APPM 4600 Lab 4

Ghulam Yahya Rajabi and Gavin Shields

Due Feb 13 2024

1 results

Aitken's Method Approximation: 1.365230013414136
Iterations needed: 5

didn't get far in this lab because the majority of the time was spent on modifying the fixedpt function with the new subroutine and making the function for aitkens acceleration. the debugging of the aitkens took the whole lab.

2 modified fixedpt

```
import numpy as np

def main():
    # Define test functions
    # The true fixed point for f1 is approximately 1.4987
    function1 = lambda x: 1 + 0.5 * np.sin(x)
    # The true fixed point for f2 is approximately 3.09
    function2 = lambda x: 3 + 2 * np.sin(x)

    # Set maximum iterations and tolerance
    max_iterations = 100
    tolerance = 1e-6

    # Test function 1
    initial_guess = 0.0
    approximate_fixed_point, error_code, iterations = find_fixed_point(function1
    print('Approximate fixed point:', approximate_fixed_point[iterations])
    print('Value of f1(approximate fixed point):', function1(approximate_fixed_p
    print('Error code:', error_code)
    # Test function 2
    initial_guess = 0.0
    approximate_fixed_point, error_code, iterations = find_fixed_point(function2
    print('Approximate fixed point:', approximate_fixed_point[iterations])
    print('Value of f2(approximate fixed point):', function2(approximate_fixed_p
    print('Error code:', error_code)

def find_fixed_point(f, initial_guess, tolerance, max_iterations):
    '''
    Finds the fixed point of a function using the fixed-point iteration method.

    Parameters:
    f (function): The function for which the fixed point is to be found.
    initial_guess (float): The initial guess for the fixed point.
    tolerance (float): The tolerance for stopping iterations.
    max_iterations (int): The maximum number of iterations allowed.

    Returns:
    approximate_fixed_point (float): The approximate fixed point.
    error_code (int): 0 for success, 1 for error.
    iterations (int): The number of iterations performed.
    '''

    # Initialize iteration counter
    iteration_count = 0

    # Initialize array to store all approximations
    approximation_history = np.zeros(max_iterations + 1)
    # Set initial value
    approximation_history[iteration_count] = initial_guess

    # Iterate until maximum iterations reached
    while iteration_count < max_iterations:
        # Increment iteration counter
        iteration_count += 1

        # Calculate next approximation using fixed-point iteration formula: x_(n
        next_approximation = f(initial_guess)
        # Store the new approximation
        approximation_history[iteration_count] = next_approximation

        # Check if tolerance criteria met
        if abs(next_approximation - initial_guess) / abs(initial_guess) < tolera
            approximate_fixed_point = next_approximation
            # Success
            error_code = 0
            return approximation_history, error_code, iteration_count

        # Update initial guess for the next iteration
        initial_guess = next_approximation

    # If maximum iterations reached without meeting tolerance criteria
    approximate_fixed_point = next_approximation
    # Failure
    error_code = 1
    return approximation_history, error_code, iteration_count

#main()
```

3 aitkens acceleration method

```
import numpy as np
from mod_fixedpt import find_fixed_point

def aitkens_acceleration(sequence, sequence_length, tolerance):
    accelerated_sequence = []
    for (value, i) in list(zip(sequence, range(sequence_length))):
        new_value = sequence[i] - (((sequence[i + 1] - sequence[i]) ** 2) / (seq
        accelerated_sequence.append(new_value)
        if (abs(accelerated_sequence[i] - accelerated_sequence[i - 1]) / abs(acc
            return (accelerated_sequence, i)

    return accelerated_sequence

# Define the function for which fixed point is to be found
function = lambda x: ((10) / (x + 4)) ** (1 / 2)

# Apply fixed point iteration method
approximate_fixed_point, error_code, iterations = find_fixed_point(function, ini
print("Fixed point approximation: " + str(approximate_fixed_point[-1]))
# Use -1 to get the last element
print("Iterations needed: " + str(iterations))
print("Error code: " + str(error_code))

# Apply Aitken's method for acceleration
accelerated_fixed_point, count = aitkens_acceleration(approximate_fixed_point, i
print("")
print("Aitken's Method Approximation: " + str(accelerated_fixed_point[count]))
print("Iterations needed: " + str(count))
```