SECTION 9: ENGINEERING APPLICATIONS

ENGR 112 – Introduction to Engineering Computing

Systems of Equations

Systems of Equations

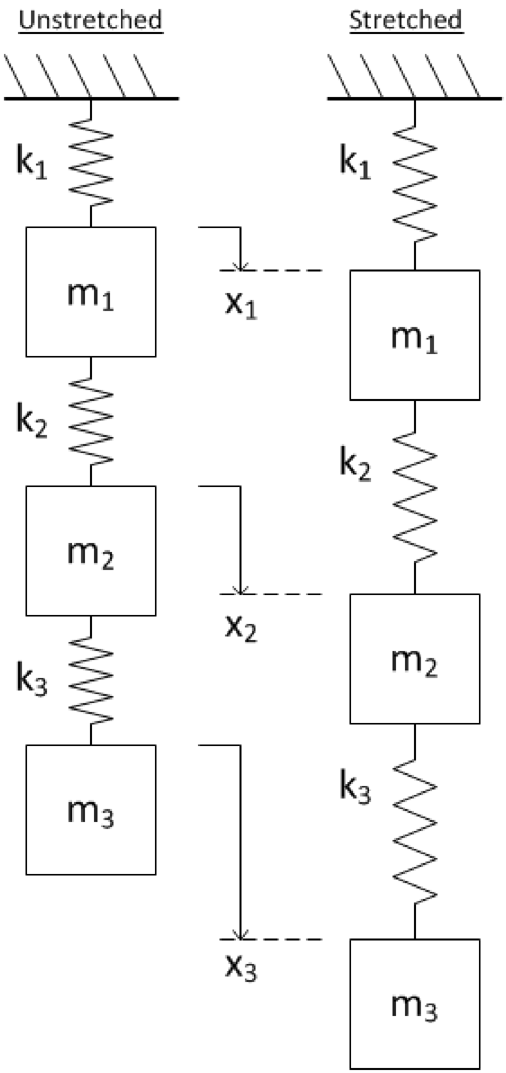
- Systems of equations common in all engineering disciplines
- For N unknown variables, we need a system of N equations
  - Can represent in matrix form:

$$\mathbf{Ax} = \mathbf{b}$$

    - A: $N \times N$ matrix of known, constant coefficients
    - x: $N \times 1$ vector of unknowns
    - b: $N \times 1$ vector of known constants

- Many tools exist for solving:
  - By hand – substitution, Gaussian elimination, etc.
  - Scientific calculators
  - Here, we will look at the tools available within MATLAB

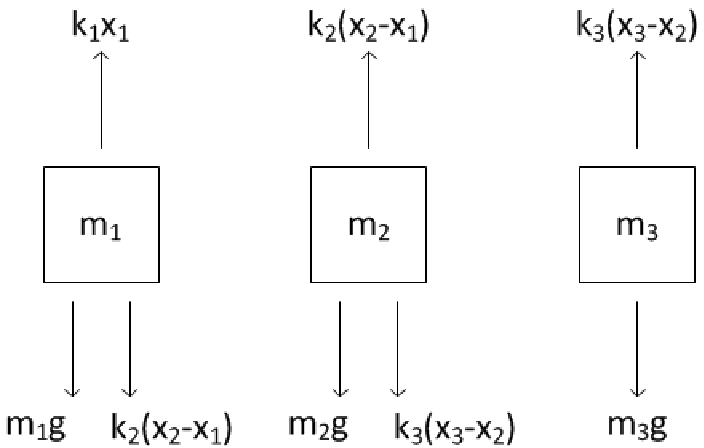
A System of Equations – Example



- Consider the following scenario
- Three masses
  - m_1, m_2, and m_3
- Three springs
  - k_1, k_2, k_3
- Connected in series and suspended
- Determine the displacement of each mass from its unstretched position

A System of Equations – Example

- Three unknown displacements: x_1, x_2, x_3
  - Need three equations to find displacements
- Apply Newton's second law to each mass



- Three equations result:

$$m_1\ddot{x}_1 = m_1 g + k_2(x_2 - x_1) - k_1 x_1$$
$$m_2\ddot{x}_2 = m_2 g + k_3(x_3 - x_2) - k_2(x_2 - x_1)$$
$$m_3\ddot{x}_3 = m_3 g - k_3(x_3 - x_2)$$

A System of Equations – Example

- Steady-state, so no acceleration: $\ddot{x}_i = 0, \;\forall i$

$$m_1 g + k_2(x_2 - x_1) - k_1 x_1 = 0$$

$$m_2 g + k_3(x_3 - x_2) - k_2(x_2 - x_1) = 0$$

$$m_3 g - k_3(x_3 - x_2) = 0$$

- Rearranging

$$\begin{aligned}
(k_1 + k_2)x_1 \quad & - k_2 x_2 & + 0x_3 &= m_1 g \\
-k_2 x_1 \quad & + (k_2 + k_3)x_2 & - k_3 x_3 &= m_2 g \\
0x_1 \quad & - k_3 x_2 & + k_3 x_3 &= m_3 g
\end{aligned}$$

A System of Equations – Example

- Our system of three equations

$$\begin{aligned} (k_1 + k_2)x_1 \quad & - k_2 x_2 \quad + 0x_3 = m_1 g \\ -k_2 x_1 \quad + (k_2 + k_3)x_2 \quad & - k_3 x_3 = m_2 g \\ 0x_1 \quad & - k_3 x_2 \quad + k_3 x_3 = m_3 g \end{aligned}$$

can be put into matrix form

$$\begin{bmatrix} (k_1 + k_2) & -k_2 & 0 \\ -k_2 & (k_2 + k_3) & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} m_1 g \\ m_2 g \\ m_3 g \end{bmatrix}$$

A System of Equations – Example

$$\begin{bmatrix} (k_1 + k_2) & -k_2 & 0 \\ -k_2 & (k_2 + k_3) & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} m_1 g \\ m_2 g \\ m_3 g \end{bmatrix}$$

- We can rewrite this matrix equation as

$$\mathbf{Ax} = \mathbf{b}$$

- Can apply tools of linear algebra to determine the vector of unknown displacements

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Solution Using Matrix Inverse

- We have a system of equations:

$$\mathbf{Ax} = \mathbf{b}$$

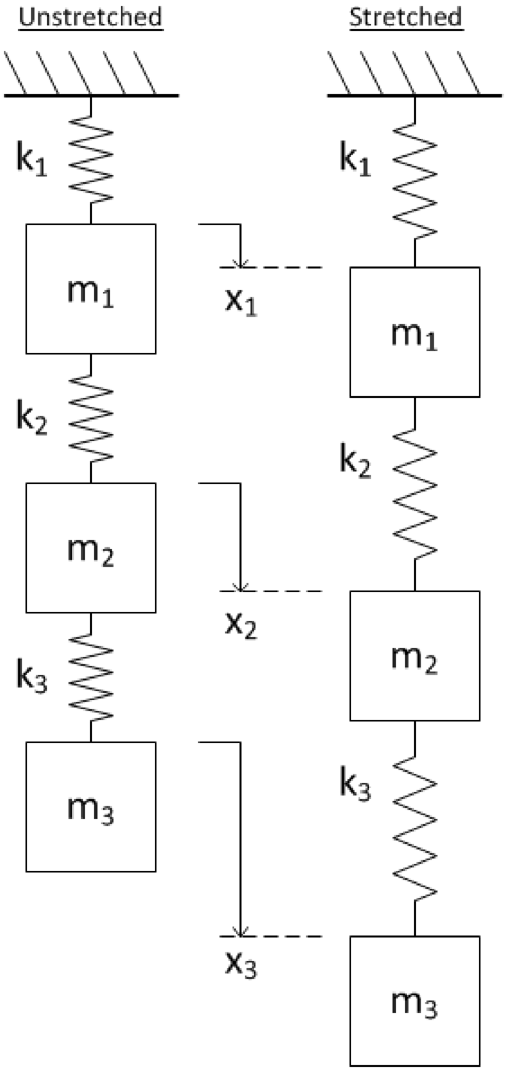
- If a solution exists, then the coefficient matrix, $\mathbf{A}$, is invertible
  - Not always the case
- Left-multiply by $\mathbf{A^{-1}}$ to solve for the vector of unknowns, x

$$\mathbf{A^{-1}Ax} = \mathbf{A^{-1}b}$$

$$\mathbf{Ix} = \mathbf{A^{-1}b}$$

$$\mathbf{x} = \mathbf{A^{-1}b}$$

Solution Using Matrix Inverse



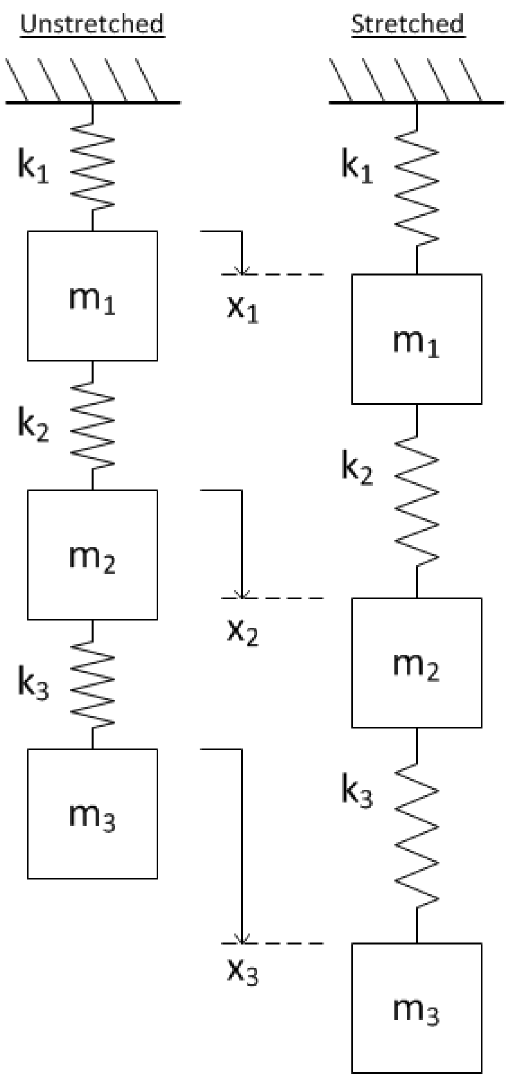
- Our linear system is described by the matrix equation

$$\begin{bmatrix} (k_1+k_2) & -k_2 & 0 \\ -k_2 & (k_2+k_3) & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} m_1 g \\ m_2 g \\ m_3 g \end{bmatrix}$$

$$\mathbf{Ax} = \mathbf{b}$$

- Find the displacements, **x**, for the following system parameters
  - $k_1 = 500\frac{N}{m},\ k_2 = 800\frac{N}{m},\ k_3 = 400\frac{N}{m}$
  - $m_1 = 3kg,\ m_2 = 1kg,\ m_3 = 7kg$

Solution Using Matrix Inverse



Editor - C:\Users\webbky\Box\KWebb\Classes\ENGR112\Notes\MATLAB...

```matlab
% linSysEx.m

clear all; clc

% spring constants   [N/m]
k1 = 500;
k2 = 800;
k3 = 400;

% masses   [kg]
m1 = 3;
m2 = 1;
m3 = 7;

% grav. accel.   [m/s^2]
g = 9.81;

A = [k1+k2,    -k2,    0;...
         -k2, k2+k3, -k3;...
            0,    -k3,   k3];

b = [m1*g; m2*g; m3*g];

x = inv(A)*b

```

Command Window

```
x =

    0.2158
    0.3139
    0.4856

>>
```

$$\boxed{x_1 = 21.6cm, \quad x_2 = 31.4cm, \quad x_3 = 48.6cm}$$

Solution Using `mldivide.m`, `\`

- MATLAB has a second division function
  - ***Matrix left division***: `mldivide.m`, `\`
- Use `mldivide` to solve

$$\mathbf{Ax} = \mathbf{b}$$

- If $\mathbf{A}^{-1}$ exists, then

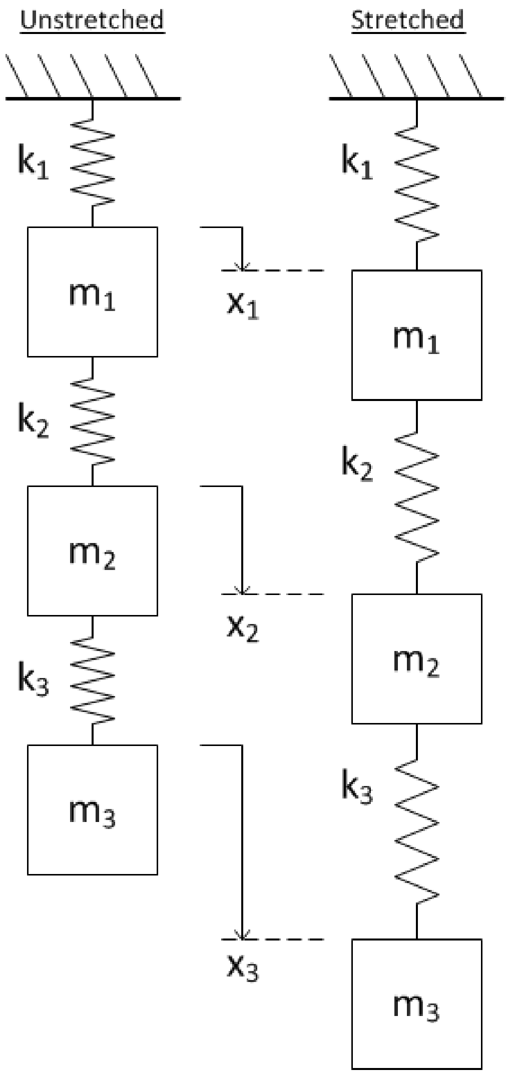
```
x = A\b;
```

is equivalent to

```
x = inv(A)*b;
```

- But, does not calculate $\mathbf{A}^{-1}$
  - Faster and more numerically robust

Solution Using `mldivide.m`, `\`



Editor - C:\Users\webbky\Box\KWebb\Classes\ENGR112\Notes\MATLAB\...

```matlab
% linSysEx.m

clear all; clc

% spring constants  [N/m]
k1 = 500;
k2 = 800;
k3 = 400;

% masses  [kg]
m1 = 3;
m2 = 1;
m3 = 7;

% grav. accel.  [m/s^2]
g = 9.81;

A = [k1+k2,   -k2,    0;...
        -k2, k2+k3, -k3;...
          0,   -k3,  k3];

b = [m1*g; m2*g; m3*g];

x = inv(A)*b

x = A\b
```

Command Window

```
x =

    0.2158
    0.3139
    0.4856


x =

    0.2158
    0.3139
    0.4856

>>
```

$$x_1 = 21.6cm, \quad x_2 = 31.4cm, \quad x_3 = 48.6cm$$

Numerical Differentiation

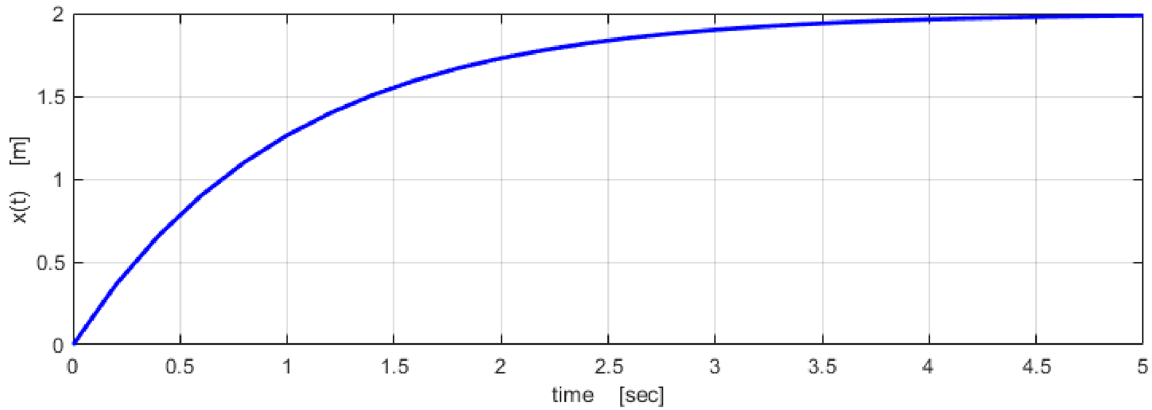
Differentiation

- As engineers, we often deal with ***rates***
  - ***Changes in on quantity with respect to another***
- Often these are rates with respect to time, e.g.:
  - *Velocity*: change in position w.r.t. time
  - *Acceleration*: change in velocity w.r.t. time
  - *Power*: time rate of energy transfer
  - Changes in *voltage* or *current* w.r.t. time
  - Etc.
- Mathematically, these rates are described by ***derivatives***
- Calculation of a derivative is ***differentiation***

Derivatives

- For example, consider an object whose ***position as a function of time*** is

$$x(t) = 2\,m \cdot (1 - e^{-t})$$



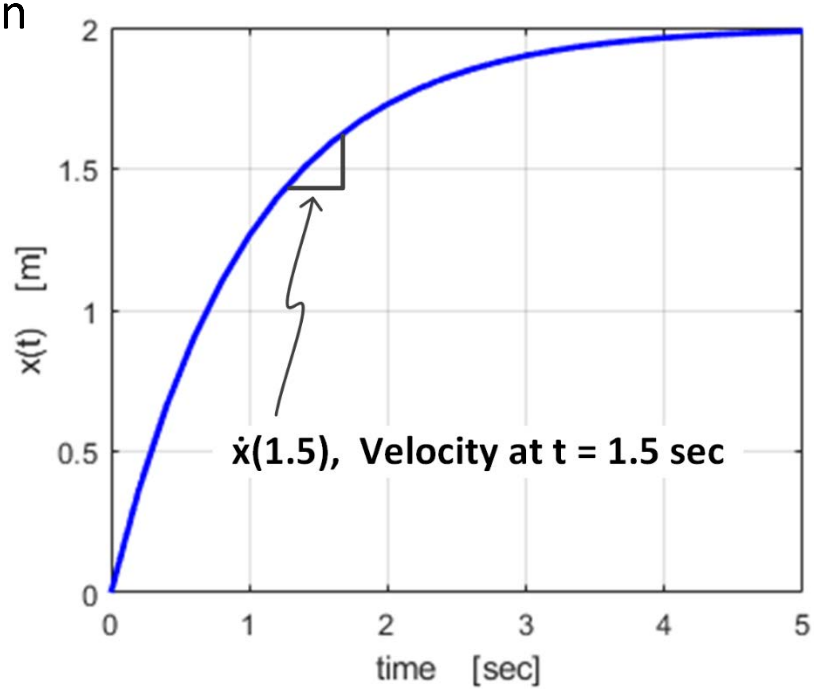
- At any point in time, t, the object's velocity, $v(t)$, is given by the time rate of change of position
  - That is, the ***derivative w.r.t. time*** of position

$$v(t) = \frac{dx}{dt} = \dot{x}(t) = x'(t)$$

Derivatives

- Velocity is the ***rate of change*** of position w.r.t. time
  - ***Slope*** of the position graph
  - The ***derivative*** of position

$$v(t) = \frac{dx}{dt} = \dot{x}(t)$$



- You know/will learn to differentiate mathematical expressions, e.g.

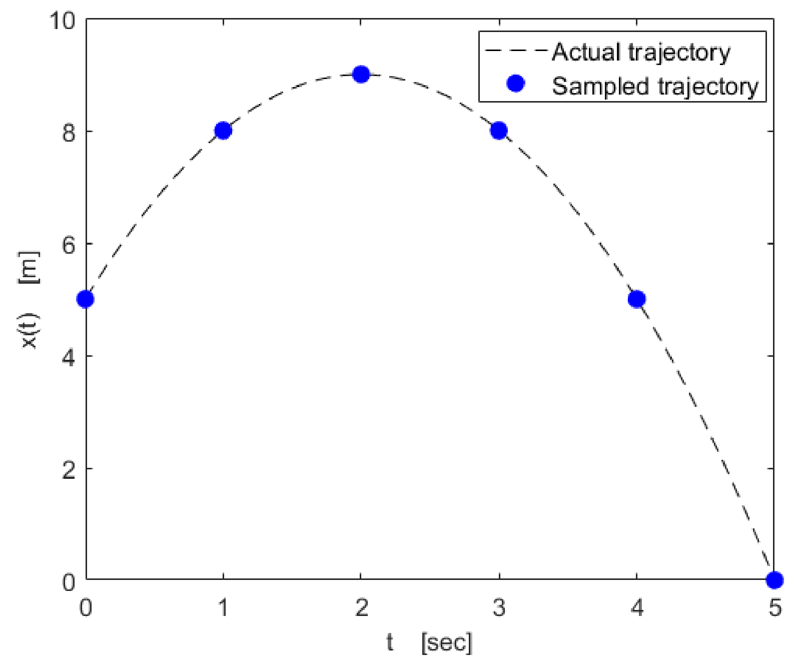
$$x(t) = 2\,m \cdot (1 - e^{-t})$$

$$\dot{x}(t) = v(t) = 2\frac{m}{s} \cdot e^{-t}$$

- Often, we would like to calculate a derivative, but we do not have a mathematical expression, e.g.
  - Measurement data
  - Simulation data, etc.
- Then, we can ***approximate*** the derivative ***numerically***

Numerical Differentiation

- Data we want to differentiate are ***discrete***
  - ***Sampled*** – not continuous
  - Data only exist at ***discrete*** points in time
  - Result of simulation or measurement, etc.

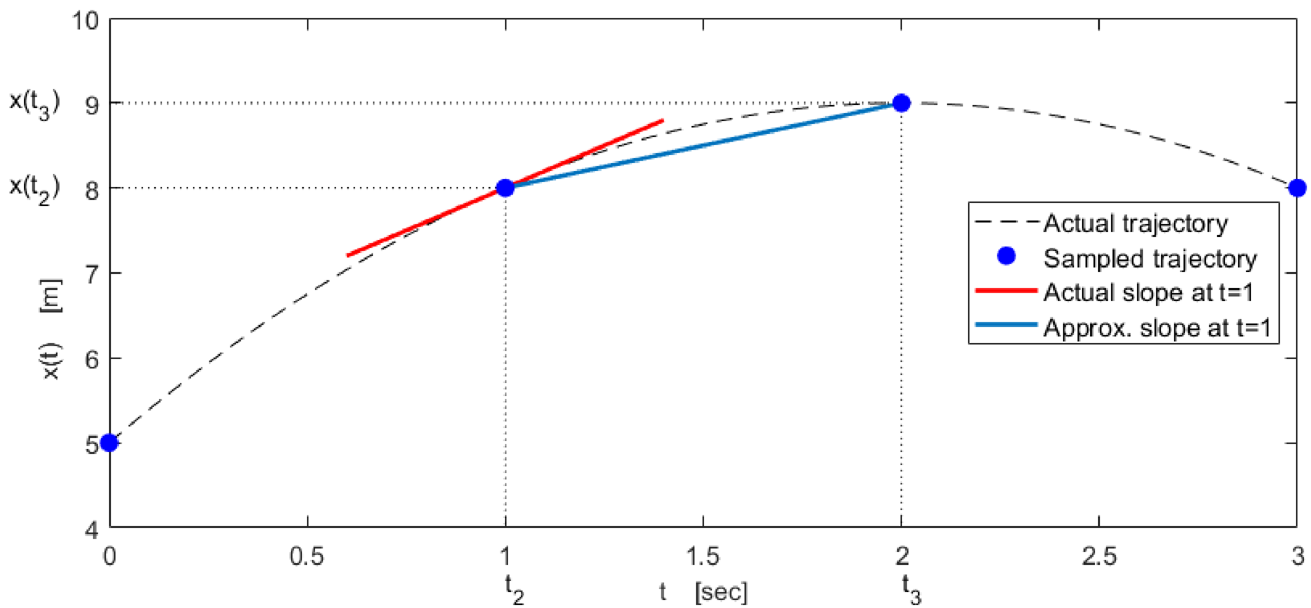


- ***Numerical differentiation***
  - Approximation of the slope at each discrete data point
- Several methods exist for numerical differentiation
  - Varying complexity and accuracy
- Here, we'll focus on the ***forward difference method***

Forward Difference Method

- ***Forward difference method***
  - Approximate $\dot{x}(t_i)$ using $x(t_i)$ and $x(t_{i+1})$
    - Data at the current time point and one time step ***forward***

$$\boxed{\dot{x}(t_i) \approx \frac{x(t_{i+1}) - x(t_i)}{t_{i+1} - t_i} = \frac{\Delta x}{\Delta t}}$$



Forward Difference in MATLAB

- Numerical differentiation in MATLAB

$$\dot{x}(t_i) \approx \frac{x(t_{i+1}) - x(t_i)}{t_{i+1} - t_i} = \frac{\Delta x}{\Delta t}$$

- We would have:
  - Time vector, t
    - Possibly, but not necessarily evenly spaced
  - Data vector, $x(t)$
    - Function to be differentiated

- Use `diff.m` to calculate Δx and Δt vectors
- Use array division, `./`, to calculate $\Delta x/\Delta t$ at each time point
  - No $\Delta x/\Delta t$ value at the last time point

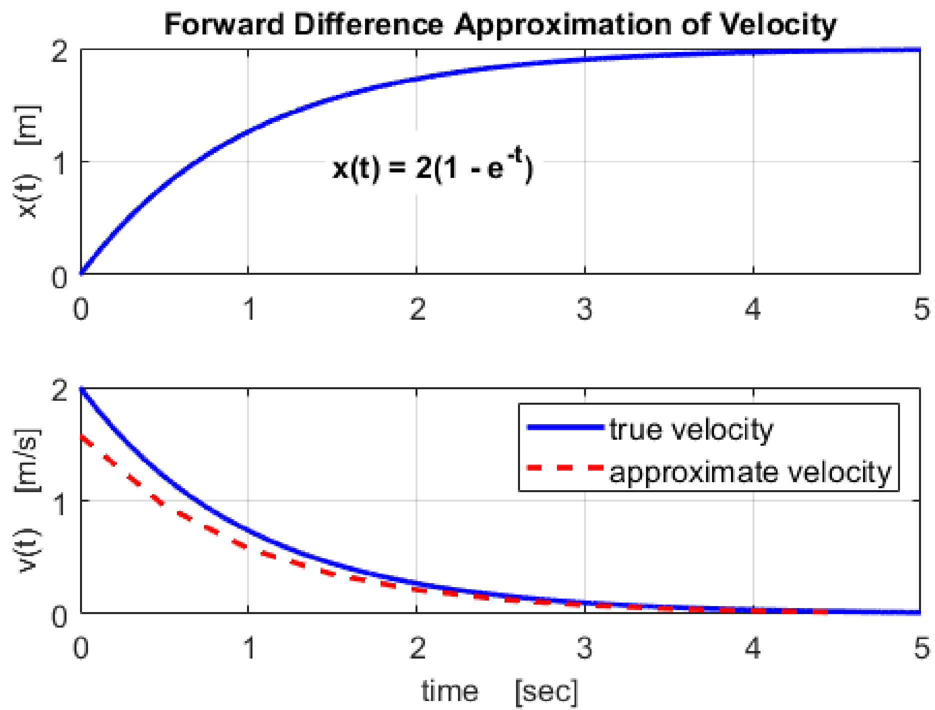
Numerical Differentiation – Example

- Consider again an object whose position is given by:

$$x(t) = 2\,m \cdot (1 - e^{-t})$$

- Use forward difference to approximate velocity
  - Assume a 200 msec sample period
- Error would improve with smaller time steps

```
% sampled function
ts = 0:500e-3:5;
xs = 2*(1-exp(-ts));

dt = diff(ts);              % time differences
dx = diff(xs);              % position differences

dxdt = dx./dt;              % approximate derivative
```



Numerical Integration

Integration

$$\int_a^b f(t)dt$$

- ***Integration*** is a mathematical operation involving the calculation of a ***continuous sum*** over some interval
  - The inverse of differentiation – the antiderivative

$$\int f'(t)dt = f(t)$$

- We have seen that the derivative represents the rate of change of a function w.r.t. its independent variable
  - For example, consider the position of an object, $x(t)$
  - Velocity of the object is the derivative of position

$$v(t) = \frac{dx}{dt} = x'(t)$$

  - The rate of change of position w.r.t. time

Integration

- ***Integration is the inverse of differentiation***
  - Mathematical transform between a rate of a quantity (e.g., $v(t) = x'(t)$) and that quantity (e.g., $x(t)$)

$$x(t) = \int v(t)\,dt = \int x'(t)\,dt$$

- Examples of integral/derivative relationships:

	Integral → / ← Derivative	
Velocity		Position
Acceleration		Velocity
Power		Energy
Current		Electrical charge

Integration

- In your calculus class you learned/will learn to calculate the integral of functions, e.g.,

$$\int_0^1 e^{-\frac{t}{2}}\,dt = -2\cdot e^{-\frac{t}{2}}\Big|_0^1$$

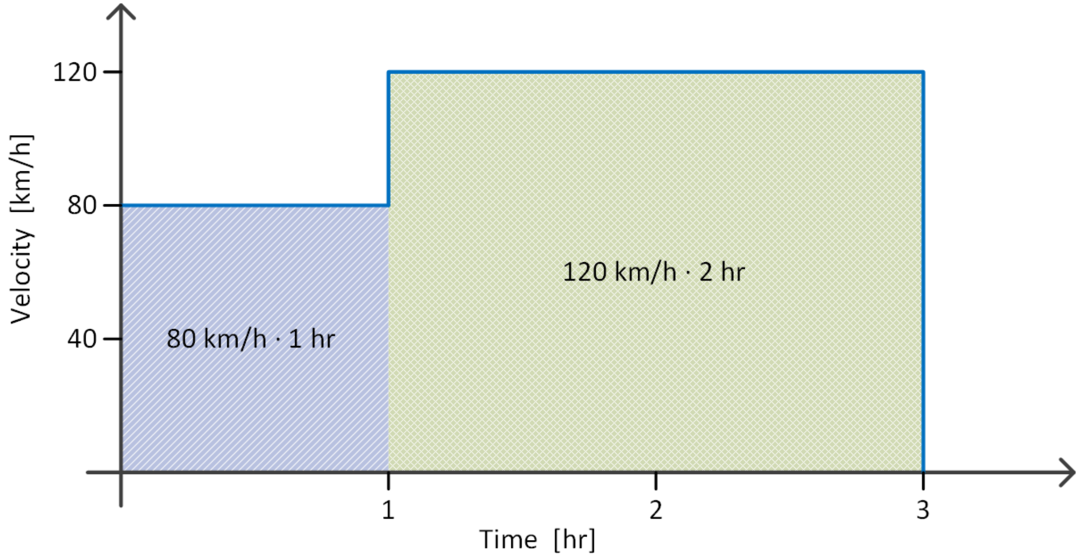
$$= -2(0.6065 - 1)$$

$$\int_0^1 e^{-\frac{t}{2}}\,dt = 0.787$$

- As was the case for differentiation, we often do not have a mathematical expression for the data we want to integrate
  - E.g., measurement data or simulation data
  - Only have discrete data points
  - Integrate ***numerically***

Numerical Integration

- The ***derivative*** of a function is the ***slope of its graph***
- The ***integral*** of a function is the ***area under its graph***
- For example, distance traveled is the integral of velocity
  - Consider a car that travels at a speed of 80 km/h for 1 hour and 120 km/h for 2 hours
    - How far has the car traveled after three hours?



Numerical Integration

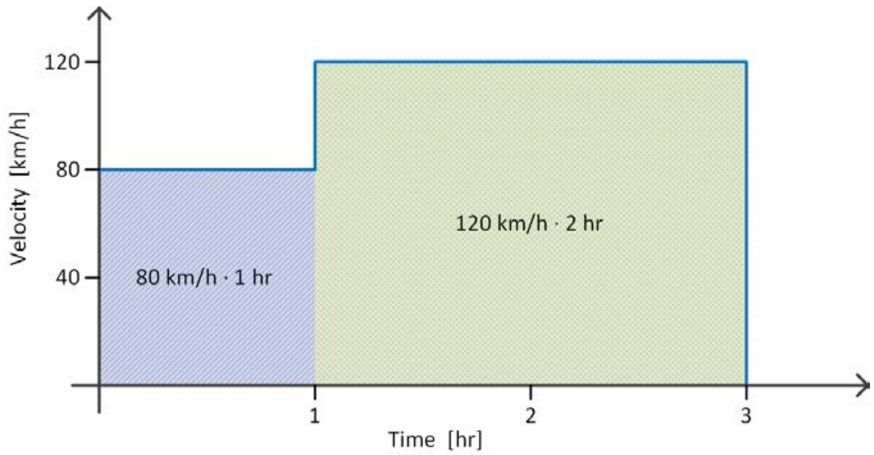
- Distance at $t = 3\ hr$:

$$x(3) = \int_0^3 v(t)\,dt$$

$$x(3) = \int_0^1 v(t)\,dt + \int_1^3 v(t)\,dt$$

$$x(3) = 80\frac{km}{h} \cdot 1\ hr + 120\frac{km}{h} \cdot 2\ hr$$

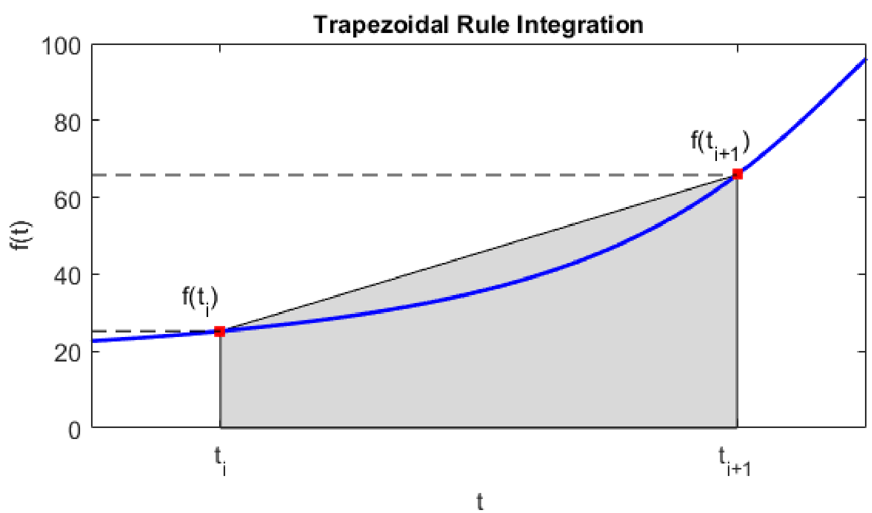
$$x(3) = 320\ km$$



- ***Numerical integration***
  - Numerical approximation of area under a curve defined by a function or a discrete data set
  - We will focus on one simple method: the ***trapezoidal rule***

Trapezoidal Rule Integration

- Approximate the integral between adjacent time point:
  - Approximate area under the curve between those time points
    - Area of a trapezoid

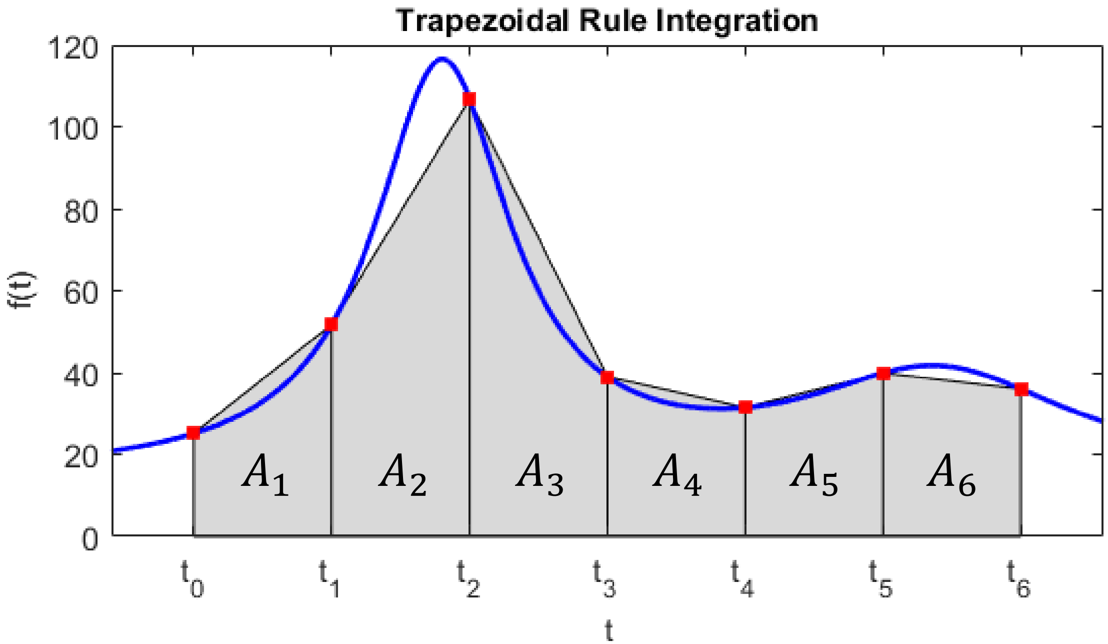


$$Area \approx \frac{f(t_i) + f(t_{i+1})}{2} \cdot (t_{i+1} - t_i)$$

$$Area \approx (Avg.height) \cdot (width)$$

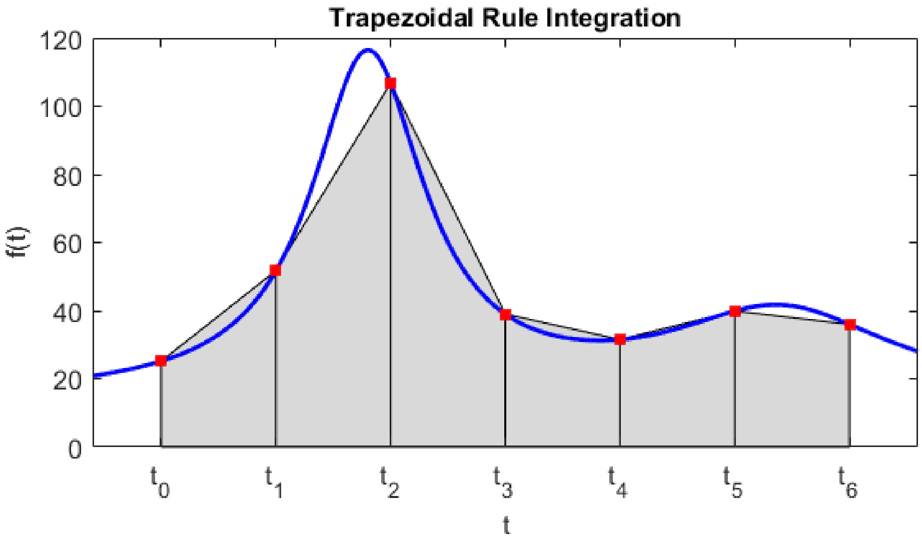
Trapezoidal Rule Integration

- Overall integral approximated by the approximate total area
  - Sum of all individual trapezoidal segment areas



$$\int_{t_0}^{t_6} f(t)\,dt \approx \sum_{i=1}^{6} A_i = A_1 + A_2 + A_3 + A_4 + A_5 + A_6$$

Trapezoidal Rule Integration



$$\int_{t_0}^{t_6} f(t)\,dt \approx \sum_{i=0}^{5} \frac{f(t_i)+f(t_{i+1})}{2}\cdot(t_{i+1}-t_i)$$

$$\int_{t_0}^{t_6} f(t)\,dt \approx \left[\frac{f(t_0)+f(t_1)}{2}\cdot(t_1-t_0)\right]+\left[\frac{f(t_1)+f(t_2)}{2}\cdot(t_2-t_1)\right]+\cdots$$

$$\cdots+\left[\frac{f(t_5)+f(t_6)}{2}\cdot(t_6-t_5)\right]$$

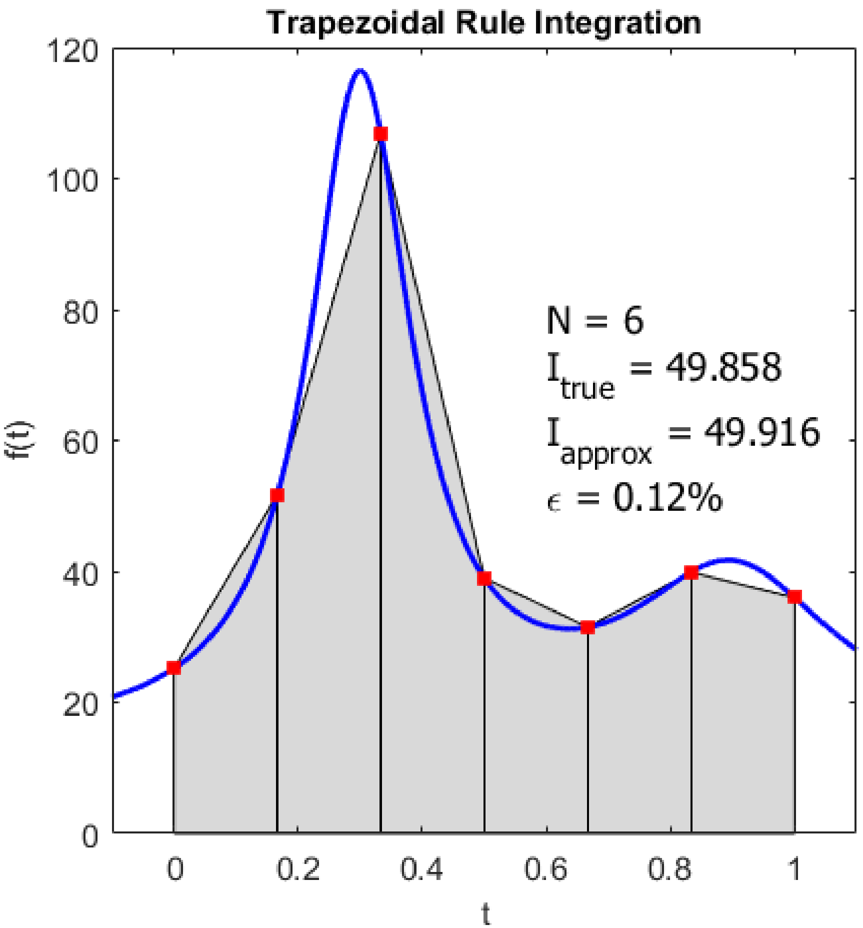
Trapezoidal Rule in MATLAB – `trapz.m`

$$\boxed{\texttt{I = trapz(x,y)}}$$

- `x`: vector of independent variable data
- `y`: vector of dependent variable data
- `I`: trapezoidal rule approximation to the integral of `y` with respect to `x` (a scalar)

- Data need not be equally-spaced
  - Segment widths calculated from `x` values

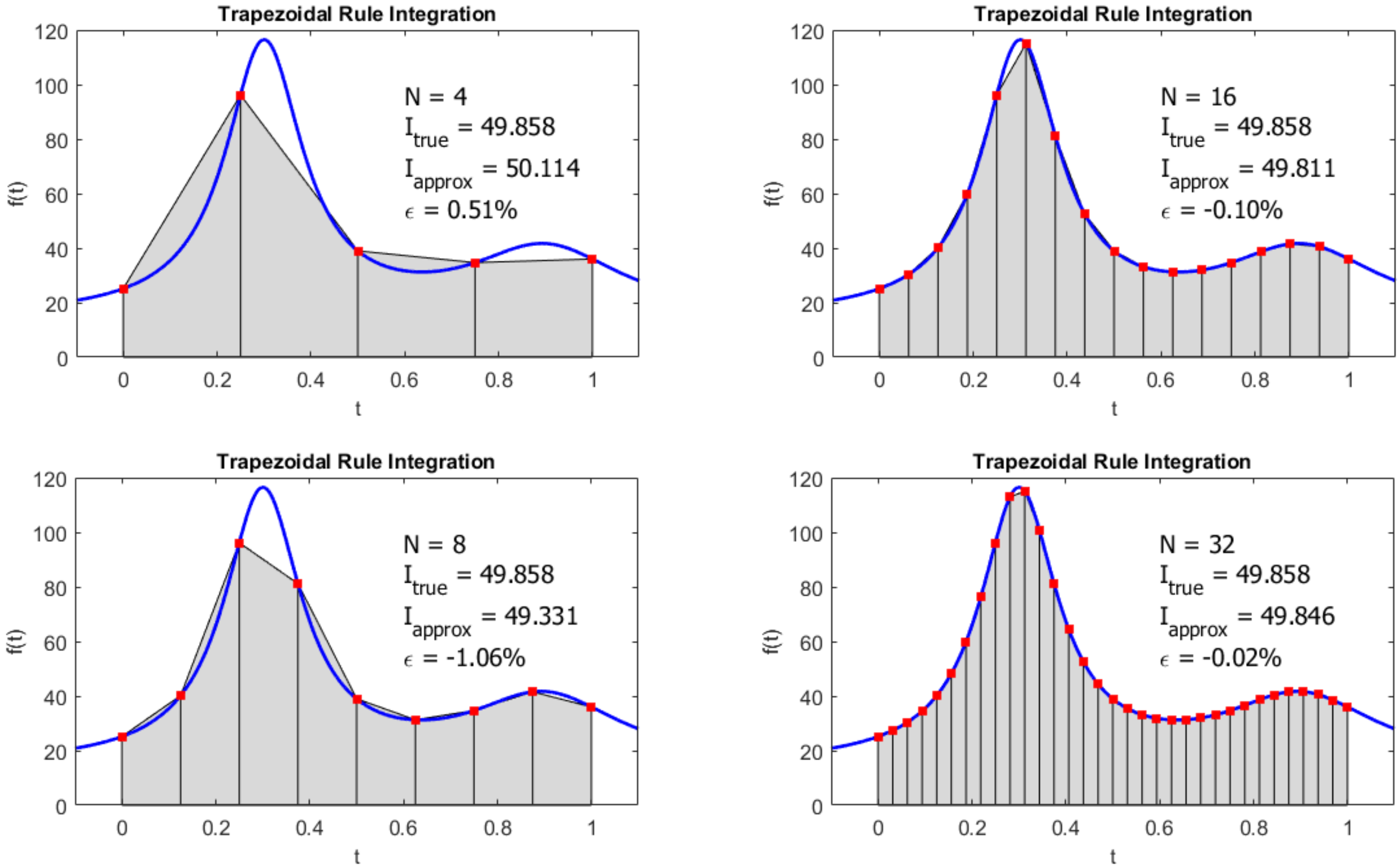
Trapezoidal Rule – Example




```
% the function to be integrated (MATLAB's humps.m  + 20)
% in practice, we would generally not have this
f = @(t) 1./((t-.3).^2 + .01) + 1./((t-.9).^2 + .04) + 14;

% expression of true integral
% in practice, we would generally not have this either
intf = @(t) 14*t + 10*atan(10*t - 3) + 5*atan(5*t - 9/2);

% evaluate f(t) over [a,b] with N segments, N+1 samples
a = 0;
b = 1;
N = 6;
t = linspace(a,b,N+1);

y = f(t);        % data to be integrated

% approx. the integral over [a,b] using trapz.m
Ihat = trapz(t,y);

% the value of the true integral over [a,b]
I = intf(b) - intf(a);

% percent error of the numerical approximation
err = (Ihat - I)/I * 100;
```

Trapezoidal Rule – Example

- Error decreases as
  - Number of segments (sampling frequency) increases
  - Segment size (sampling period) decreases



Indefinite Integrals

- Sometimes, we want to know the result of an integral from a to b
  - A ***definite integral***
  - A number
  - E.g., given velocity $v(t)$, find the total distance traveled

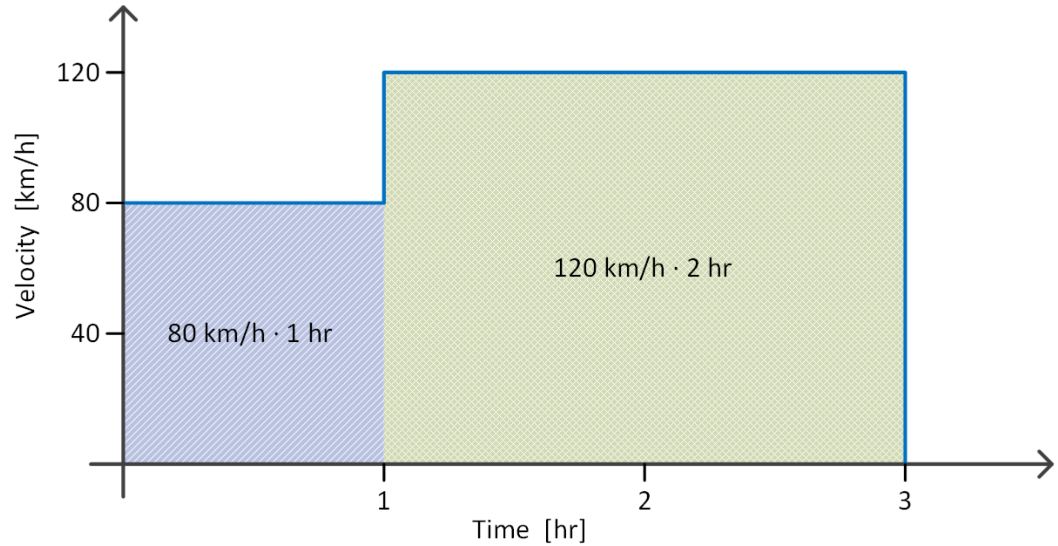
$$\Delta x = x(b) - x(a) = \int_a^b v(t)\,dt$$

- Other times, we would like the result of an integral as a function of time
  - An ***indefinite integral*** or a ***cumulative integral***
  - E.g., given $v(t)$, find the distance traveled as a function of time

$$x(t) = \int_0^t v(\tau)\,d\tau$$

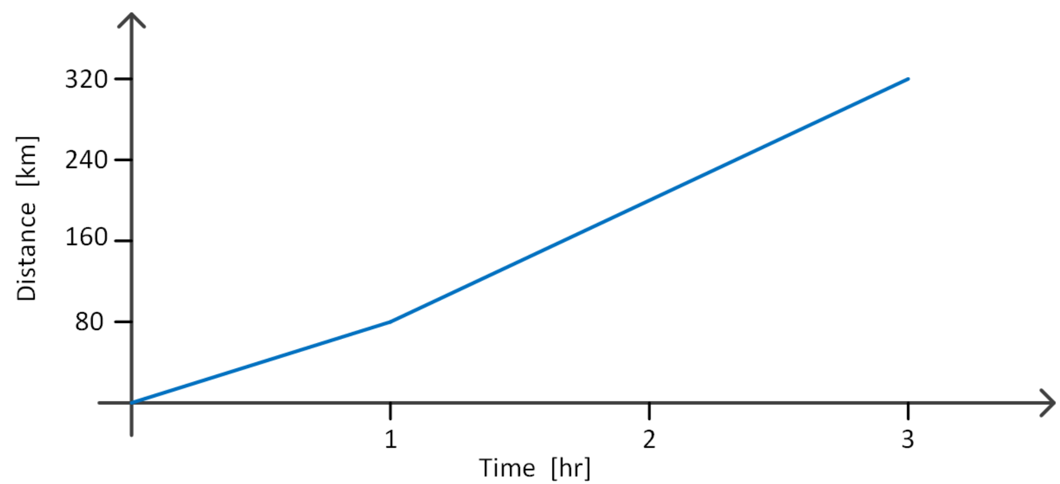
Indefinite Integrals

- Velocity, $v(t)$:



- Integrate velocity to get distance as a function of time:

$$x(t) = \int v(t)\, dt$$



Trapezoidal Rule in MATLAB – `cumtrapz.m`


```
I = cumtrapz(x,y)
```

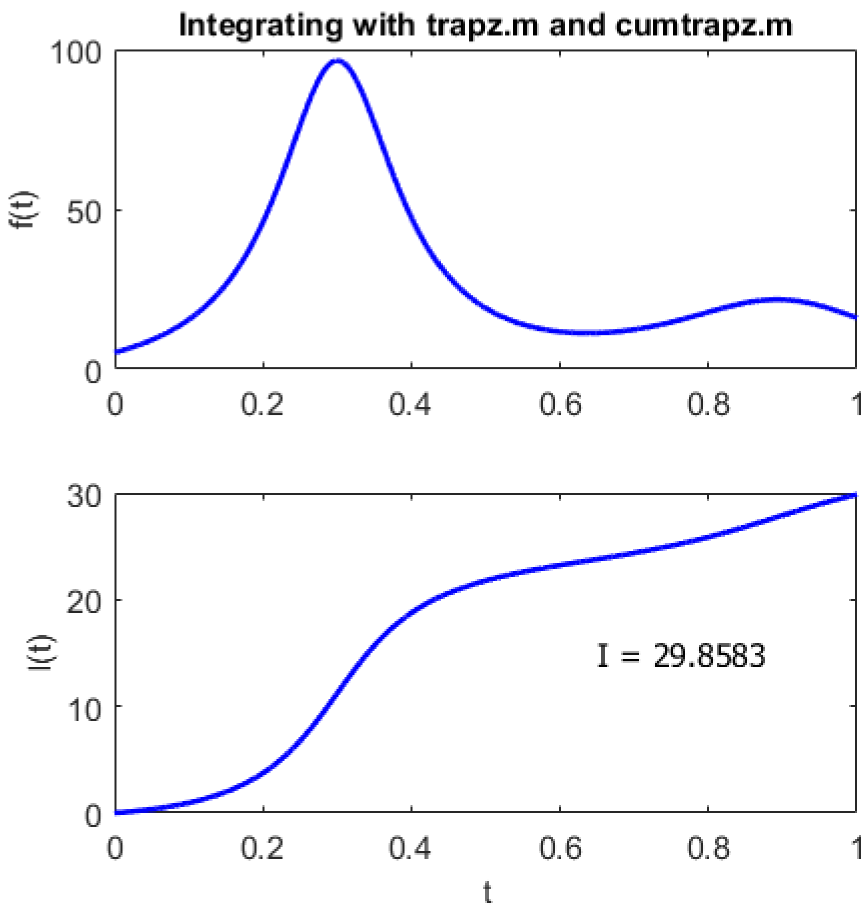
- `x`: n-vector of independent variable data
- `y`: n-vector of dependent variable data
- `I`: trapezoidal rule approximation to the ***cumulative integral*** of `y` with respect to `x` (an n-vector)

- Result is a vector – equivalent to:

$$I(x) = \int_{x_1}^{x} y(\tilde{x})\, d\tilde{x}$$

- Data need not be equally-spaced

trapz.m and cumtrapz.m




```
% trapz_test.m

clear all; clc

% create the data to be integrated
x = linspace(0,1,2000);
y = humps(x);

% definite integral
I = trapz(x,y);

% cumulative or indefinite integral
Ic = cumtrapz(x,y);

figure(1); clf
subplot(211)
plot(x,y,'-b','LineWidth',2);
ylabel('f(x)')
title('Integrating with trapz.m and cumtrapz.m',...
    'FontWeight','Bold')

subplot(212)
plot(x,Ic,'-b','LineWidth',2);
xlabel('x'); ylabel('I(x)')
text(0.65,15,['I = ',num2str(I,'%1.4f')],...
    'FontSize',12,'FontName','Tahoma')
```

Integrating Functions – `integral.m`

- If we do have an expression for the function to be integrated, we can use MATLAB's `integral.m` function:

```
I = integral(f,a,b)
```

  - `f`: handle to the function to be integrated
  - `a`: lower integration limit
  - `b`: upper integration limit
  - `I`: numerical approximation of the integral

- Calculates $I = \int_a^b f(x)dx$

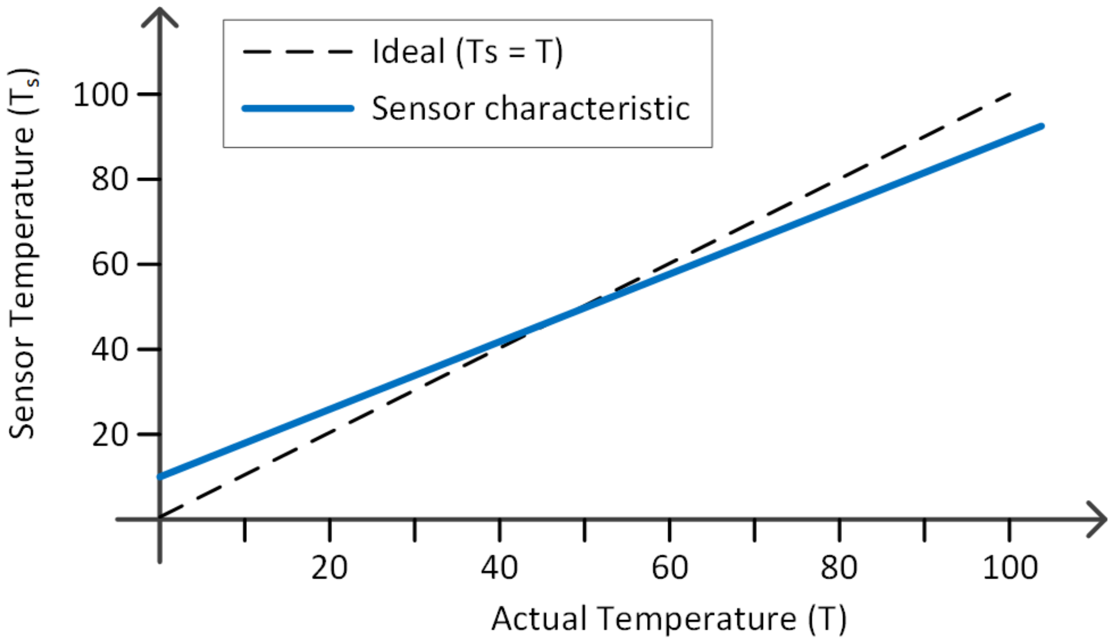
Curve Fitting

Curve Fitting

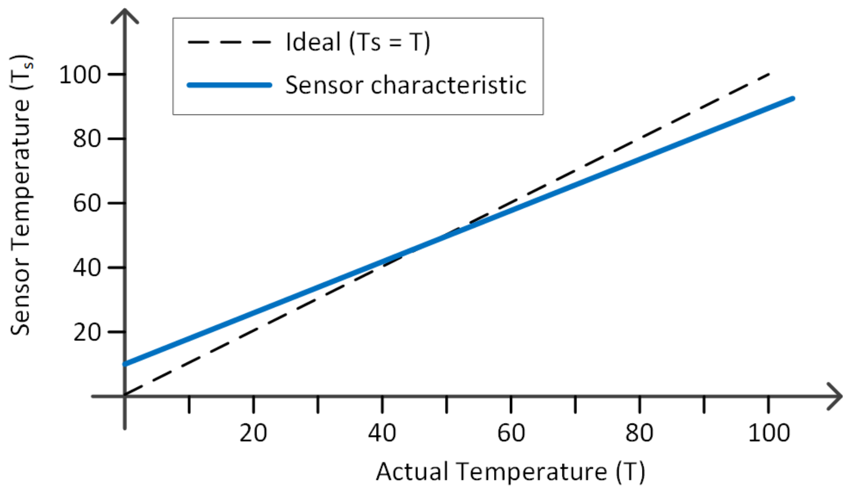
- Engineers often deal with discrete data sets, e.g.
  - E.g., measurement or simulation data
- Typically, that data is noisy
  - Measurement noise
  - Random variations, external disturbances, etc.
- Typically don't have a mathematical expression for the data
  - But, we may want one
  - Sometimes, we may know the data should follow a certain type of function
    - E.g., linear, quadratic, exponential, etc.
- We can ***fit a curve to the data***
  - Determine function parameters that best fit the data
    - E.g., slope and intercept values for a linear relationship
  - Or, determine what type of function provides the best fit
    - E.g., linear, quadratic, exponential, etc.

Curve Fitting

- Consider the following engineering example:
- An inexpensive temperature sensor is to be used to measure ambient temperature
  - Temperature measured and recorded by a micro-controller
  - Low accuracy (inexpensive)
- Sensor output compared to actual temperature may look like:



Curve Fitting



- Ideally, the sensor temperature, T_s, would equal the true temperature, T:

$$T_s = T$$

- But, due to inaccuracy:

$$T_s = a_1 \cdot T + a_0$$

  - a_1: proportional error
  - a_0: offset error

Curve Fitting

- To achieve accurate measurements, we could ***calibrate*** the sensor
  - Measure a range of temperatures with the inexpensive sensor and an accurate sensor
  - Obtain a dataset representing sensor temperature, T_s, as a function of true temperature, T
  - That is, determine a_1 and a_0 such that

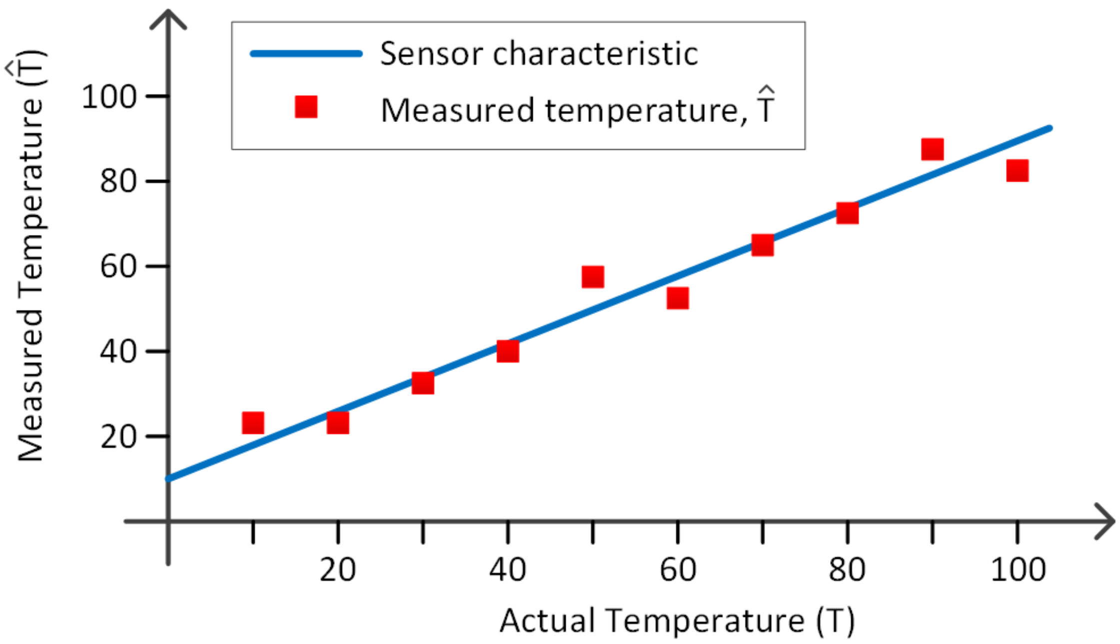
$$T_s = f(T) = a_1 T + a_0$$

- Then, we can map sensor temperature to true temperature

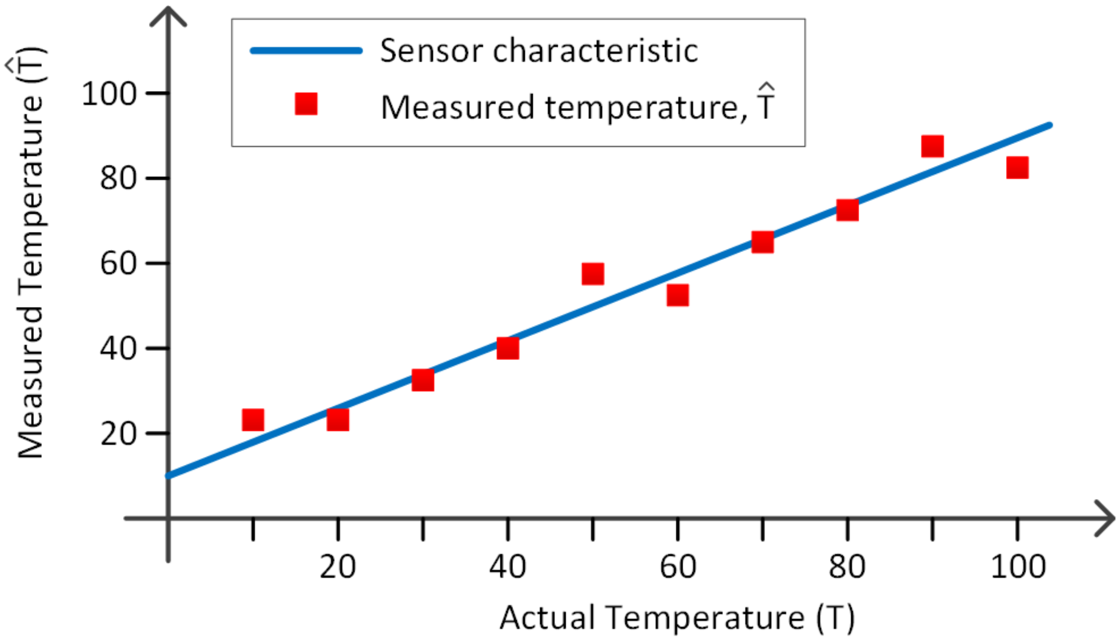
$$T = \frac{T_s}{a_1} - \frac{a_0}{a_1}$$

Curve Fitting

- In practice, there would be two sources of error between actual and measured temperatures
  - Inherent sensor inaccuracy
  - ***Measurement noise***
- Actual ***measured*** data, $\hat{T}$, may look like:



Curve Fitting



- Determine the blue line (a_1 and a_0) that provides the ***best fit*** to the measured data (red squares)
- How do we define "***best fit***"?

Least-Squares Fit

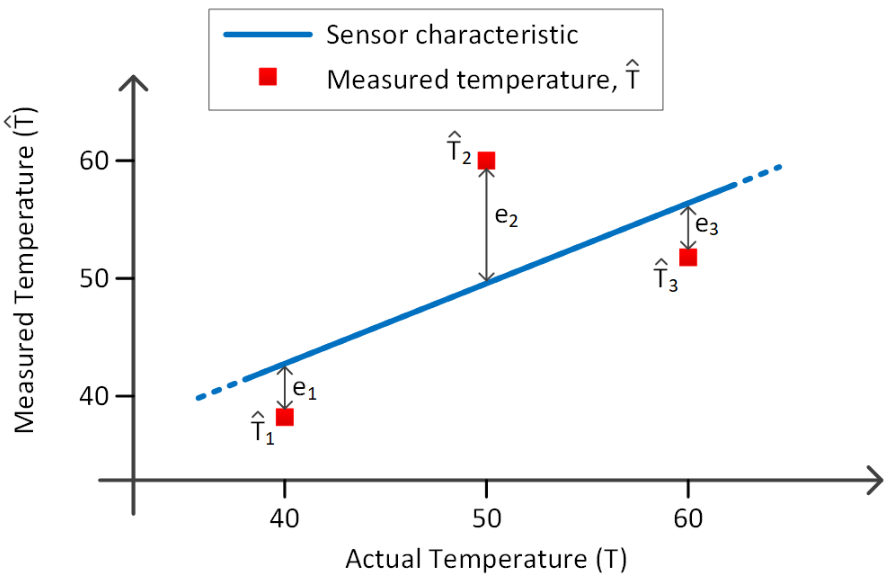
- What constitutes the ***best fit***?
- Want to determine inherent sensor behavior,

$$T_s = a_1 \cdot T + a_0$$

given noisy measurement data,

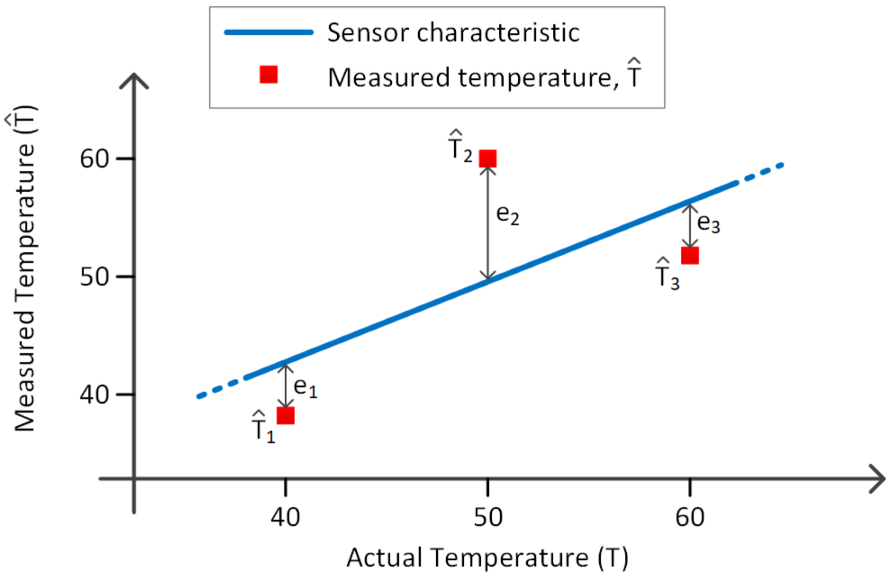
$$\hat{T} = T_s + e$$

where e represents measurement error



Least-Squares Fit

- Errors between data points and the line fit to the data are called ***residuals***
- Best fit criterion:
  - Minimize the ***sum of the squares of the residuals***
  - ***A least-squares fit***



- Minimize:

$$S_r = \sum_i e_i^2 = \sum_i \left[\hat{T}_i - (a_1 T_i + a_0)\right]^2$$

Goodness of Fit

- How well does a function fit the data?
- Is a linear fit best? A quadratic, higher-order polynomial, or other non-linear function?
- Want a way to be able to quantify ***goodness of fit***

- Quantify ***spread of data about the mean*** prior to regression:

$$S_t = \sum (\hat{y}_i - \bar{y})^2$$

- Following regression, quantify ***spread of data about the regression line*** (or curve):

$$S_r = \sum (\hat{y}_i - a_0 - a_1 x_i)^2$$

Goodness of Fit

- S_t quantifies the spread of the data about the mean
- S_r quantifies spread about the best-fit line (curve)
  - The spread that remains after the trend is explained
  - The ***unexplained sum of the squares***
- $S_t - S_r$ represents the reduction in data spread after regression explains the underlying trend
- Normalize to S_t - the ***coefficient of determination***

$$\boxed{r^2 = \frac{S_t - S_r}{S_t}}$$

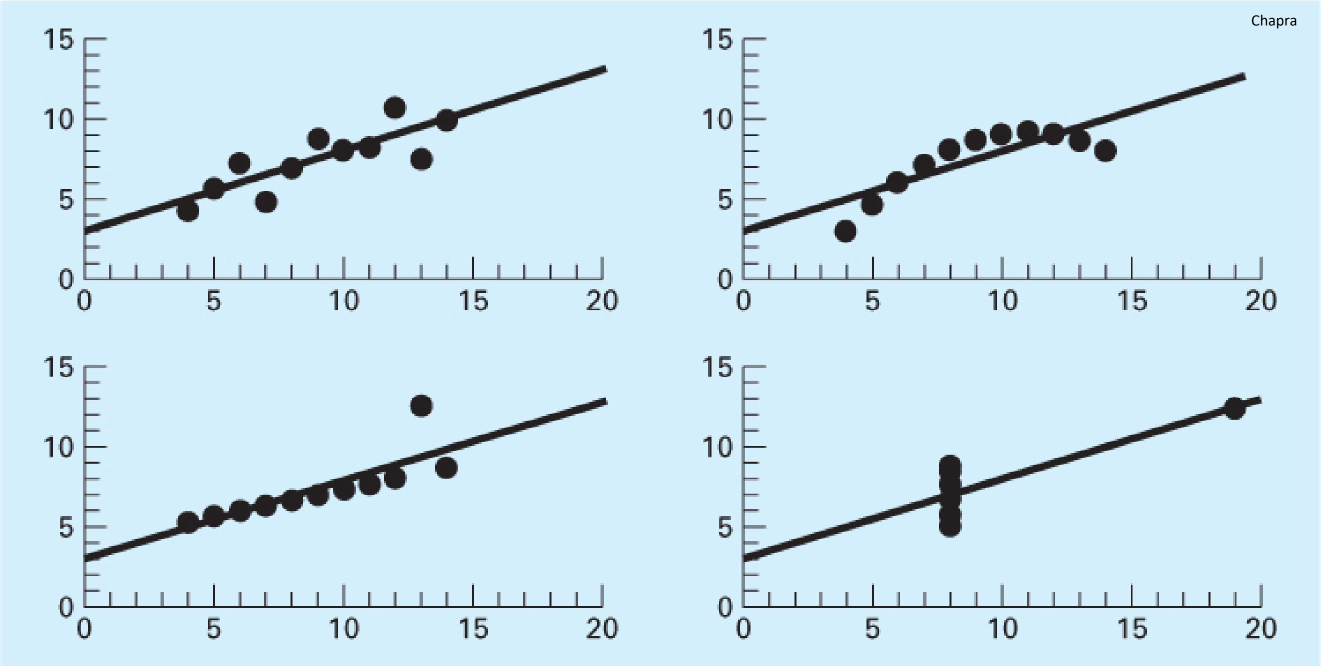
Coefficient of Determination

$$r^2 = \frac{S_t - S_r}{S_t}$$

- For a perfect fit:
  - No variation in data about the regression line
  - $S_r = 0 \quad \rightarrow \quad r^2 = 1$
- If the fit provides no improvement over simply characterizing data by its mean value:
  - $S_r = S_t \quad \rightarrow \quad r^2 = 0$
- If the fit is worse at explaining the data than their mean value:
  - $S_r > S_t \quad \rightarrow \quad r^2 < 0$

Coefficient of Determination

- Don't rely too heavily on the value of r^2
- Anscombe's famous data sets:



- Same line fit to all four data sets
- $r^2 = 0.67$ in each case

Curve Fitting in MATLAB

- So far we have considered fitting a line to data
  - A linear least-squares line fit
- Can also fit other functions to data, e.g.,
  - Higher-order polynomials – quadratic, cubic, etc.
  - Exponentials
  - Sinusoids
  - Power equation, etc.
- MATLAB has built-in functions to perform curve fitting
  - `polyfit.m` – for fitting polynomials
  - `fit.m` – for fitting any other user-specified curves

Polynomial Regression – `polyfit.m`


```
p = polyfit(x,y,m)
```

- `x`: n-vector of independent variable data values
- `y`: n-vector of dependent variable data values
- `m`: order of the polynomial to be fit to the data ($m < n$)
- `p`: $(m + 1)$-vector of best-fit polynomial coefficients

- Polynomial coefficients in MATLAB
  - Consider a polynomial created by `polyfit.m`

$$y = a_2 x^2 + a_1 x + a_0$$

  - MATLAB would return

$$\mathtt{p = [a_2, a_1, a_0]}$$

Polynomial Evaluation – polyval.m

- n^th-order polynomial represented as $(n+1)$-vector
- For example, the cubic polynomial

$$y = 2x^3 - 8x^2 + 3x - 4$$

would be represented as

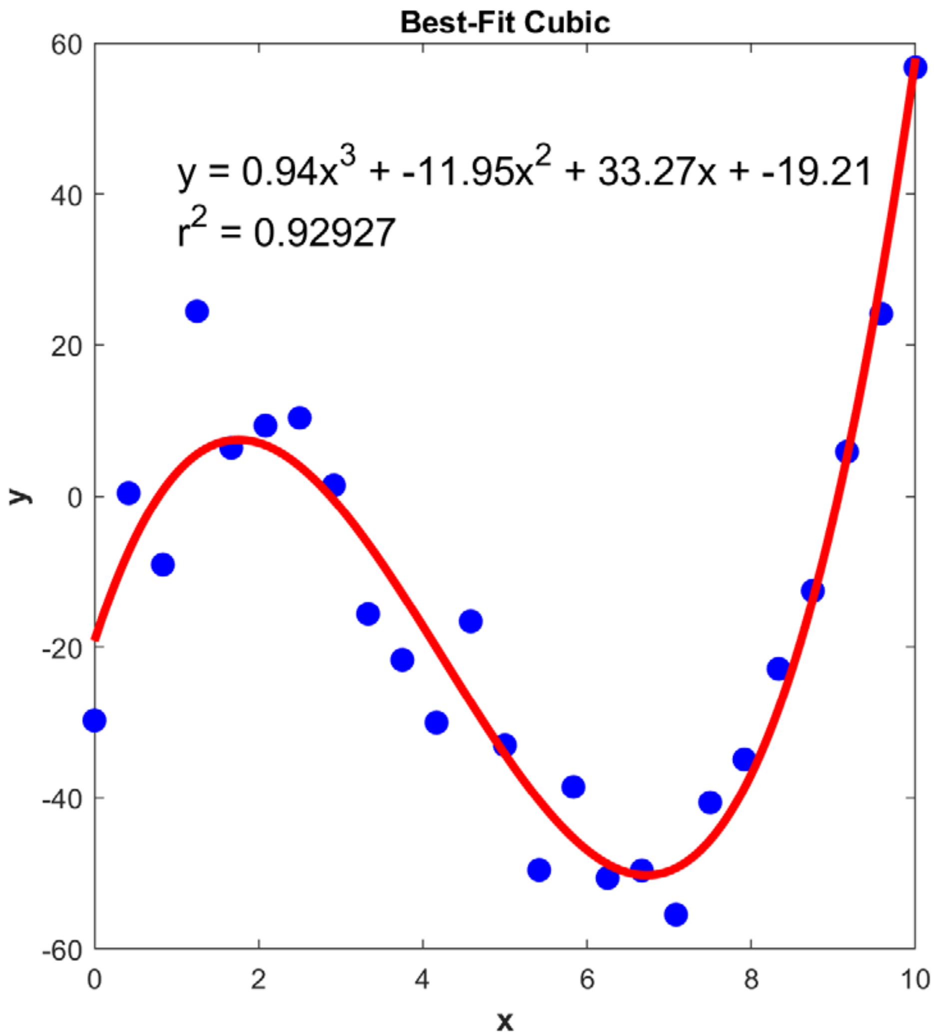
```
p = [2,-8,3,-4]
```

- Use polyval.m to evaluate that polynomial over a vector of independent variable values

```
y = polyval(p,x)
```

  - p: $(n+1)$-vector of n^{th}-order polynomial coefficients
  - x: vector of independent variable data values
  - y: vector result of evaluating the polynomial at all values in x

Polynomial Fit – Example




```matlab
%% create dataset
% noiseless data
% polynomial with roots at 1, 3, and 9
% y = x^3 - 13x^2 + 39x - 27
p = poly([1,3,9]);
x = linspace(0,10,25);
y = polyval(p,x);

% add noise to y data
sig = 0*8;
v = sig*randn(size(y));
yn = y + v;


%% use polyfit.m to perform the fit

pfit = polyfit(x,yn,3);


%% evaluate the best-fit cubic

xfit = linspace(min(x),max(x),200);
y3 = polyval(pfit,xfit);
y3r2 = polyval(pfit,x);


%% coefficient of determination

ybar = mean(yn);
St = sum((yn - ybar).^2);
Sr = sum((yn - y3r2).^2);
r2 = (St - Sr)/St
```

Fitting User-Specified Curves – `fit.m`

- To fit a curve other than a polynomial, use `fit.m`

```
fitobject = fit(x,y,fittype)
```

  - `x`: ***column*** vector of independent variable data values
  - `y`: ***column*** vector of dependent variable data values
  - `fittype`: model type to fit – specified as a library model or created with the `fittype.m` function
  - `fitobject`: `cfit` object containing fit parameters

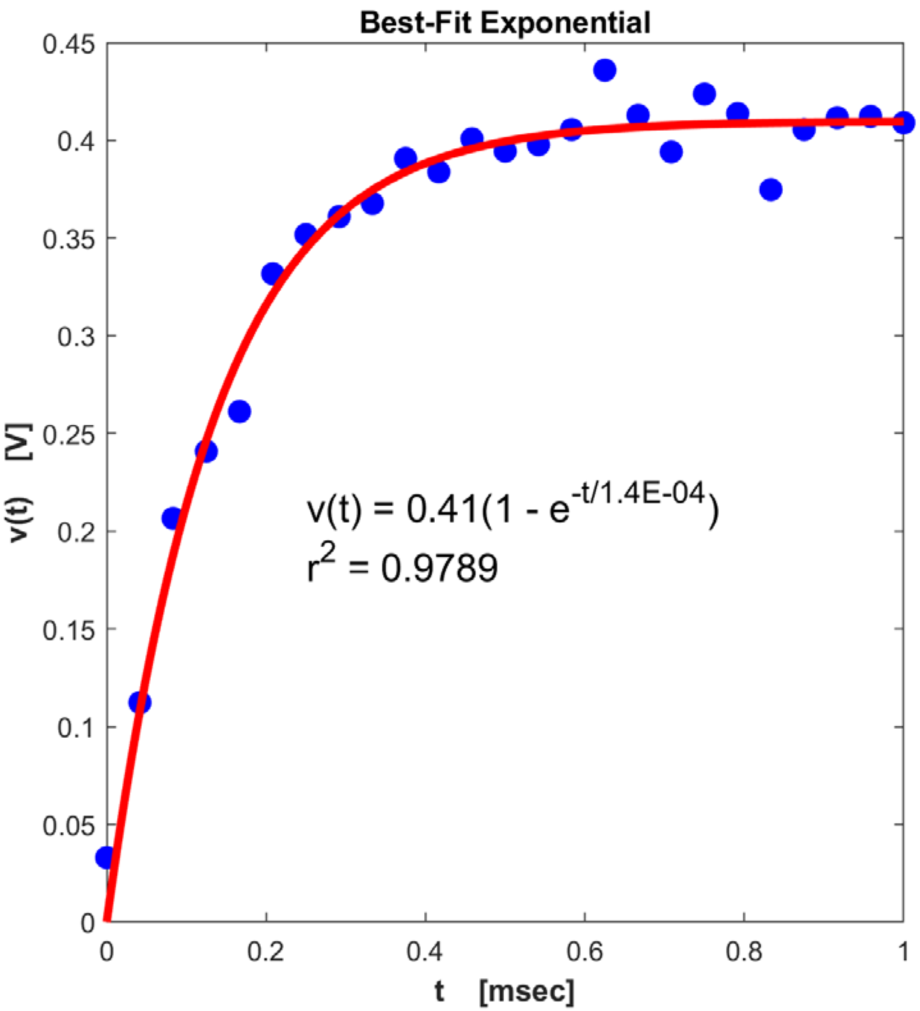
Specifying the Model – `fittype.m`

- Define the model to be used by `fit.m`

```
fitmod = fittype(expression,name,value,…)
```

  - `expression`: mathematical function to be fit to data – user-specified or a standard library model (see help)
  - `name`: property name to specify (see help)
  - `value`: value assigned to `name` – can specify multiple property name/value pairs
  - `fitmod`: `fittype` object to be passed to fit.m

Exponential Fit - Example




```
%% create dataset
% noiseless data
t = linspace(0,1e-3,25)';
tau = 120e-6;                       % time constant
v = 400e-3*(1 - exp(-t/tau));

% add noise to y data
sig = 15e-3;
n = sig*randn(size(v));
vn = v + n;

%% fit an exponential to the data
expFit = fittype('Vf*(1 - exp(-t/tau))',...
    'independent','t');

expFitObj = fit(t,vn,expFit);

%% extract fit parameters from cfit object
Vf_fit = expFitObj.Vf;
tau_fit = expFitObj.tau;

%% evaluate the fit
tfit = linspace(0,t(end),2000);
vfit = Vf_fit*(1 - exp(-tfit/tau_fit));
vfitr2 = Vf_fit*(1 - exp(-t/tau_fit));

%% coefficient of determination
vbar = mean(vn);
St = sum((vn - vbar).^2);
Sr = sum((vn - vfitr2).^2);
r2 = (St - Sr)/St
```