MATH 376: Numerical Analysis

Project 4: Thermocline at Platte Lake

Quan Vu

November 12, 2017

1 Abstract

This project investigates the thermocline, which is the horizontal plane separating regions of different temperatures in lakes. By using data, which give the depth of water and the corresponding temperature at that depth, collected at Platte Lake, we use the method of cubic spline interpolation to find a function that gives the water temperature at any given depth. Using this function and its first and second derivatives, we are able to find the position at which the thermocline occurs, and from there we can also calculate the heat flux across this thermocline.

2 Introduction

2.1 Background information

In this section, we explore the method of cubic spline interpolation, which we will use to interpolate data points collected by experimentation. The data given, reflecting the various water depths and the corresponding water temperature, are collected from Platte Lake.

2.2 Problem description

The first task in this problem is to find a function that models water temperature with respect to changing water depths. After that, we need to find the position of the thermocline. The thermocline is defined to be the point of inflection for the temperature depth graph. In other words, this is the point where $\frac{d^2T}{dx^2} = 0$, where T is the temperature, measured in Celcius degrees, and x is the depth, measured in meters. Also, the thermocline is the position at which the absolute value of the first derivative of the graph is at maximum.
After finding the thermocline, we determine the heat flux across the thermocline by using this equation:

$$J = -\alpha \times \rho \times C \times \frac{dT}{dx}$$

where J is the heat flux in $cal/(cm^2s$, α is the eddy diffusion coefficient cm^2/s, ρ is the density of water($\approx 1g/cm^3$), and C is the specific heat of water $\approx 1cal/gC$.

2.3 Outline

Here the method chosen to find such a function is the method of cubic spline interpolation. Since this method introduces $3N - 3$ unknowns, but we can only construct $3N - 5$ linear equations using the properties discussed in class, we have to supply 2 more equations in order to find all the unknowns. The problem wants to explore natural cubic spline interpolation, meaning that if $f(x)$ is our function, then $f''(x) = 0$ at both of the outermost data points.

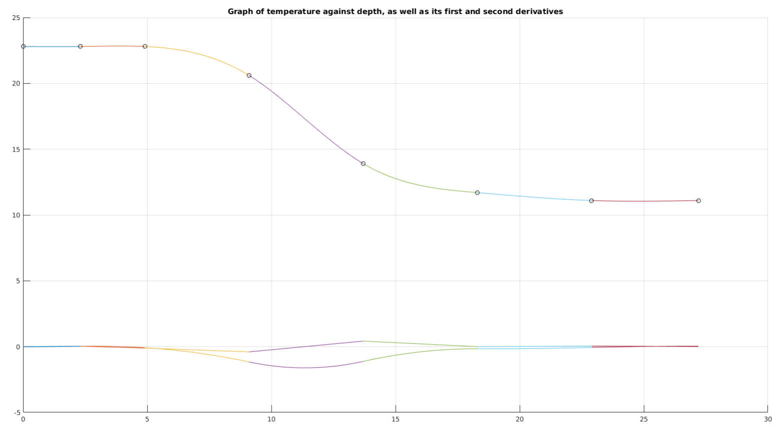
3 Numerical methods

In this project, we use the method of cubic spline interpolation, with natural endpoint conditions. Given the data points $(x1, y1), (x2, y2), ...(xN, yN)$, the method constructs a function that interpolates all the data points. The code to implement this is attached in the appendix.

4 Results

4.1 Graph of temperature against depth

This is the graph produced by using Matlab to construct the cubic spline that interpolates all the data points given:



4.2 Discussion

From the graph given, we see that the thermocline occurs in the depth between the 4th and the 5th data points, or in the purple cubic function. We proceed to find the function modeling the temperature against depth between those 2 points using the coefficients calculated from before. The, we take the second derivative of that function with respect to the depth. After that, we use the bisection method to find the root of the second derivative in this interval, to give that the thermocline is ≈ 11.3463673830. Plugging this into the equation for the heat flux, we find that the heat flux around the thermocline is $J \approx 1.614055517555187e-04$

4.3 Appendix

4.3.1 Natural Cubic Spline calculation in Matlab

```matlab
function [result] = spline_natural(x, y)
    [k, n] = size(x);
    [l, m] = size(y);
    if n ~= m || k ~= l
        error('Wrong arguments given. Exiting');
    end
    % Matrix to hold results. bi is at entry 3 * (i-1) + 1, ci is at 3 *
    % (i-1) + 2, di is at 3 * (i-1) + 3
    result = zeros(3 * (n - 1), 1);

    % Matrix to hold all equations
    eq = zeros(3 * (n-1), 3 * (n-1));

    % Column vector to hold the constants of the equations
    const = zeros(3 * (n - 1), 1);
    % constructing equations according to the first property
    % Use a currentInterval variable to keep track of what intervals we are
    % dealing with

    % Use currentEquation to keep track of which equation we are
    % constructing

    currentEquation = 1;
    % Applying endpoint conditions (natural)
    eq(currentEquation, 2) = 2;
    currentEquation = currentEquation + 1;
    for currentInterval=1:n-1
        % Construct equation according to the first property
        xdiff = x(currentInterval + 1) - x(currentInterval);
        ydiff = y(currentInterval + 1) - y(currentInterval);
        % bi
        eq(currentEquation, 3 * (currentInterval - 1) + 1) = xdiff;
        % ci
        eq(currentEquation, 3 * (currentInterval - 1) + 2) = xdiff.^2;
        % di
        eq(currentEquation, 3 * (currentInterval - 1) + 3) = xdiff.^3;
        % constant
        const(currentEquation) = ydiff;
        currentEquation = currentEquation + 1;

        if currentInterval ~= n - 1
            % Construct equation according to the second property
            eq(currentEquation, 3 * (currentInterval - 1) + 1) = 1;
            eq(currentEquation, 3 * (currentInterval - 1) + 2) = 2 * xdiff;
            eq(currentEquation, 3 * (currentInterval - 1) + 3) = 3 * xdiff.^2;
            eq(currentEquation, 3 * (currentInterval - 1) + 4) = -1;
            currentEquation = currentEquation + 1;
            % Construct the equation according to the third property
            eq(currentEquation, 3 * (currentInterval - 1) + 2) = 2;
            eq(currentEquation, 3 * (currentInterval - 1) + 3) = 6 * xdiff;
            eq(currentEquation, 3 * (currentInterval - 1) + 5) = -2;
            currentEquation = currentEquation + 1;
        end

        if currentInterval == n - 1
            % Apply last endpoint condition (natural)
            eq(currentEquation, 3 * (currentInterval - 1) + 2) = 2;
            eq(currentEquation, 3 * (currentInterval - 1) + 3) = 6 * xdiff;
```

```
                currentEquation = currentEquation + 1;
            end
        end
        result = eq\const;
end
```