```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

1 Klassisches und modifiziertes Gram-Schmidt-Verfahren

Hier werden die Plots aus den Folien zur Wiederholung der Linearen Algebra (LinAWdh.pdf) erstellt. Gleichzeitig sind hier die Musterlösungen zur Aufgabe 41. Diese Lektion wurde nicht in der Vorlesung gezeigt.

```
import numpy as np;
from scipy.linalg import norm;
from matplotlib import pyplot as plt;

floatFormatter = lambda x: "{0: .2f}".format(x);
np.set_printoptions(formatter = { 'float_kind': floatFormatter });

M = 5;
N = M - 1;
A = np.random.rand(M, N);
```

1.1 Klassisches Gram-Schmidt-Verfahren

1.1.1 Version 1

```
def GramSchmidt_V1(A):
    Q = np.zeros_like(A) # Float-Nullen der gleichen Größe, wie A
    R = np.zeros((A.shape[1], A.shape[1])) # Platz für die obere Dreiecksmatrix
    for k in range(A.shape[1]):
        Q[:, k] = A[:, k]
        if k != 0: # Verfahren aus der Vorlesung
            for j in range(k):
                R[j, k] = Q[:, j].T@Q[:, k]
            Q[:, k] = Q[:, k] - Q[:, :k]@R[:k, k]
        R[k, k] = np.sqrt(Q[:, k]@(Q[:, k].T))
        Q[:, k] = Q[:, k]/R[k, k]
    return Q, R
```

1.1.2 Version 2 (vektorisiert)

```
def GramSchmidt_V2(A):
    m, n = A.shape;
    Q = 0. * A;         # Float-Nullen der gleichen Größe, wie A
    R = 0. * A[:n, :]; # Platz für die obere Dreiecksmatrix
    for kk in range(n):
        a_k = A[:, kk];
        # Orthogonalisierungsfaktoren
        # der letzte ist Null, da Q in Spalte j noch Nullen enthält
        r_k = a_k @ Q[:, :kk+1];
        # Orthogonalisierung durchführen (alles in einem Schritt)
        q_k = a_k - Q[:, :kk] @ r_k[:-1];
        # Normieren
```

```
        r_k[-1] = norm(q_k);
        # Eintragen
        Q[:, kk] = q_k / r_k[-1];
        R[:kk+1, kk] = r_k;
    return Q, R;
```

1.1.3 Vergleich

```
Q_, R_ = GramSchmidt_V1(A);
Q, R = GramSchmidt_V2(A);
print("Q =\n{0}\n\nR =\n{1}\n".format(Q, R));
print("Version 1 und Version 2 stimmen ueberein",np.allclose(Q_,Q) and np.allclose(R_,R))
print("max(| A - Q*R |) = {0:g}, max(| Q^T*Q - Id |) = {1:g}".format(
        abs(A - Q@R).max(), abs(Q.T @ Q - np.eye(N)).max()));
```

```
Q =
[[ 0.31 -0.31 -0.26 -0.28]
 [ 0.46 -0.71  0.45  0.12]
 [ 0.42  0.15 -0.39  0.80]
 [ 0.64  0.29 -0.30 -0.51]
 [ 0.32  0.55  0.70  0.04]]

R =
[[ 1.55  0.79  0.77  1.07]
 [ 0.00  0.47  0.07 -0.20]
 [ 0.00  0.00  0.36 -0.13]
 [ 0.00  0.00  0.00  0.61]]

Version 1 und Version 2 stimmen ueberein True
max(| A - Q*R |) = 1.11022e-16, max(| Q^T*Q - Id |) = 4.93617e-16
```

1.2 Modifiziertes Gram-Schmidt-Verfahren

```
M = 5;
N = M - 1;
A = np.random.rand(M, N);
```

1.2.1 Version 1

```
def GramSchmidtMod_V1(A):
    Q = np.zeros_like(A)
    R = np.zeros((A.shape[1],A.shape[1]))
    for k in range(A.shape[1]):
        Q[:, k] = A[:, k]
        for j in range(k):
            R[j, k] = Q[:, j].T@Q[:, k]
            Q[:, k] -= Q[:,j]*R[j, k]
        R[k, k] = np.sqrt(Q[:, k]@(Q[:, k].T))
        Q[:, k] = Q[:,k]/R[k, k]
    return Q,R
```

1.2.2 Version 2

```
def GramSchmidtMod_V2(A):
    m, n = A.shape;
    Q = 0 * A;          # Nullen des gleichen Datentyps, wie A
    R = 0 * A[:n, :];
    for kk in range(n):
        q_k = A[:, kk].copy();
        for jj in range(kk):
            R[jj, kk] = q_k @ Q[:, jj];
            q_k -= R[jj, kk] * Q[:, jj];
        R[kk, kk] = norm(q_k);
        # Eintragen
        Q[:, kk] = q_k / R[kk, kk];
    return Q, R
```

1.2.3 Vergleich

```
Q, R = GramSchmidtMod_V1(A);
Q_, R_ = GramSchmidtMod_V2(A);
print("Q =\n{0}\n\nR =\n{1}\n".format(Q, R));
print("Version 1 und Version 2 stimmen ueberein",np.allclose(Q_,Q) and np.allclose(R_,R)
print("max(| A - Q*R |) = {0:g}, max(| Q^T*Q - Id |) = {1:g}".format(
        abs(A - Q@R).max(), abs(Q.T @ Q - np.eye(N)).max()));
```

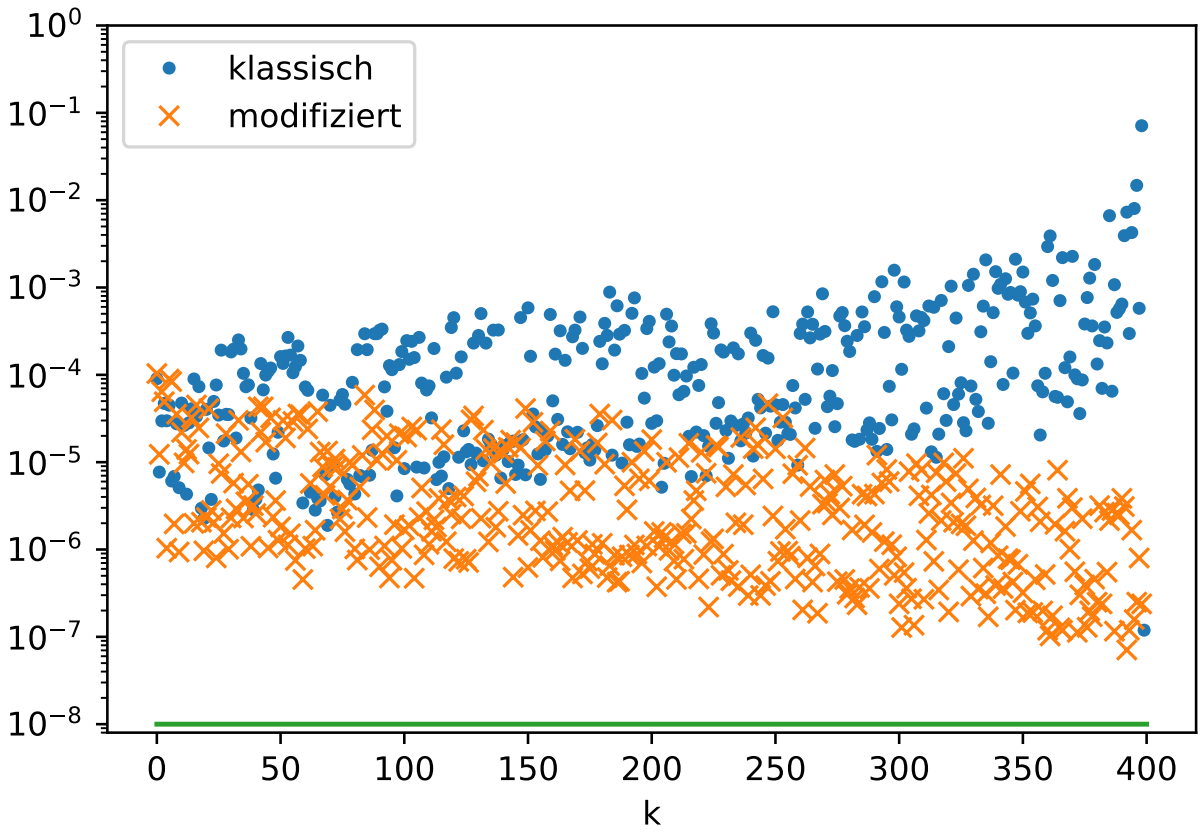
```
Q =
[[ 0.68  0.08 -0.54  0.48]
 [ 0.02  0.35  0.47  0.31]
 [ 0.32  0.79  0.06 -0.48]
 [ 0.62 -0.50  0.33 -0.46]
 [ 0.22 -0.01  0.61  0.49]]

R =
[[ 1.35  1.16  0.81  1.04]
 [ 0.00  0.78 -0.29  0.84]
 [ 0.00  0.00  0.89  0.65]
 [ 0.00  0.00  0.00  0.73]]

Version 1 und Version 2 stimmen ueberein True
max(| A - Q*R |) = 1.11022e-16, max(| Q^T*Q - Id |) = 2.09243e-16
```

1.3 Vergleich von klassischem und modifiziertem QR

```
M = 400;
N = M;
A = np.random.rand(M, N);  # Zufallsmatrix
A1 = A.astype(np.float32); # Einfache Genauigkeit
abs(A1 - A).max()

Q1, R1 = GramSchmidt_V1(A1);
print("max(| A - Q1*R1 |) = {0:g}, max(| Q1^T * Q1 - Id |) = {1:g}".format(
        abs(A1 - Q1@R1).max(), (Q1.T @ Q1 - np.eye(N)).max()));

Q2, R2 = GramSchmidtMod_V1(A1);
print("max(| A - Q2*R2 |) = {0:g}, max(| Q2^T * Q2 - Id |) = {1:g}".format(
```

```
        abs(A1 - Q2@R2).max(), (Q2.T @ Q2 - np.eye(N)).max()));


res1 = np.triu(Q1.transpose() @ Q1 - np.eye(N)).max(1);
res2 = np.triu(Q2.transpose() @ Q2 - np.eye(N)).max(1);


plt.figure(1)
plt.semilogy(res1, ".", label = "klassisch");
plt.semilogy(res2, "x", label = "modifiziert")
plt.plot([ 0, N ], [ 1e-8, 1e-8 ]); # Maschinengenauigkeit (zur Referenz)
plt.ylim([ 8e-9, 1 ]);
plt.legend(numpoints = 1, fancybox = True);
plt.xlabel("k");
plt.savefig("gramschmidt.pdf");
```

```
max(| A - Q1*R1 |) = 6.17066e-08, max(| Q1^T * Q1 - Id |) = 0.0712262
max(| A - Q2*R2 |) = 7.82871e-07, max(| Q2^T * Q2 - Id |) =
0.000103524
```



M = 40;

```
N = M;
A = np.random.rand(M, N);
A1 = A.astype(np.float32);
abs(A1 - A).max()

Q1, R1 = GramSchmidt_V1(A1);
Q2, R2 = GramSchmidtMod_V2(A1);
```

```
orth1 = abs(Q1.T @ Q1);
orth1[orth1 == 0] = 1e-8;
orthLog1 = np.log10(orth1);
orth2 = abs(Q2.T @ Q2);
orth2[orth2 == 0] = 1e-8;
orthLog2 = np.log10(orth2);

plt.figure(2)
plt.subplot(1, 2, 1);
h = plt.imshow(orthLog1, cmap = "gray", interpolation = "nearest");
h.set_clim(vmin = -8, vmax = 0);
plt.title("Klassisch")
plt.subplot(1, 2, 2);
h = plt.imshow(orthLog2, cmap = "gray", interpolation = "nearest");
h.set_clim(vmin = -8, vmax = 0);
plt.title("Modifiziert");
plt.savefig("gramschmidt-orth.pdf");
```