EGR 5110: Homework #5

Due on May 5th, 2024 at 11:59pm

Professor Nissenson

Francisco Sanudo

Contents

Background

Homework #5 required the construction of a numerical integrator using quadratic splines to calculate the area under a general set of data points that could be unevenly spaced.

Given the following set of inputs:

1. `x` – Time vector corresponding to velocity data
2. `fx` – Velocity vector
3. `tinstant` – Time at which to determine the instantaneous velocity
4. `t1 & t2` – Initial and final times for distance calculations

The numerical integrator was expected to return the following outputs:

1. `totaldist` – Total distance traveled from the beginning to the end of the time vector
2. `vinstant` – Instantaneous velocity at the specified time `tinstant`
3. `subdist` – Distance traveled between the specified times `t1` and `t2`

Testing The Integrator

To assess the validity of the integrator, it's important to test it against a function with a known analytical solution. Consider the following quadratic function:

$$f(x) = x^2 \tag{1}$$

Integrating from x_1 to x_2 yields:

$$\begin{aligned}\int_{x_1}^{x_2} x^2 &= \frac{1}{3}x^3\Big|_{x_1}^{x_2} \\ &= \frac{1}{3}x_2^3 - \frac{1}{3}x_1^3\end{aligned}$$

For example, let's test the integrator using the following:

```
x = [1 2 6 7 10 13 20 25];        % Time vector
fx = x.^2;                        % Velocity vector
tinstant = 8;                     % Instantaneous velocity query time
t1 = 3;                           % Start time for distance calculation
t2 = 14;                          % End time for distance calculation
```

Listing 1: Integrator Inputs

Integrating (1) over the entire time range (1 to 25 seconds) yields:

$$\int_{1}^{25} x^2 = \frac{1}{3}25^3 - \frac{1}{3}1^3 = 5208$$

Integrating (1) over the subset time range (3 to 14 seconds) yields:

$$\int_{3}^{14} x^2 = \frac{1}{3}14^3 - \frac{1}{3}3^3 = 905.6667$$

The velocity at `tinstant` (8 seconds) is:

$$f(8) = 8^2 = 64$$

Numerical Integration Accuracy

The implementation of the quadratic spline integrator introduces several important considerations and potential sources of discrepancy when computing the integral of a velocity function over specified intervals. One primary consideration is the accuracy of numerical integration methods employed in the code. The integrator utilizes quadratic splines to approximate the velocity function between data points. However, the accuracy of this approximation relies on the number of spline segments (N) and the degree of the polynomial used. With quadratic splines, while providing a reasonable approximation, the method might not capture rapid changes in the velocity function accurately, particularly over larger intervals. In addition, error is expected because we assume the second derivative is 0 at the first data point, making the first spline a line instead of curve.

The following shows the output from the numerical integrator:

```
>> [totaldist,vinstant,subdist] = quadspline(x,fx,tinstant,t1,t2)

totaldist =

       5210.1


vinstant =

        64.03


subdist =

       907.27
```

Listing 2: Integrator Output

The results shown in Listing 2 indicate that the numerical solution aligns very closely to the analytical one, with some small error. Compared to the analytical solution, `totaldist` is within 0.04%, `vinstant` is within 0.04%, while `subdist` fell within 0.17%. For the given set of data points, the integrator showed great performance but could be improved by introducing a larger set of data points (i.e., increasing the number of quadratic spline segments), or emplying higher-order polynomial approximations (e.g. cubic splines) if necessary.

Figure 1 depicts the quadratic spline fit performed on the given set of data points.

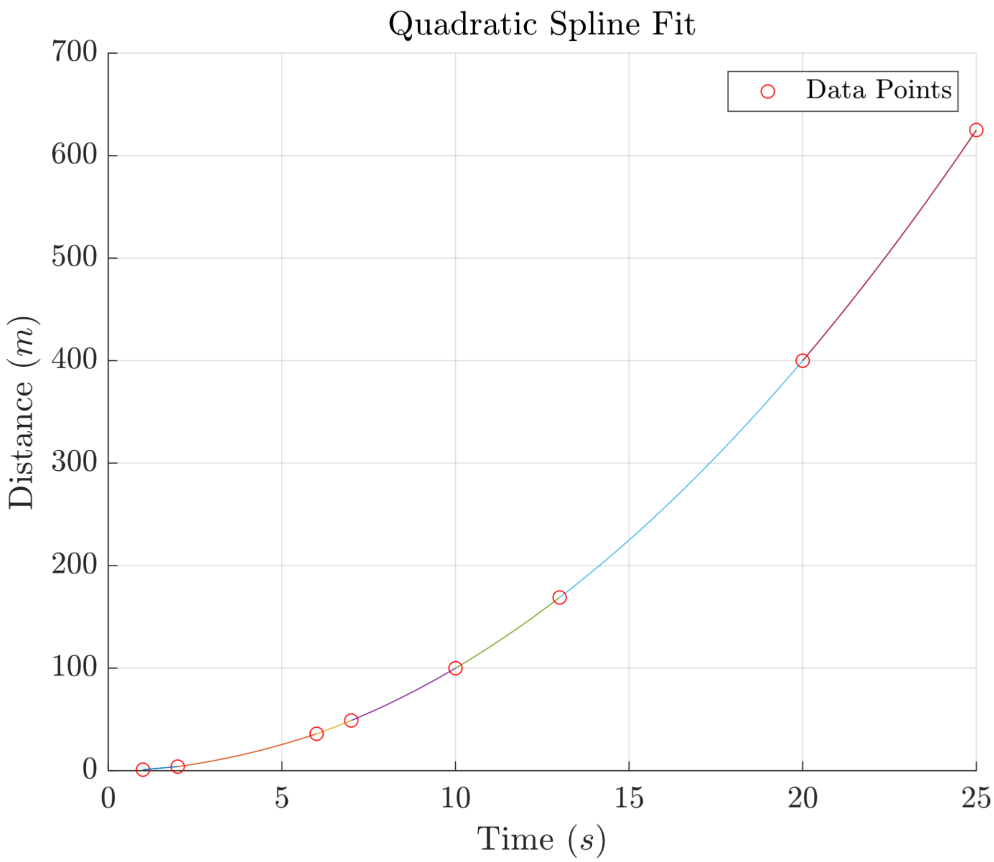

Figure 1: Quadratic Spline Interpolation Between Data Points

Quadratic Spline Integrator – MATLAB Code

```matlab
% Written by: Francisco Sanudo
% Date: 4/26/24
%
% PURPOSE
% quadspline interpolates velocity data using quadratic splines and provide
% computations related to these splines, including:
%
%   1. Interpolation: Fit quadratic splines to given time and velocity data
%   points
%
%   2. Integration: Compute the integral of velocity over specified time
%   intervals to determine distances traveled
%
%   3. Velocity Query: Determine the instantaneous velocity at a queried
%   time
%
%   4. Distance Calculation: Compute the distance traveled between two
%   specified times
%
% INPUTS
% - x        : Time vector corresponding to velocity data
% - fx       : Velocity vector
% - tinstant : Time at which to determine the instantaneous velocity
% - t1 & t2  : Initial and final times for distance calculations
%
% OUTPUTS
% - totaldist : Total distance traveled from the beginning to the end of the time vector
% - vinstant  : Instantaneous velocity at the specified time 'tinstant'
% - subdist   : Distance traveled between the specified times 't1' and 't2'
%
% EXAMPLE
% x = [0, 1, 2, 3];               % Time vector
% fx = [0, 2, 1, 3];              % Velocity vector
% tinstant = 1.5;                 % Instantaneous velocity query time
% t1 = 0.5;                       % Start time for distance calculation
% t2 = 2.5;                       % End time for distance calculation
%
% [totaldist, vinstant, subdist] = quadspline(x, fx, tinstant, t1, t2);
%
% OTHER
% .m files required              : MAIN.m (calling script)
% Files required (not .m)        : none
% Built-in MATLAB functions used : numel, zeros, fplot
% User-defined functions         : applyFigureProperties
%
% REFERENCES
% Numerical Integration (notes), P. Nissenson
%

function [totaldist,vinstant,subdist] = quadspline(x,fx,tinstant,t1,t2)

```

```
% Initialize variables
N = numel(x) - 1;    % total number of splines
b = zeros(3*N,1);    % column matrix of knowns
A = zeros(3*N,3*N); % coefficient matrix

%% Generate coefficients of splines

j = 1; % spline index
k = 1; % starting column index

% Condition #1: Functions are continuous at interior knots
for i = 2:2:2*N
    A(i,k:k+2)    = [x(j)^2, x(j), 1];
    b(i)          = fx(j);
    j             = j + 1;             % increment spline number
    A(i+1,k:k+2) = [x(j)^2, x(j), 1];
    b(i+1)        = fx(j);
    k             = k + 3;             % increment starting column index
end

j = 1; % spline index [reset]
k = 1; % starting column index [reset]

% Condition #2: First and last functions pass through end knots
%   <Satisfied in loop above>

% Condition #3: First derivatives are continuous at interior knots
for i = 2*N+2:3*N
    A(i,k:k+4) = [2*x(j), 1, 0, -2*x(j), -1];
    j          = j + 1;
    k          = k + 3;
end

% Condition #4: Assume second derivative is 0 at first knot
A(1,1) = 1;

% Solve for unknown coefficients
c = A\b;

%% Plot splines

j = 1;  % spline index [reset]

% Create figure and apply figure properties
f = figure;
position = [0.2, 0.2, 0.5, 0.6];
applyFigureProperties(f, position)

hold on % plot over same axes

% Begin loop to plot all splines
for i = 1:N
    spline = @(X) c(j)*X.^2 + c(j+1).*X + c(j+2); % create spline function handle
```

```
    fplot(spline,[x(i) x(i+1)])                    % plot current spline between x(i) and
        x(i+1)

    j = j + 3; % increment to next set of coefficients
end

% Plot data points
h = plot(x,fx,'ro');

% Axis properties
set(gca,'TickLabelInterpreter','latex')
title('Quadratic Spline Fit')
xlabel('Time ($s$)');
ylabel('Distance ($m$)')
legend(h, 'Data Points')
grid on

%% Calculate total distance traveled

% Initialize variables
totaldist = 0;
k = 1;          % coefficient index

% Integration of the velocity function over the current segment

for j = 2:N+1 % N+1 data points where N is the number of splines
    I = (c(k)*x(j)^3/3 + c(k+1)*x(j)^2/2 + c(k+2)*x(j)) ...
        - (c(k)*x(j-1)^3/3 + c(k+1)*x(j-1)^2/2 + c(k+2)*x(j-1));
    totaldist = totaldist + I;

    k = k + 3; % increment to next set of coefficients
end

%% Calculate velocity at tinstant

% Find the relevant spline that contains tinstant
spline_idx = findSpline(x,tinstant);

% Starting coefficient index
k = 3*spline_idx - 2;

% Calculate the velocity at tinstant
vinstant = c(k)*tinstant^2 + c(k+1)*tinstant + c(k+2);

%% Calculate the distance traveled from t1 to t2

% Find the relevant spline that contains t1 and t2
spline_idx_t1 = findSpline(x,t1);
spline_idx_t2 = findSpline(x,t2);

% Initialize variables for distance calculation
subdist = 0;
k = 3*spline_idx_t1 - 2;  % Starting coefficient index for t1 segment
```

```matlab

% Loop through spline segments from t1 to t2
for j = (spline_idx_t1):(spline_idx_t2)

    % Integration of the velocity function over the current segment

    % [t1, x(j+1)]
    if j == spline_idx_t1
        I = (c(k)*x(j+1)^3/3 + c(k+1)*x(j+1)^2/2 + c(k+2)*x(j+1)) ...
            - (c(k)*t1^3/3 + c(k+1)*t1^2/2 + c(k+2)*t1);
        subdist = subdist + I;
    % [x(j), t2]
    elseif j == spline_idx_t2
        I = (c(k)*t2^3/3 + c(k+1)*t2^2/2 + c(k+2)*t2) ...
            - (c(k)*x(j)^3/3 + c(k+1)*x(j)^2/2 + c(k+2)*x(j));
        subdist = subdist + I;
    % [x(j), x(j+1)]
    else
        I = (c(k)*x(j+1)^3/3 + c(k+1)*x(j+1)^2/2 + c(k+2)*x(j+1)) ...
            - (c(k)*x(j)^3/3 + c(k+1)*x(j)^2/2 + c(k+2)*x(j));
        subdist = subdist + I;
    end

    % Update coefficient index for the next segment
    k = k + 3;
end

end

%-----------------------------------------------------------------------

function idx = findSpline(x, tinstant)
% Finds the spline segment index that contains tinstant

% Initialize segment index
idx = 0;

% Check each interval [x(i), x(i+1)] to find the relevant segment
for i = 1:numel(x)-1
    if tinstant >= x(i) && tinstant < x(i+1)
        idx = i;
        break;  % Exit loop once segment is found
    end
end

% If tinstant is out of range (before first element or after last element)
if isempty(idx)
    error('tinstant is out of the range of input time vector x.');
end

end

%------------------------------------------------------------------------
```

```
function applyFigureProperties(figHandle, position)
set(figHandle, ...
    'Units', 'normalized', ...
    'Position', position, ...
    'DefaultTextInterpreter', 'latex', ...
    'DefaultLegendInterpreter', 'latex', ...
    'DefaultAxesFontSize', 14);
end
```

Listing 3: Quadratic Spline Integrator