.argmax(), L.shape)
62      alpha = 0.5 * np.arctan(2*A[i, j] / (A[i, i]-A[j, j]))
63
64      V = np.eye(A.shape[0])
65      V[i, i], V[j, j] = np.cos(alpha), np.cos(alpha)
66      V[i, j], V[j, i] = -np.sin(alpha), np.sin(alpha)
67
68      A = np.dot(V.T, A.dot(V))
69      U = U.dot(V)

```



```
69         iterations += 1
70
71         # Sort by eigenvalue (descending order) and flatten A
72         A = np.diag(A)
73         order = np.abs(A).argsort()[::-1]
74         if debug:
75             return iterations
76
77         return A[order], U[:, order]
78
79
80 def qrm(X, maxiter=15000, debug=False):
81     """
82     Compute Eigenvalues and Eigenvectors using the QR-Method.
83
84     Parameters:
85         - X: square numpy ndarray.
86
87     Returns:
88         - Eigenvalues of A.
89         - Eigenvectors of A.
90     """
91     n, m = X.shape
92     assert n == m
93
94     # First stage: transform to upper Hessenberg-matrix.
95     A = copy.deepcopy(X)
96     conv = False
97     k = 0
98
99     # Second stage: perform QR-transformations.
```

```

99     while (not conv) and (k < maxiter):
100         k += 1
101         Q, R = helpers.qr_factorize(A)
102         A = R.dot(Q)
103
104         conv = np.alltrue(np.isclose(np.tril(A, k=-1), np.zeros((n
105             , n))))
106
107     if not conv:
108         warnings.warn("Convergence was not reached. Consider
109             raising maxiter.")
110
111     if debug:
112         return k
113
114     Evals = A.diagonal()
115     order = np.abs(Evals).argsort()[::-1]
116     return Evals[order], Q[order, :]
117
118 def qrm2(X, maxiter=15000, debug=False):
119     """
120     First compute similar matrix in Hessenberg form, then compute
121     the
122     Eigenvalues and Eigenvectors using the QR-Method.
123
124     Parameters:
125         - X: square numpy ndarray.
126
127     Returns:
128         - Eigenvalues of A.
129         - Eigenvectors of A.
130     """

```

```

126     n, m = X.shape
127     assert n == m
128
129     # First stage: transform to upper Hessenberg-matrix.
130     A = lin.hessenberg(X)
131     conv = False
132     k = 0
133
134     # Second stage: perform QR-transformations.
135     while (not conv) and (k < maxiter):
136         k += 1
137         Q, R = helpers.qr_factorize(A)
138         A = R.dot(Q)
139
140         conv = np.alltrue(np.isclose(np.tril(A, k=-1), np.zeros((n
141             , n))))
142
143     if not conv:
144         warnings.warn("Convergence was not reached. Consider
145             raising maxiter.")
146
147     if debug:
148         return k
149
150     Evals = A.diagonal()
151     order = np.abs(Evals).argsort()[::-1]
152     return Evals[order], Q[order, :]
153
154 def qrm3(X, maxiter=15000, debug=False):
155     """
156     First compute similar matrix in Hessenberg form, then compute

```

```

154         the
155         Eigenvalues and Eigenvectors using the QR-Method.
156
157     Parameters:
158         - X: square numpy ndarray.
159
160     Returns:
161         - Eigenvalues of A.
162         - Eigenvectors of A.
163
164     """
165     n, m = X.shape
166     assert n == m
167
168     # First stage: transform to upper Hessenberg-matrix.
169     T = lin.hessenberg(X)
170
171     conv = False
172     k = 0
173
174     # Second stage: perform QR-transformations.
175     while (not conv) and (k < maxiter):
176         k += 1
177         Q, R = helpers.qr_factorize(T - T[n-1, n-1] * np.eye(n))
178         T = R.dot(Q) + T[n-1, n-1] * np.eye(n)
179
180         conv = np.alltrue(np.isclose(np.tril(T, k=-1), np.zeros((n
181             , n))))
182
183     if not conv:
184         warnings.warn("Convergence was not reached. Consider
185             raising maxiter.")

```

```
181     if debug:
182         return k
183     Evals = T.diagonal()
184     order = np.abs(Evals).argsort()[::-1]
185     return Evals[order], Q[order, :]
```

6.2 Analysis: Figures

```
1 import os
2 import copy
3 import pandas as pd
4 import numpy as np
5 import seaborn as sns
6 from scipy import linalg as lin
7 from scipy.stats import ortho_group
8 from matplotlib import pyplot as plt
9
10 datadir = os.path.join("analysis", "benchmarks.csv")
11 outpath = os.path.join("media", "plots")
12 trials = pd.read_csv(datadir, index_col=0)
13
14 trials.groupby(["algorithm", "dimension"]).iterations.describe()
15
16 # Boxplot iteration:
17 fig = plt.figure(figsize=(10, 5))
18 sns.boxplot(x="dimension", y="iterations", hue="algorithm", data=
19     trials)
20 plt.yscale("log")
21 plt.title("Iterations needed before Convergence")
```

```

21 plt.savefig(os.path.join(outpath, "iterations_boxplot.png"))
22 plt.show()
23 plt.close()
24
25 # Boxplot elapsed time:
26 fig = plt.figure(figsize=(10, 5))
27 sns.boxplot(x="dimension", y="time", hue="algorithm", data=trials)
28 plt.title("Time needed before Convergence")
29 plt.ylabel("time (sec)")
30 plt.yscale('log')
31 plt.savefig(os.path.join(outpath, "time_boxplot.png"))
32 plt.show()
33 plt.close()
34
35 # Visualize Algorithm-Progress:
36 np.random.seed(42)
37 size = 5
38 Lambda = np.diag(np.random.randint(low=0, high=10, size=size))
39 G = ortho_group.rvs(dim=size)
40 X = np.dot(G, Lambda.dot(G.T))
41
42
43 def plot_factory(func):
44     def plotter(savepath, **fig_kw):
45         def algorithm_generator(*args, **kwargs):
46             return func(*args, **kwargs)
47
48         fig, ax = plt.subplots(nrows=2, ncols=2, **fig_kw)
49         algorithm_iterator = algorithm_generator()
50         j = -1

```

```
51
52     for i, A in enumerate(algorithm_iterator):
53         if i in (0, 1, 10, 75):
54             j += 1
55
56             hm = ax[j // 2, j % 2].imshow(A,
57                                           cmap=plt.get_cmap('
58                                           seismic'),
59                                           vmin=-X.max(),
60                                           vmax=X.max())
61
62             ax[j // 2, j % 2].set_yticks([])
63             ax[j // 2, j % 2].set_xticks([])
64             ax[j // 2, j % 2].set_title("Iteration: " + str(i)
65                                         )
66
67             if i > 75:
68                 break
69
70
71     fig.subplots_adjust(right=0.8)
72     cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
73     fig.colorbar(hm, cax=cbar_ax)
74
75     sup_title = "Demonstrate {} on a {}x{} matrix".format(
76         func.__name__,
77         *X.shape)
78
79     fig.suptitle(sup_title)
80     fig.savefig(savepath)
81
82     return fig, ax
```

```

79
80     return plotter
81
82
83 @plot_factory
84 def jacobi():
85     """
86     Compute Eigenvalues and Eigenvectors for symmetric matrices
87     using the
88     jacobi method.
89
90     Yields:
91         * A - 2D numpy array of current iteration step.
92     """
93     A = copy.deepcopy(X)
94     U = np.eye(A.shape[0])
95     L = np.array([1])
96     iterations = 0
97
98     while iterations < 5000:
99         L = np.abs(np.tril(A, k=0) - np.diag(A.diagonal()))
100         i, j = np.unravel_index(L.argmax(), L.shape)
101         alpha = 0.5 * np.arctan(2*A[i, j] / (A[i, i]-A[j, j]))
102
103         V = np.eye(A.shape[0])
104         V[i, i], V[j, j] = np.cos(alpha), np.cos(alpha)
105         V[i, j], V[j, i] = -np.sin(alpha), np.sin(alpha)
106
107         A = np.dot(V.T, A.dot(V))
108         U = U.dot(V)

```



```

108         iterations += 1
109         yield A
110
111
112 @plot_factory
113 def qrm1():
114     """
115     Create generator for transformed matrices after applying the
116     QR-Method.
117
118     Yields:
119         - T: 2D-numpy array. Similar matrix to X.
120     """
121     # First stage: transform to upper Hessenberg-matrix.
122     T = copy.deepcopy(X)
123
124     k = 0
125     # Second stage: perform QR-transformations.
126     while k < 5000:
127         k += 1
128         Q, R = np.linalg.qr(T)
129         T = R.dot(Q)
130         yield T
131
132 @plot_factory
133 def qrm2():
134     """
135     Create generator for transformed matrices after applying the
136     QR-Method.

```

```

136
137     Yields:
138         - T: 2D-numpy array. Similar matrix to X.
139     """
140     # First stage: transform to upper Hessenberg-matrix.
141     T = lin.hessenberg(X)
142
143     k = 0
144     # Second stage: perform QR-transformations.
145     while k < 5000:
146         if k == 0:
147             yield X
148             k += 1
149             Q, R = np.linalg.qr(T)
150             T = R.dot(Q)
151             yield T
152
153
154 @plot_factory
155 def qrm3():
156     """
157     First compute similar matrix in Hessenberg form, then compute
158     the
159     Eigenvalues and Eigenvectors using the accelerated QR-Method.
160
161     Yields:
162         * T - 2D numpy array of current iteration step.
163     """
164     # First stage: transform to upper Hessenberg-matrix.
165     T = lin.hessenberg(X)

```

```
165     k = 0
166     n, _ = X.shape
167
168     # Second stage: perform QR-transformations.
169     while k < 5000:
170         if k == 0:
171             yield X
172             k += 1
173             Q, R = np.linalg.qr(T - T[n-1, n-1] * np.eye(n))
174             T = R.dot(Q) + T[n-1, n-1] * np.eye(n)
175
176             yield T
177
178
179     jacobi(os.path.join(outpath, "jacobi.png"))
180     qrm1(os.path.join(outpath, "qrm1.png"))
181     qrm2(os.path.join(outpath, "qrm2.png"))
182     qrm3(os.path.join(outpath, "qrm3.png"))
183
184     plt.show()
185     plt.close()
```

6.3 Analysis: Unit tests

```
1  """
2  Automated tests for different algorithms.
3  """
4  import os
5  import numpy as np
```

```
6 import threading
7 import pandas as pd
8 from algorithms import eigen
9 from scipy.stats import ortho_group
10 from tqdm import trange, tqdm
11
12 data_out = os.path.join("data", "accuracy_tests.csv")
13
14
15 def get_test_matrix(dim):
16     """Return matrix with associated Eigenvalues."""
17     eigenvalues = np.random.uniform(size=dim)
18     eigenvectors = ortho_group.rvs(dim=dim)
19     Lambda = np.diag(eigenvalues)
20
21     matrix = np.dot(eigenvectors, Lambda).dot(eigenvectors.T)
22
23     order = np.abs(eigenvalues).argsort()[::-1]
24     return matrix, eigenvalues[order]
25
26
27 def test_algo(algo, Ntests=1000, dim=3, *args, **kwargs):
28     """
29     Test routine that allows for threading. Note that the
30     variables:
31     failed, critical and problematic need to be defined in the
32     enveloping or
33     global scope beforehand.
34
35     Parameters:
```

```
34         - algo: algorithm to be tested
35         - Ntests: number of tests to compute
36         - dim: dimensions of matrix
37         - *args, **kwargs: additional arguments to be passed to
           algo.
38
39     Returns:
40         - None, but will update the variables failed, critical and
           problematic.
41         + failed: number of failed tests
42         + critical: number of ZeroDivisionErrors
43         + problematic: list of numpy arrays which led to wrong
           eigenvalues.
44
45     """
46     global failed
47     global critical
48     global problematic
49
50     for _ in range(Ntests):
51         try:
52             A, true_eig = get_test_matrix(dim=dim)
53             my_eig, _ = algo(A, *args, **kwargs)
54             assert np.alltrue(np.isclose(my_eig, true_eig))
55
56         except AssertionError:
57             failed += 1
58             problematic.append(A)
59
60         except ZeroDivisionError:
61             critical += 1
```

```
61
62
63 def threaded_tests(algo, N, nWorkers=10, verbose=True, *args, **
    kwargs):
64     global failed
65     global critical
66     global problematic
67
68     assert N % nWorkers == 0
69
70     n = N // nWorkers
71     threadlist = [None] * nWorkers
72
73     for i in range(nWorkers):
74         threadlist[i] = threading.Thread(target=test_algo,
75                                           args=(algo, n, *args))
76         threadlist[i].start()
77
78     for i in range(nWorkers):
79         threadlist[i].join()
80
81     logstr = """
82     {} out of {} tests failed.
83     {} tests failed critically.
84     """.format(failed, N, critical)
85
86     if verbose:
87         print(logstr)
88
89
```

```
90 # Tests
91 results = {
92     "algorithm": [],
93     "dimension": [],
94     "maxiter": [],
95     "failed": []}
96
97 for algo in tqdm([eigen.jacobi, eigen.qrm, eigen.qrm2, eigen.qrm3
98 ]):
99     for dim in trange(3, 15):
100         for maxiter in 1000, 10000, 100000:
101             if algo.__name__ == eigen.jacobi:
102                 maxiter = 1e-6
103
104                 failed = 0
105                 critical = 0
106                 problematic = []
107                 threaded_tests(algo, 1000, 20, False, dim, maxiter)
108                 results["algorithm"].append(algo.__name__)
109                 results["dimension"].append(dim)
110                 results["maxiter"].append(maxiter)
111                 results["failed"].append(failed)
112
113 test_data = pd.DataFrame(results)
114 test_data.to_csv(data_out)
```

References

- [1] Seffen Börm and Christian Mehl. *Numerical Methods for Eigenvalue Problems*. Walter de Gruyter GmbH & Co.KG, Berlin/Boston, 2012.
- [2] James E. Gentle. *Numerical Linear Algebra for Applications in Statistics*. Springer Science and Business Media, New York, 1998.
- [3] Wolfgang K. Härdle and Léopold Simar. *Applied Multivariate Statistical Analysis*. Springer-Verlag GmbH, Berlin, Heidelberg, 2015.
- [4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.
- [5] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.
- [6] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.