Modellierung WS 22-23 Blatt 2

Abgabe: 14.11.22 23:59 auf Mattermost oder per Mail an jadissel

Aufgabe 1: Orthogonalität und Skalarprodukt

In der Vorlesung wurde das **Orthogonale Komplement** eines Untervektorraums definiert. Hierzu wird jedoch ein Skalarprodukt benötigt. Wir wollen uns hier näher mit orthogonalität beschäftigen. Seien $u, v, w \in V$ linear unabhängige Vektoren.

1. Zeige: u steht orthogonal auf $v' = v - \frac{<u,v>}{<u,u>}u$
2. Zeige: u und v' stehen orthogonal auf $w' = w - \frac{<w,v'>}{<v',v'>}v' - \frac{<w,u>}{<u,u>}u$

Sei nun V der Vektorraum aller Polynome bis Grad 2. (Also 3, x, $x^2 + 5x + 7$,...)
Ein häufig verwendetes Skalarprodukt in diesem Raum ist folgendermaßen definiert:

$$\langle f, g \rangle = \int_0^1 f(x)g(x)dx$$

3. Zeige, dass diese Vorschrift tatsächlich ein Skalarprodukt definiert. Hierfür genügt zu zeigen, dass:

- $\langle f, g \rangle = \langle g, f \rangle$
- $\langle f, g \rangle$ ist linear im ersten Argument
- Für $f \neq 0$ ist $\langle f, f \rangle > 0$

4. Berechne eine Orthogonalbasis für V (eine Basis aus drei Vektoren die orthogonal aufeinander stehen)

Aufgabe 2: Algorithmen zum Lösen Linearer Gleichungssysteme

In dieser Aufgabe wollen wir zwei einfach zu implementierende verfahren zum lösen Linearer Gleichungssysteme vergleichen. Wir werden hierbei die **Cramersche Regel** und das **Jacobi Verfahren** betrachten.

(Das **Jacobi Verfahren** ist übrigens ein Spezialfall des Gauss-Seidel Lösers der in der Vorlesung angesprochen wurde)

1. Schaut euch die Verfahren auf Wikipedia an. Für die Implementierung reicht in beiden Fällen der erste Abschnitt nach dem Inhaltsverzeichnis aus.
2. Implementiert die Verfahren.
    - Als Hilfestellung gibt es weiter unten ein Python script template welches unter anderem Methoden zum Testen der Implementierung und zum Messen der Zeit enthält
    - Praktische funktionen von numpy:
        - *numpy.linalg.det* - berechnet eine Determinante
        - *numpy.diag* - wandelt Diagonale einer Matrix in ein Array (und andersrum) um
        - @ - Numpy operator für Matrix-multiplikation (A@b multipliziert Matrix A mit Vektor b)

```python
import numpy as np

#berechnet die determinante einer matrix
from numpy.linalg import det
from numpy import diag

##########   Cramer Regel ######################
def subdeterminant(A, b, i):
    """Berechne die Subdeterminante von A, wenn die i-te Spalte durch b ersetzt
wird"""
    pass

def cramers_rule(A,b):
    """Löse das lineare Gleichungssystem Ax=b mit der Cramerschen Regel"""
    pass

##########   Jacobi Verfahren ######################
def split_matrix(A):
    """Teile eine Matrix in eine Diagonalmatrix und den Rest auf"""
    pass

def single_iteration(D_inv,LU,x,b):
    """Führe eine Iteration des Jacobi-Verfahrens durch"""
    pass

def jacobi_method(A,b):
    """Führe das Jacobi-Verfahren für Ax=b bis zur kovergenz durch"""
    pass



########## Funktionen zum testen ############################
def create_linear_problem(n):
    """Erstelle ein lineares Gleichungssystem der Form Ax=b,
    wobei A diagonal dominant ist"""
    A = np.random.rand(n,n)
    A = A + np.diag(np.sum(A, axis=0)) #make diagonally dominant
    b = np.random.rand(n)
    return A,b

def time_method(method, A, b):
    """Misst Zeit die die Ausführung einer Methode benötigt"""
    start = time.time()
    method(A,b)
    end = time.time()
    return end-start

def test_method(method):
```

```
    """Testet eine Methode, indem die Lösung mit der Lösung von numpy verglichen
wird"""
    correct = 0
    for i in range(10):
        A,b = create_linear_problem(100)
        x = method(A,b)
        x_np = np.linalg.solve(A,b)
        if x is not None:
            correct += np.allclose(x,x_np)
    print("%s: Correct: %d/10" % (method.__name__,correct))

def plot_time(n):
    import matplotlib.pyplot as plt
    """Plottet die benötigte Zeit für die Jacobi- und Cramersche Regel"""
    timesJacobi = []
    timesCramer = []
    for i in range(1,n):
        A,b = create_linear_problem(i)
        timesJacobi.append(time_method(jacobi_method, A, b))
        timesCramer.append(time_method(cramers_rule, A, b))
    plt.plot(range(1,n), timesJacobi, label='Jacobi')
    plt.plot(range(1,n), timesCramer, label='Cramer')
    plt.legend()
    plt.show()

test_method(jacobi_method)
test_method(cramers_rule)

#plot_time(150)
```