Homework 1

Rebekah Grant A02297991

Jan 22, 2024

Question 1: Write out the details for the accuracy analysis for, $D\_f(x) = \frac{f(x)-f(x-h)}{h}$

we know that the taylor series expansion of $u(x)$ is as follows,

$$u(x+h) = u(x) + hu'(x) + \frac{1}{2}h^2u"(x) + \frac{1}{6}u'''(x) + \frac{1}{24}h^4(\xi)$$

$$D\_f(\bar{x}) = \frac{f(x) - f(\bar{x}-h)}{h}$$

$$D\_f(x) = u'(x) + \frac{1}{2}hu"(x) + \frac{1}{6}h^2u'''(x) + \frac{1}{24}h^4(\xi)$$

$\frac{1}{24}h^4(\xi) could be subituted for O(h^4)$, where $O(h^k)$ represents the remaining terms in the taylor series expansion. The error is represented by the function below,

$$|u(x+h) - D\_u(x)| \leq Ch$$
$$|u(x+h) - D\_u(x)| = \frac{1}{2}hf''(\xi)$$

compute in terms of h and a constant. What conditions must be satisfied in orfer to use the difference quotient?

To use the difference quotient and have $c < error < ch$ must be twice differentiable on some interval containing $\bar{x}$

Quetion 2: Write a code that returns the coefficients for a difference quotient approximating the first derivative of a function at a specified point given an input array of points.

Python Code

```
import numpy as np
def difQuotientApprox(f, x0, xList: np.array, h) -> np.array:
    coefficients = np.array.empty
    for i in xList:
        c= f(i)+ centralDiff(f, i, h) * (x0 - i)
        np.append(coefficients,c)

    return np.array

def centralDiff(f, x, h):
    return (f(x+h) + f(x-h))/ (2*h)
```

Question 3: Write a code that will return the coefficients of a derivative of a given order specified at a minimal number of points specified by the user.

```
import numpy as np
def difQuotient_Coeff(xbar: int, xList: np.array, order: int) -> np.array:
    #generate array of size n x n
    n = len(xList)
    A = np.ones((n,n))
    xrow = np.vander((xList)-xbar, increasing=True)
    b= np.zeros((n))
    b[order] = 1

    #solve system of eq
    c = np.linalg.solve(A,b)
    return c.flatten()
```

Question 4: Write a code that will determine the accuracy of a specified difference quotient. That is, instead of computing the coefficients, input the coefficients and determine the number of equations that should be satisfied.

Python Code:

```
def check(c, order, x)-> np.array:
    n = len(c)
    x_val = np.aray([x+i * order for i in range(n)])
    A = np.vander(x_val, increasing=true)
    result = np.linalg.solve(A,c)
    return result
```

Implementation:

```
def main():
    coefficients = np.array([1,2,3,4])
    order = 2
    x_value = 0

    result = check(coefficients, order, x_value)
    print(result)
```