Lecture Module - Numerical Integration and Curve Fitting

ME3001 - Mechanical Engineering Analysis

Mechanical Engineering

Tennessee Technological University

Module 5 - Numerical Integration and Curve Fitting

Module 5 - Numerical Integration and Curve Fitting

- Topic 1 - Overview and Motivation
- Topic 2 - Linear Regression
- Topic 3 - Interpolation and Splines
- Topic 4 - Lagrange Polynomials

Topic 1 - Overview and Motivation

- Problem Definition
- Engineering Applications
-
-

Problem Definition

What is curve fitting?

- various techniques to fit a curve or function to discrete data

- "Data is often given for discrete values along a continuum. However, you may require estimates at points between the discrete values" - Numerical Methods for Engineers, Chapra and Canale

- additional problem is to find a simpler form of a complicated function by fitting function to data sampled from original function

Problem Definition

Two General Approaches

- 1) Given data with random error, find a single curve that represents the overall trend of the data.

  - "Because any individual data point may be incorrect, we make no effort to intersect every point" - Numerical Methods for Engineers, Chapra and Canale

  - Common method is *regression* (LSR)

- 2) Given data assumed to be precise or specified, find a curve that directly passes through each data point

  - Known as *interpolation*, *extrapolation*

Engineering Applications

Example Applications in Engineering

- Calibration Curves, Sensors and Instrumentation
- Table Interpolation, Mechanics, Thermo, Statistics
- Velocity Profile Generation, Dynamics of Machinery, Robotics

Two General Problems

- Trend Analysis - predictions from dataset using interpolation polynomial or LSR
- Hypothesis Testing - compare predicted to measured data for model performance or selection

Topic 2 - Linear Regression

- Overview
- Fit Criteria
- Linear Least Squares
- MATLAB Example

Overview

Consider fitting a straight line to a dataset

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$$

with a function

$$y = a_o + a_1$$

A value y can be defined in terms of the function with an error term e

$$y = a_0 + a_1 + e$$

This can be rearranged to show the **error** as

$$e = y - a_0 - a_1 x$$

The goal is to find the coefficients of a function that minimizes the error

Fit Criteria

To find the coefficents of the fit line, the minimization objective must be considered carefully. You might consider fitting a model that mimizes the error directly, but this will not work. The absolute value approach is also problematic.

- $\Sigma_{i=1}^{n} e_i = (y_i - a_0 - a_1 x_i)$
- $\Sigma_{i=1}^{n} |e_i| = |y_i - a_0 - a_1 x_i|$

To solve these issues, the common technique is to ________________ the error.

- $\Sigma_{i=1}^{n} e_i^2 = (y_i - a_0 - a_1 x_i)^2$

Linear Least Squares

To fit a straight line to the data, we must find the values a_o and a_1 that minimize the square of the error. First find the partial derivatives of the sqaured error and set these equal to zero

$$S_r = \Sigma_{i=1}^{n} e_i^2 = (y_i - a_0 - a_1 x_i)^2$$

$$\frac{\delta S_r}{\delta a_0} = -2\Sigma \left(y_i - a_0 - a_1 x_i\right)$$

$$\frac{\delta S_r}{\delta a_1} = -2\Sigma \left[\left(y_i - a_0 - a_1 x_i\right) x_i\right]$$

$$0 = \Sigma y_i - \Sigma a_0 - \Sigma a_1 x_i$$

$$0 = \Sigma y_i x_i - \Sigma a_0 x_i - \Sigma a_i x_i^2$$

Linear Least Squares

Use $\Sigma a_0 = na_0$ and the resulting equations can be solved as a linear system in terms of the coefficients a_0, a_1, and number of data points n.

$$0 = \Sigma y_i - \Sigma a_0 - \Sigma a_1 x_i$$

$$0 = \Sigma y_i x_i - \Sigma a_0 x_i - \Sigma a_i x_i^2$$

This leads to the standard equations

$$a_1 = \frac{n\Sigma x_i y_i - \Sigma x_i \Sigma y_i}{n\Sigma x_i^2 - (\Sigma x_i^2)}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

This alternate form can be found by multipying by $1 = \frac{-1}{-1}$

$$a_1 = \frac{\Sigma x_i \Sigma y_i - n\Sigma x_i y_i}{(\Sigma x_i)^2 - n\Sigma x_i^2}$$

$$a_0 = \frac{\Sigma x_i \Sigma x_i y_i - \Sigma x_i^2 \Sigma y_i}{(\Sigma x_i)^2 - n\Sigma x_i^2}$$

MATLAB Example

This standard technique is built into the MATLAB function *polyfit*. This function can also be used for higher order regression lines.

```
% ME3001, TNTech, Tristan Hill, October 29, 2024
% Curve fitting with Linear Regression
% this program will
% 1) generate dataset with random noise
% 2) find best fit using 'linear least sqaures regression' from
    eqs in notes
% 3) find best fit using LSR with MATLAB polyfit()
clear; clc; close all
```

MATLAB Example

```
% step 1) - generate dataset
m=-3; b=1.5;
error_scale=5;

xdata=-5:.5:5;
n=length(xdata);
ydata=m*xdata+b+rand(1,n)*error_scale;

figure(1); hold on
plot(xdata,ydata,'o')
grid on
```

MATLAB Example

```
% step 2) - fit line with LSR equations
a1=(n*sum(xdata.*ydata)-sum(xdata)*sum(ydata))/...
    (n*sum(xdata.^2)-sum(xdata.^2))
a0=sum(ydata)/n

% compare with equations from ME3023
a1=(sum(xdata)*sum(ydata)-n*sum(xdata.*ydata))/...
    (sum(xdata)^2-n*sum(xdata.^2))
a0=(sum(xdata)*sum(xdata.*ydata)-sum(xdata.^2)*sum(ydata))/...
    (sum(xdata)^2-n*sum(xdata.^2))
```

MATLAB Example

```
% compute and plot values on the best fit line
xfit=-5:.1:5;
yfit=a1*xfit+a0;

plot(xfit,yfit,'-')

% step 3) - fit line with LSR in MATLAB
A=polyfit(xdata,ydata,1) % get second the coefficients

pfit=A(2)+A(1)*xfit; % calculate points on curve
plot(xfit,pfit,':g','LineWidth',5)
```

Download *linear_regression_example1.m* for the complete program.

Topic 3 - Interpolation and Splines

- Polynomial Interpolation Functions
- Polynomial Splines
- Linear Splines
- Cubic Splines

Polynomial Interpolation Functions

Polynomial Interpolation Functions

Polynomial Splines

Polynomial Splines

Linear Splines

Fit a **spline** function consisting of multiple *linear* functions to a set of data.

- the spline must pass through n data points $(x_i, f_i)_{i=1,2,\ldots,n}$
- $n-1$ intervals are defined by spline functions $s_i(x)_{i=1,2,\ldots,n-1}$

$$s_i\,(x) = a_i + b_i(x - x_i)$$

$$a_i = f_i$$

$$b_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$

- the $(x - x_i)$ term handles the shift to the i^{th} spline function
- substitute b_i into $s_i(x)$ to get the following description of the spline

$$s_i\,(x) = f_i + \left(\frac{f_{i+1} - f_i}{x_{i+1} - x_i}\right)(x - x_i)$$

Derivation: Applied Numerical Methods with MATLAB Steven Chapra

Linear Splines

Cubic Splines

"Cubic splines are most commonly used in practice"

Fit a **spline** function consisting of multiple *cubic* functions to the data

- the spline must pass through n data points $(x_i, f_i)_{i=1,2,\ldots,n}$
- $n-1$ intervals are defined by spline functions $s_i(x)_{i=1,2,\ldots,n-1}$

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

- coeficients a_i, b_i, c_i, d_i must be found $\implies 4(n-1)$ unknowns
- the slope at each point must match for a smooth spline
- two additional conditions are required due to no slope match at ends

$$2 * (n-1) + 2 * (n-1) - 2 + 2 = 4(n-1)$$

Cubic Splines

The functions passes through all the data points

$$s_i\left(x\right) = a_i + b_i\left(x - x_i\right) + c_i\left(x - x_i\right)^2 + d_i\left(x - x_i\right)^3$$

$$s_i\left(x_i\right) = f_i$$

$$f_i = a_i + b_i\left(x_i - x_i\right) + c_i\left(x_i - x_i\right)^2 + d_i\left(x_i - x_i\right)^3 = a_i$$

The a_i coeficients can be replaced with the function values f_i

$$f_i = a_i$$

$$s_i\left(x\right) = f_i + b_i\left(x - x_i\right) + c_i\left(x - x_i\right)^2 + d_i\left(x - x_i\right)^3$$

Define the i^{th} stepsize for convenience

$$h_i = x_{i+1} - x_i$$

Cubic Splines

The function values are equal at each point

$$f_i + b_i\left(h_i\right) + c_i\left(h_i\right)^2 + d_i\left(h_i\right)^3 = f_{i+1}$$

The slope (first derivative) matches at each point between intervals

$$s_i'\left(x\right) = b_i + 2c_i\left(x - x_i\right) + 3d_i\left(x - x_i\right)^2$$

$$b_i + 2c_i\left(x_{i+1} - x_i\right) + 3d_i\left(x_{i+1} - x_i\right)^2$$

$$= b_{i+1} + 2c_{i+1}\left(x_{i+1} - x_{i+1}\right) + d_{i+1}\left(x_{i+1} - x_{i+1}\right)$$

$$\implies b_i + 2c_ih_i + 3d_ih_i^2 = b_{i+1}$$

The second derivative is also matches at the nodes for a *natural spline*

$$s_i''\left(x\right) = 2c_i + 6d_i\left(x - x_i\right)$$

$$2c_i + 6d_ih_i = 2c_{i+1} + 6d_i\left(x_{i+1} - x_{i+1}\right) \implies d_i = \frac{c_{i+1} - c_i}{3h_i}$$

Cubic Splines

Subsitute d_i and solve for b_i

$$f_i + b_i h_i + c_i h_i^2 + \left(\frac{c_{i+1} - c_i}{3h_i}\right) h_i^3 = f_{i+1}$$

$$f_i + b_i h_i + \frac{h_i^2}{3}\left(2c_i + c_{i+1}\right) = f_{i+1}$$

$$\implies b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3}\left(2c_i + c_{i+1}\right)$$

repeat for derivative condition equation

$$b_i + 2c_i h_i + 3\left(\frac{c_{i+1} - c_i}{3h_i}\right) h_i^2 = b_{i+1} \implies b_{i+1} = b_i + h_i\left(c_i + c_{i+1}\right)$$

Cubic Splines

Use the result from above

$$b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3}\left(2c_i + c_{i+1}\right)$$

this should hold for all nodes ..., $i-1, i, i+1, \ldots$
reduce the index by 1

$$b_{i-1} = \frac{f_i - f_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{3}\left(2c_{i-1} + c_i\right)$$

repeat for the result from the derivative condition to get

$$b_i = b_{i+1} + h_{i-1}\left(c_{i-1} + c_i\right)$$

Cubic Splines

Use the result from above

$$b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3}\left(2c_i + c_{i+1}\right)$$

this should hold for all nodes ..., $i-1, i, i+1, \ldots$
reduce the index by 1

$$b_{i-1} = \frac{f_i - f_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{3}\left(2c_{i-1} + c_i\right)$$

repeat for the result from the derivative condition to get

$$b_i = b_{i+1} + h_{i-1}\left(c_{i-1} + c_i\right)$$

Cubic Splines

Combine to find final equation

$$\frac{f_{i+1}-f_i}{h_i}-\frac{h_i}{3}\left(2c_i+c_{i+1}\right)=\frac{f_i-f_{i-1}}{h_{i-1}}-\frac{h_{i-1}}{3}\left(2c_{i+1+c_i}\right)+h_{i-1}\left(c_{i-1}+c_i\right)$$

Topic 3 - Lagrange Polynomials

-
-
-
-