CPE 381: Fundamentals of Signals and Systems for Computer Engineers

02 Continuous & Discrete Representation

Fall 2025

Rahul Bhadani

Outline

Continuous and Discrete Representations

Continuous and Discrete Representations

- **Continuous-time signals:** Depend continuously on time.
- **Discrete-time signals:** Sequences of measurements typically made at uniform times.
- **Sampling process:**

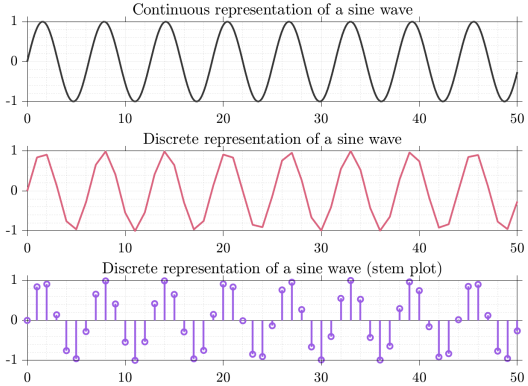
$$x[n] = x(nT_s) = x(t)|_{t=nT_s}$$

- **Example:**
  - Analog signal:

$$x(t) = 2\cos(2\pi t)$$

  - Sampled at $T_{s1} = 0.1$ s: $x_1[n] = 2\cos(2\pi n/10)$
  - Sampled at $T_{s2} = 1$ s: $x_2[n] = 2\cos(2\pi n) = 2$
- **Key point:** Choosing an appropriate sampling period T_s is crucial to preserve information.

Sampling Continuous Time Signals



$x[n] = x(nT_s)$, T_s = sample time.

Code for the figure: https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/sampled_sine_wave.m

Sampling Process

- $x[n] = x(nT_s) = x(t)|_{t=nT_s}$
- This equation represents the **sampling process** where a continuous-time signal $x(t)$ is sampled at uniform intervals T_s to produce a discrete-time signal $x[n]$.
- **Key Points:**
  - n is an integer representing the sample index.
  - T_s is the sampling period.
  - The continuous-time signal $x(t)$ is sampled at $t = nT_s$.

Derivatives and Finite Differences

Continuous-Time Derivatives:

- Derivatives measure the rate of change of a function.
- Defined as the limit of the difference quotient as the interval approaches zero.
- Represented as $\frac{dy}{dt}$ for a function $y(t)$.

Finite Differences

Discrete-Time Derivatives:

- Finite differences approximate derivatives for discrete-time signals.
- Forward difference: $\Delta x[n] = x[n+1] - x[n]$.
- Backward difference: $\Delta x[n] = x[n] - x[n-1]$.

$x[n] = x(nT_s)$ when looking at continuous to discrete domain where T_s is the sampling time, n is any positive integer. We will look at this much later in detail.

Central Differences

Central Difference:

- Provides a more accurate approximation.
- Defined as $\delta x[n] = \frac{x[n+1]-x[n-1]}{2}$.
- Reduces error compared to forward and backward differences.

Relation between the Derivative and Finite Difference

The derivative and the finite-difference operators are not the same. In the limit, we have:

$$\frac{dx(t)}{dt}\Big|_{t=nT_s} = \lim_{T_s \to 0} \frac{\Delta[x(nT_s)]}{T_s}$$

Applications

Applications of Finite Differences:

- Used in numerical methods for solving differential equations.
- Essential in digital signal processing and control systems.
- Helps in approximating continuous-time derivatives in discrete systems.

Discrete Integration

Recall, the integration in the continuous domain is

$$I(t) = \int_{t_0}^{t} x(\tau) d\tau$$

then, we have

$$\frac{d}{dt} \int_{t_0}^{t} x(\tau) d\tau = x(t)$$

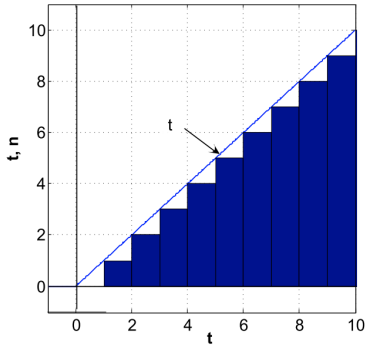
If we use D for the derivative operator, then D^{-1} is the integration operator.

$$D[D^{-1}[x(t)]] = x(t))$$

Example

- Computational integration using sums:

$$\int_0^{10} t\,dt = \frac{t^2}{2}\Big|_0^{10} = 50$$



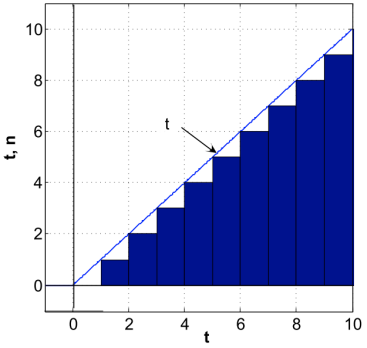
Discrete Approximation

- Approximation using pulses ($T_s = 1$)(rectangle of width $T_s = 1$, height = n):

$$\sum_{n=0}^{9} p[n] = \sum_{n=0}^{9} n = 45$$

- Generalized sum:

$$\sum_{n=0}^{N-1} n = \frac{N \times (N-1)}{2}$$



Improved Discrete Approximation

- Using $T_s = 10^{-3}$:

$$\sum_{n=0}^{(10/T_s)-1} nT_s^2 = \sum_{n=0}^{(10/T_s)-1} n10^{-6} = 49.995$$

The height of each pulse is nT_s and the width is T_s.

Hence, the sampling time (or its inverse – sampling frequency matters).

Numerical Computation in MATLAB

Signal Generation in MATLAB

1. Choosing time-points
2. Uniformly sampled (ideal) vs non-uniformly sampled (real-life)
3. Specification: frequency, or sampling-time
4. Specify function
5. We primarily study uniformly-sampled signals in this course.

Example in MATLAB

```
%% Signal with Fixed Sampling-time
sampling_time = 0.5; % 0.5 seconds
final_time = 10; % 10 seconds
% Generate time points on which to evaluate the function
t = 0:sampling_time:final_time;
x = sin(t);
% Plotting
f = figure;
f.Position = [744   358   914   592]; % Position of figure on the screen
hold on;
plot(t, x, 'LineStyle','-', 'LineWidth',2,'Marker','o', ...
    'Color','#426642', ...
    'DisplayName', 'Sine Wave with Uniform Sampling');
```

Example in MATLAB

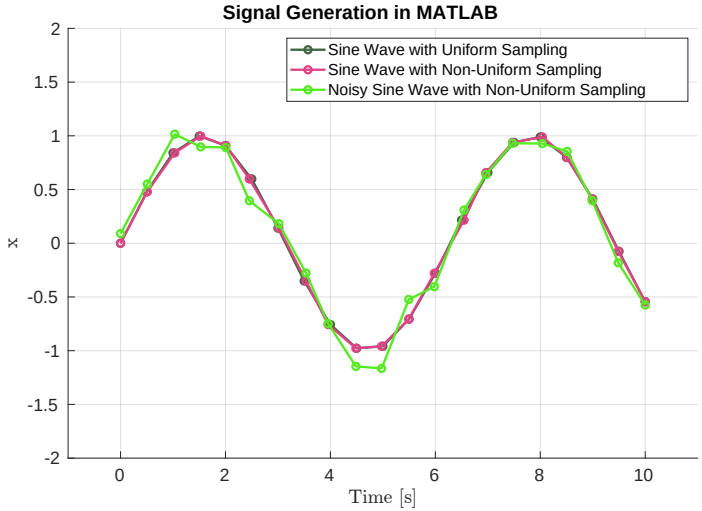
```
%% Signal with non-uniform sampling time

% Add small random perturbations
noise_amplitude = 0.05; % 10% of sampling_time
random_noise = (rand(size(t)) - 0.5) * 2 * noise_amplitude;
t_nonuniform = t + random_noise;
% ensure the first and last points of time-index remain same
t_nonuniform(1) = 0;
t_nonuniform(end) = final_time;
plot(t_nonuniform, x, 'LineStyle','-', 'LineWidth',2, 'Marker','o', ...
    'Color','#E34288', ...
    'DisplayName','Sine Wave with Non-Uniform Sampling');
```

Example in MATLAB

```
% Signal with gaussian noise

mean = 0.0; std_dev = 0.1;
gaussian_noise = std_dev * randn(size(t));
noisy_x = x + gaussian_noise;
plot(t_nonuniform, noisy_x, 'LineStyle','-',  'LineWidth',2, 'Marker','o', ...
    'Color','#56EA23', 'DisplayName','Noisy Sine Wave with Non-Uniform Sampling');
xlabel('Time [s]', 'Interpreter', 'latex', 'FontSize', 14);
xlim([-1, final_time+1]);
ylim([-2, 2]);
ylabel('x', 'FontSize',15, 'Interpreter','latex');
title('Signal Generation in MATLAB');
% Set tick font size
set(gca, 'FontSize', 15); legend; grid on;
```

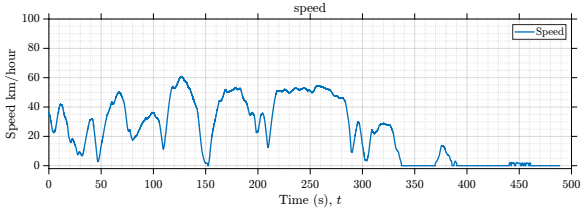


Full code: https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_Signal_Generation.m

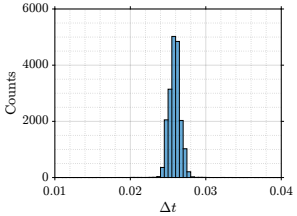
Working with Real-life Signal

- Real-life signals are noisy, i.e., not smooth, may not be differentiable at every time points.
- May have high frequency noise – usually due to another process
- May have low frequency noise – may be due to missing data, packet drop, inefficiency acquisition of data
- Non-uniform sampling time

Example of Real-life Signal



Histogram of Δt



- Speed data was collected from Toyota RAV4.
- Mean $\Delta t = 0.025826$.
- Code at https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_real_world_speed_data.m
- Data at https://github.com/rahulbhadani/CPE381_FA25/blob/main/Data/vel.csv

Pre-processing before we can use real-life Signals

What do we want and what do we need to do?
What do we want?

- Signals at uniformly sampled time points.
- Remove noise.
- Resample at different sampling time or frequency.

What can we do to meet our requirements?

- Realign time points by assuming sampling time as constant, equal to mean sampling time.
- Use interpolation methods for evaluation at new uniformly sampled points. Other techniques using frequency domain exists that we will study.
- Removing noise is equivalent of smoothing signal data. We will learn in later chapters on smoothing a signal using new tools developed over the course of this semester.

Interpolation Methods

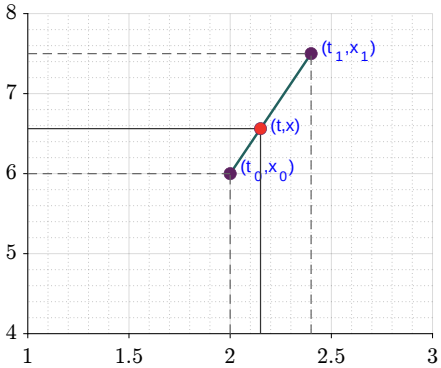
Interpolation finds the value between any two points based on some approximation by fitting a line or a curve.

- We create uniformly sampled time-points, and approximate signal values at those points to create uniformly-sampled signals.

Linear Interpolation

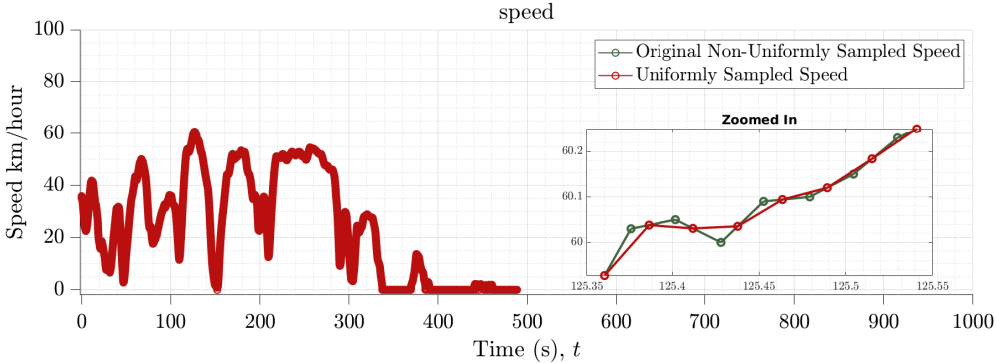
Linear Interpolation

Two consecutive points are used to fit a line, and and a signal value is computed over an intermediate time point. The line is called **interpolant**.



- A line equation is $x = mt + c$, m is the slope of the line, and c is the intercept.
- Two-point line equation can equivalently be written as $x = x_0 + \dfrac{(x_1 - x_0)}{(t_1 - t_0)}(t - t_0)$
- $m = \dfrac{(x_1 - x_0)}{(t_1 - t_0)}$, $c = x_0 - mt_0$.
- For every original consecutive time points $[t_k, t_{k+1}]$, compute the localized line equation, and then, for a new time points $t_k \leq t \leq t_{k+1}$, compute the signal value x.

Linear Interpolation: Implementation in MATLAB



Code: https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_Linear_Interpolation.m

Cubic Spline Interpolation

Cubic Spline Interpolation

Instead of a line, we could fit a polynomial of degree $n + 1$ using n data points. However, we must limit to low-degree polynomials to avoid spurious approximations at new time-points.

How many data points do we need to create a cubic spline interpolant?

Cubic Spline Interpolation

Cubic function is $f(t) = at^3 + bt^2 + ct + d$.

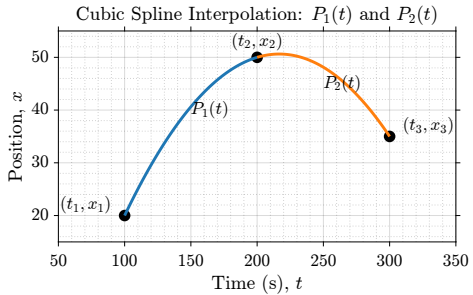
How do we choose coefficients a, b, c, d.

We could use four successive points to fit a curve, but that won't give as smooth fit on the entire data.

Why?

Cubic Spline Interpolation

Consider interpolants at two successive points, $P_1(t)$ and $P_2(t)$.



How do we fit a curve of degree 3 with only two points?

Answer: We utilize continuity and smoothness that we desire at knot points.

$(t_1, x_1), (t_2, x_2), \cdots$ are knot points.

Cubic Spline Interpolation

We write n cubic polynomails as $P_i(t) = a_i(t - t_i)^3 + b_i(t - t_i)^2 + c_i(t - t_i) + d_i$, $i = 1, 2, \cdots, n$
We desire

1. $P_1'(t_2) = P_2'(t_2)$ at interior knots.
2. $P_1''(t_2) = P_2''(t_2)$ at interior knots.

Consider $n + 1$ data points: $(t_1, x_1), (t_2, x_2), \cdots, (t_{n+1}, x_{n+1})$. We will have a total of n interpolant.
Hence, to determine coefficient of two consecutive cubic spline interpolant $P_i(t)$ and $P_{i+1}(t)$, we need,

1. $P_1(t_1) = x_1$, $P_n(t_{n+1}) = x_{n+1}$
2. $P_{i+1}(t_{i+1}) = P_i(t_{i+1})$, $i = 1, 2, \cdots, n - 1$
3. $P_i(t_i) = x_i, i = 2, 3, \cdots, n$
4. $P_i'(t_{i+1}) = P_{i+1}'(t_{i+1})$, $i = 1, 2, \cdots, n - 1$
5. $P_i''(t_{i+1}) = P_{i+1}''(t_{i+1})$, $i = 1, 2, \cdots, n - 1$

A total of $4n - 2$ equations. We also need boundary conditions.
Boundary conditions indicate the manner in which the first spline departs from the first data point and the last spline arrives at the last data point.

Cubic Spline Interpolation

Coefficient to determine: a_i, b_i, c_i, d_i for $i = 1$ to n.

$$P_i(t) = a_i(t - t_i)^3 + b_i(t - t_i)^2 + c_i(t - t_i) + d_i \tag{1}$$

Using this, we have:

$$P_i(t_i) = d_i = x_i \tag{2}$$

Let $h_i = t_{i+1} - t_i$, the spacing between time points.

Cubic Spline Interpolation

Using the second condition, we have:

$$P_{i+1}(t_{i+1}) = P_i(t_{i+1}) \tag{3}$$

$$\begin{aligned} d_{i+1} &= a_i(t_{i+1} - t_i)^3 + b_i(t_{i+1} - t_i)^2 + c_i(t_{i+1} - t_i) + d_i \\ d_{i+1} &= a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i, \quad i = 1, \ldots, n-1 \end{aligned} \tag{4}$$

If we define $d_{n+1} = y_{n+1}$, then

$$d_{i+1} = a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i, \quad i = 1, \ldots, n \tag{5}$$

Cubic Spline Interpolation

Taking the first derivative of $P_i(t)$ and applying the 4th condition:

$$c_{i+1} = 3a_i h_i^2 + 2b_i h_i + c_i, \quad i = 1, 2, \cdots, n-1 \tag{6}$$

If we define, $c_{n+1} = P'_(t_{n+1})$, then

$$c_{i+1} = 3a_i h_i^2 + 2b_i h_i + c_i, \quad i = 1, 2, \cdots, n \tag{7}$$

Cubic Spline Interpolation

Taking the second derivative of $P_i(t)$ and applying the 5th condition:

$$2b_{i+1} = 6a_i h_i + 2b_i$$
$$b_{i+1} = 3a_i h_i + b_i, \quad i = 1, \ldots, n-1 \tag{8}$$

If we define $b_{n+1} = \frac{1}{2} P_n''(t_{n+1})$, then

$$b_{i+1} = 3a_i h_i + b_i, \quad i = 1, \ldots, n \tag{9}$$

From this, we can express a_i as:

$$a_i = \frac{b_{i+1} - b_i}{3h_i} \tag{10}$$

Cubic Spline Interpolation

Putting Equation (10) into Equation (5) and Equation (7), we have,

$$d_{i+1} = \frac{1}{3}(2b_i + b_{i+1})h_i^2 + c_i h_i + d_i \quad i = 1, \ldots, n \tag{11}$$

and

$$\begin{aligned} c_{i+1} &= (b_i + b_{i+1})h_i + c_i, \quad i = 1, \ldots, n \\ c_i &= \frac{d_{i+1} - d_i}{h_i} - \frac{1}{3}(2b_i + b_{i+1})h_i \end{aligned} \tag{12}$$

Changing i to $i - 1$

$$c_{i-1} = \frac{d_i - d_{i-1}}{h_i} - \frac{1}{3}(2b_{i-1} + b_i)h_{i-1} \tag{13}$$

and from Equation (12),

$$c_i = (b_{i-1} + b_i)h_{i-1} + c_{i-1} \tag{14}$$

Cubic Spline Interpolation

Using Equation (12), and Equation (14), we can write:

$$b_{i-1}h_{i-1} + 2b_i(h_i + h_{i-1}) + b_{i+1}h_i = 3\left(\frac{d_{i+1} - d_i}{h_i} - \frac{d_i - d_{i-1}}{h_{i-1}}\right), \quad i = 2, \cdots, n \tag{15}$$

This describes a system of equations whose only uknowns are b_i.

However, Equation (15) generates a system of $(n-1)$ equations in $n+1$ unknowns $(b_1, \ldots, b_{n+1})$. We need two more equations that come from boundary conditions.

Cubic Spline Interpolation

$i = 1$ gives

$$c_1 = \frac{d_2 - d_1}{h_1} - \frac{1}{3}(2b_1 + b_2)h_1 \tag{16}$$

We can have two kind of boundary conditions: (i) Clamped Boundary Conditions where $P_1'(t) = p$, and $P_n'(t) = q$; (ii) Free Boundary Conditions where $P_1''(t) = 0$, and $P_n''(t) = 0$;

Cubic Spline Interpolation

For **Clamped Boundary Conditions**, $c_1 = P_1'(t) = p$, then

$$(2b_1 + b_2)h = \frac{3(d_2 - d_1)}{h_1} - 3p \tag{17}$$

From Equation (12),

$$c_{n+1} = (b_n + b_{n+1})h_n + c_n \tag{18}$$

and from the boundary condition,

$$\begin{aligned} c_{n+1} &= P_n'(t_{n+1}) = q \\ c_n &= q - (b_n + b_{n+1})h_n \end{aligned} \tag{19}$$

Cubic Spline Interpolation

Clamped Boundary Conditions

Equation (12), when $i = n$,

$$c_n = \frac{d_{n+1} - d_n}{h_n} - \frac{1}{3}(2b_n + b_{n+1})h_n \tag{20}$$

which gives

$$(2b_{n+1} + b_n)h_n = -\frac{3(d_{n+1} - d_n}{h_n} + 3q \tag{21}$$

Together, Equation (15), Equation (17), and Equation (21) yield a system of $n + 1$ equations in $n + 1$ unknowns.

Cubic Spline Interpolation

Clamped Boundary Conditions

In summary:

1. $d_i = x_i, \quad i = 1, \cdots n$
2. $(2b_1 + b_2)h_1 = \frac{3(d_2 - d_1)}{h_1} - 3p$
3. $b_{i-1}h_{i-1} + 2b_i(h_i + h_{i-1}) + b_{i+1}h_i = 3\left(\frac{d_{i+1} - d_i}{h_i} - \frac{d_i - d_{i-1}}{h_{i-1}}\right), \quad i = 2, \cdots, n$
4. $(2b_{n+1} + b_n)h_n = -\frac{d_{n+1} - d_n}{h_n} + 3q$

2 and 4 are single equation each, while 3 generates $n - 1$ equations.

Once b_i's are known, we can find c_i's using Equation (12), and then using Equation (10), we can find a_i's.

Cubic Spline Interpolation

In Matrix Notation:

$$B\mathbf{b} = \mathbf{y}$$

$$\begin{bmatrix} 2h_1 & h_1 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & h_n & 2h_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n+1} \end{bmatrix} = 3 \begin{bmatrix} \frac{d_2-d_1}{h_1} - p \\ \frac{d_3-d_2}{h_2} - \frac{d_2-d_1}{h_1} \\ \frac{d_4-d_3}{h_3} - \frac{d_3-d_2}{h_2} \\ \vdots \\ q - \frac{d_{n+1}-d_n}{h_n} \end{bmatrix} \tag{22}$$

That we can solve in MATLAB to find `b` using `B / y`.

Cubic Spline Interpolation

Free Boundary Conditions

For free boundary conditions, $P_1''(t_1) = 0$, $P_n''(t_{n+1}) = 0$
Knowing,

$$P_1''(t) = 6a_1(t - t_1) + 2b_1 \tag{23}$$

which yields $b_1 = 0$.

And, we get $b_{n+1} = \frac{1}{2}P_n''(t_{n+1})$ which implies $b_{n+1} = 0$.
In summary,

1. $b_1 = 0$
2. $b_{i-1}h_{i-1} + 2b_i(h_i + h_{i-1}) + b_{i+1}h_i = \frac{3(d_{i+1}-d_i)}{h_i} - \frac{3(d_i-d_{i-1})}{h_{i-1}}$, $i = 2, 3, \cdots n$
3. $b_{n+1} = 0$

Once b_i's are known, we can proceed in similar manner as described for finding coefficients for the case of **Clamped Boundation Conditions**.

Cubic Spline Interpolation

In Matrix Notation:

$$B\mathbf{b} = \mathbf{y}$$

$$\begin{bmatrix} 2(h_1+h_2) & h_2 & 0 & \cdots & 0 \\ h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 \\ 0 & h_3 & 2(h_3+h_4) & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{n-1} & 2(h_{n-1}+h_n) \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} \frac{3(d_3-d_2)}{h_2} - \frac{3(d_2-d_1)}{h_1} \\ \frac{3(d_4-d_3)}{h_3} - \frac{3(d_3-d_2)}{h_2} \\ \frac{3(d_5-d_4)}{h_4} - \frac{3(d_4-d_3)}{h_3} \\ \vdots \\ \frac{3(d_{n+1}-d_n)}{h_n} - \frac{3(d_n-d_{n-1})}{h_{n-1}} \end{bmatrix} \tag{24}$$

Note that b_1 and b_{n+1} are given as 0, so they don't need to be solved for.

MATLAB Implementation

Example
Consider $t = [0, 1.01, 2.03, 3.01]$ and $x = [0, 0.5, 2, 1.5]$.

See the code for MATLAB function implementation: https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_cubic_spline_interpolation.m

Demo usage: https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_demo_cubic_spline.m

Reference: Numerical Methods for Engineers and Scientists Using MATLAB, Second Edition, Ramin S. Esfandiari

Differentiation in MATLAB

Differentiation in MATLAB

Differentiation in MATLAB is achieved through discrete derivation

Table: First Derivative Difference Formulas

Difference Formula	First Derivative
Two-point backward	$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$
Two-point forward	$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$
Two-point central	$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$
Three-point backward	$f'(x_i) = \frac{f(x_{i-2}) - 4f(x_{i-1}) + 3f(x_i)}{2h}$
Three-point forward	$f'(x_i) = \frac{-3f(x_i) + 4f(x_{i+1}) - f(x_{i+2})}{2h}$
Four-point central	$f'(x_i) = \frac{f(x_{i-2}) - 8f(x_{i-1}) + 8f(x_{i+1}) - f(x_{i+2})}{12h}$

Here, $x_i = x[i]$.

Differentiation in MATLAB

Table: Second Derivative Difference Formulas

Difference Formula	Second Derivative
Three-point backward	$f''(x_i) = \frac{f(x_{i-2}) - 2f(x_{i-1}) + f(x_i)}{h^2}$
Three-point forward	$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$
Three-point central	$f''(x_i) = \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2}$
Four-point backward	$f''(x_i) = \frac{-f(x_{i-3}) + 4f(x_{i-2}) - 5f(x_{i-1}) + 2f(x_i)}{h^2}$
Four-point forward	$f''(x_i) = \frac{2f(x_i) - 5f(x_{i+1}) + 4f(x_{i+2}) - f(x_{i+3})}{h^2}$
Five-point central	$f''(x_i) = \frac{-f(x_{i-2}) + 16f(x_{i-1}) - 30f(x_i) + 16f(x_{i+1}) - f(x_{i+2})}{12h^2}$

Differentiation in MATLAB: Example

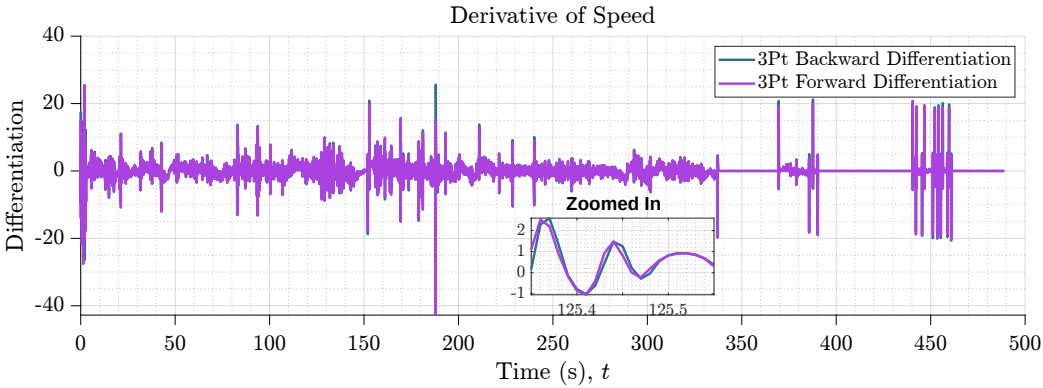
$$y(t) = \cos\left(t^2\right)$$

Differentiation in MATLAB: Example

```
%% Solving derivatives symbolically
% y(t) = cos(t^2)
% dy/dt = -2*t*sin(t^2)
%% Symbolic derivative: the ground truth
syms t y z % we define symbols
y = cos(t^2);
z = diff(y);
figure(1);
subplot(2, 1, 1)
% symbolic plotting
fplot(y, [0, 2*pi], 'LineWidth', 3);
grid on;
hold on;
subplot(2, 1, 2);
fplot(z, [0, 2*pi], 'LineWidth', 3);
grid on;
hold on;
```

```
%% Numerical derivative
Ts = 0.1;
t1 = 0:Ts:2*pi;
y1 = cos(t1.^2);
z1 = diff(y1)./diff(t1);
figure(1);
subplot(2, 1, 1);
stem(t1, y1, 'r');
subplot(2, 1, 2);
stem(t1(1:length(y1) -1), z1, 'm' );
```

Differentiation of a Real Signal



Code:

https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_Differentiation_Real_Signal.m

Integration in MATLAB

Integration Example

$$\int \frac{x^3}{x+2}dx$$

$$\begin{aligned}
\int \frac{x^3}{x+2}dx &= \int \frac{x^3+8-8}{x+2}dx && \text{Add and subtract 8 to the numerator} \\
&= \int \frac{(x+2)(x^2-2x+4)-8}{x+2}dx && \text{Factor the numerator} \\
&= \int \left(x^2-2x+4-\frac{8}{x+2}\right)dx && \text{Split the fraction} \quad (25) \\
&= \int x^2dx - 2\int xdx + 4\int dx - 8\int \frac{1}{x+2}dx && \text{Split the integral into separate terms} \\
&= \frac{x^3}{3} - x^2 + 4x - 8\ln|x+2| + C && \text{Evaluate each integral}
\end{aligned}$$

Integration in MATLAB

```
%% Symbolic Integration
% x^3/(x + 2) dx
syms x
f = x^3/(x + 2)
% indefinite integral
int(f)
% definite integral
int(f, 0, 10)
```

```
%% Numerical Integration using trapezoidal rule,
%% Numerical integration is always definite
x = 0:0.1:10;
f = x.^3./(x + 2);
trapz(x, f)
```

Solving Differential Equation in MATLAB

Differential Equation: Initial Value Problems

- An nth-order differential equation, requires n conditions
- When these conditions are provided at the same initial value of the independent variable, we have an **initial-value problem (IVP)**.

A single, first-order initial-value problem is formulated as

$$\frac{dx(t)}{dt} = x(t), x(0) = x_0, 0 \leq t \leq t_n \tag{26}$$

Solving Ordinary Differential Equations using Numerical Methods

Recognizing that derivatives can be approximated as difference equations in the discrete domain, we have some methods to solve ordinary differential equations, suitable for computer implementations.
Some common numerical methods to solve ODE are:

- Euler's Method
- Runge-Kutta Method (4th Order)

Euler's Method

- Consider the ODE: $\frac{dy}{dt} = f(t, y)$
- Initial condition: $y(t_0) = y_0$
- Euler's method formula: $y_{n+1} = y_n + hf(t_n, y_n)$
- Example: $\frac{dy}{dt} = -2y, \quad y(0) = 1$
- Using $h = 0.1$: $y_{n+1} = y_n - 0.2y_n = y_n(1 - 0.2)$

Euler's Method

Euler's method:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

Example: Solve $y' = 2t$ with $y(0) = 1$ using Euler's method with $h = 0.1$

- In this case, $f(t, y) = 2t$

t_n	y_n	y_{n+1}
0	1	1.0
0.1	1.0	1.02
0.2	1.02	1.06
⋮	⋮	⋮

Runge-Kutta Method (4th Order)

- Consider the ODE: $\frac{dy}{dt} = f(t, y)$
- Initial condition: $y(t_0) = y_0$
- Runge-Kutta method formula:

$$k_1 = hf(t_n, y_n)$$
$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right)$$
$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right)$$
$$k_4 = hf(t_n + h, y_n + hk_3)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- Example: $\frac{dy}{dt} = t + y, \quad y(0) = 1$
- Using $h = 0.1$:
  Write a MATLAB code to implement that.

Runge-Kutta Methods

Runge-Kutta methods:

- More accurate than Euler's method
- Use multiple function evaluations at each step

Example: Solve $y' = 2x$ with $y(0) = 1$ using the Runge-Kutta method of order 4 with $h = 0.1$

x_n	y_n	y_{n+1}
$\vdots$	$\vdots$	$\vdots$

Ordinary Differential Equation in MATLAB

Solve the initial value problem $ty' + 3y = 0, y(1) = 2$, assuming $t > 0$. We write the equation in standard form: $y' + 3y/t = 0$.

$$P(t) = \int -\frac{3}{t} dt = -3 \ln t$$

and

$$y = Ae^{-3 \ln t} = At^{-3}$$

Substitute to find A: $2 = A(1)^{-3}$, so the solution is $y = 2t^{-3}$.

Ordinary Differential Equation in MATLAB

```
%% ty; + 3y = 0; y(1) = 2
% solution y = 2/t^3
syms y(t)
eqn = diff(y, t) + 3*y/t ==0;
cond = y(1) == 2;
y(t) = dsolve(eqn, cond);
```

Euler's Method Implementation

Consider t_0 be the initial time, h be the sampling time. Then $t_1 = t_0 + h$.

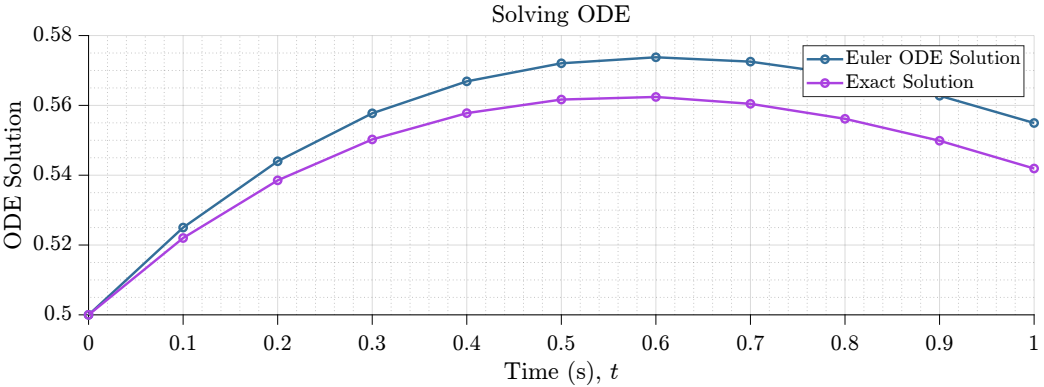
Using Taylor's Series:

$$x(t_1) = x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{1}{2!}h^2x''(t_0) + \cdots + \tag{27}$$

$$x(t_1) \approx x(t_0) + hx'(t_0)$$

Example and MATLAB Implementation of Euler ODE

Consider $2x' + x(t) = e^{-t}, x(0) = 1/2, 0 \leq t \leq 1$



Code:
https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_EulerODE.m
https://github.com/rahulbhadani/CPE381_FA25/blob/main/Code/Ch02_demo_EulerODE.m

Up Next

- Operations on Continuous-time Signals