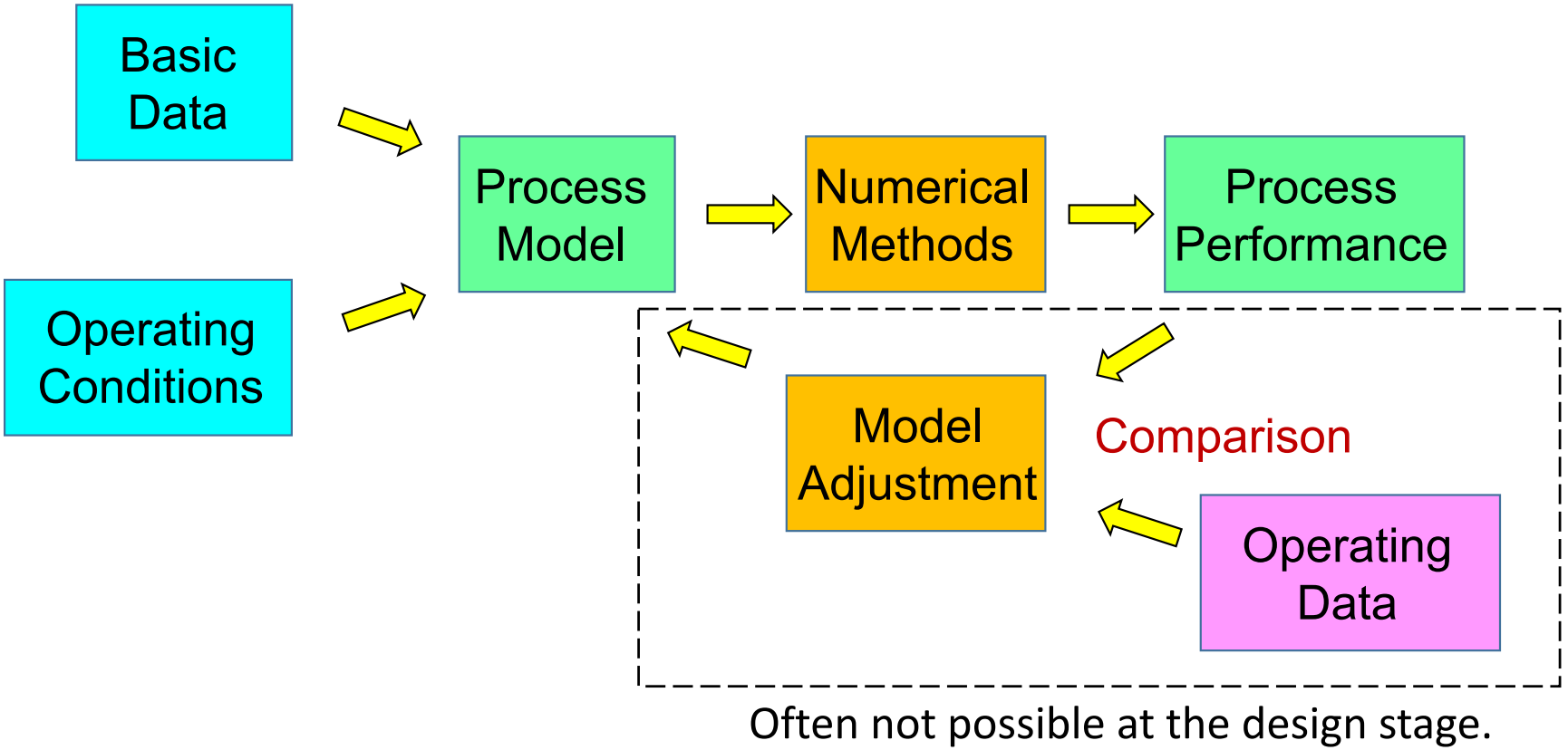
MATLAB Bootcamps 1, 2 and 3

- ✓ 1: Getting up to speed (or back up to speed) with MATLAB
- 2: Learning to use MATLAB to solve typical problem scenarios
- 3: Detailed modeling of packed-bed and plug-flow reactors

Bootcamp 2 Outline

- Process modeling & numerical characteristics
- Algebraic models
  - Single, nonlinear
  - Linear sets
  - Nonlinear sets
- ODE models
  - Initial value problems
  - Split boundary problems
- Optimization
- Curve-fitting

Process Modeling



Often not possible at the design stage.

Process Modeling

Developing the Process Model

Conservation Balances

- Material
- Energy
  - Mechanical/Momentum
  - Thermal
- Thermodynamics
  - Equilibrium, Phase and Chemical
  - Heat Transfer
- Mass Transfer
- Separations
- Reaction Kinetics
- Monitoring and Control

Equipment

- Vessels
  - Tanks, Drums
  - Columns
  - Reactors
- Heat Exchangers
- Piping, Valves, Fittings
- Pumps, Compressors
- Columns
- Reactors
- Solids-handling
  - Crystallization
  - Filtration
- Instrumentation

Numerical Characteristics

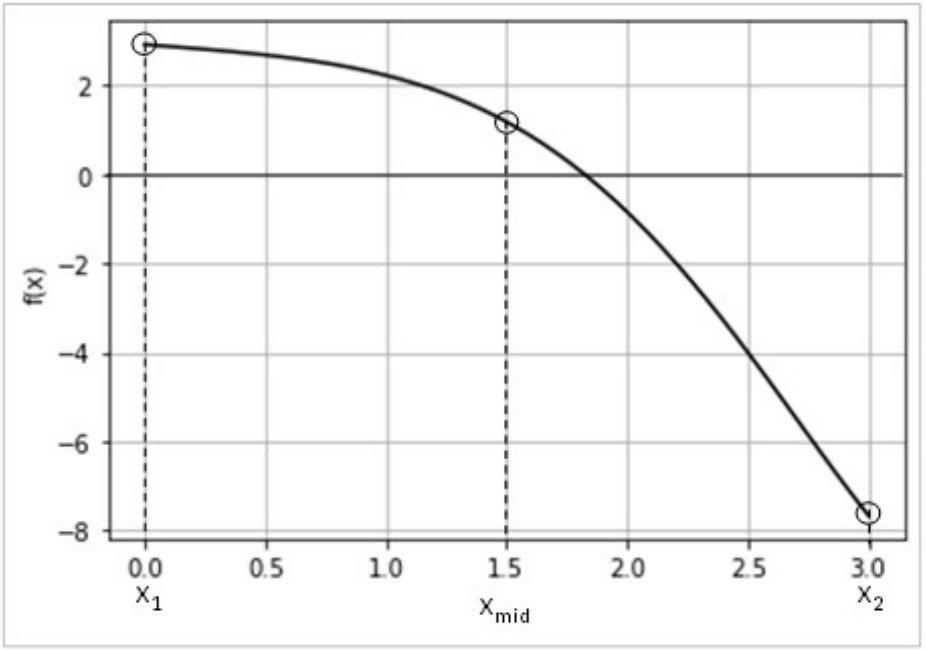
- Algebraic equations
  - Nonlinear, one or more
  - Linear sets
- Differential equations
  - Ordinary (ODEs)
  - Partial (PDEs)
- Optimization
- Curve-fitting

when combined:
Differential-Algebraic Systems (DAEs)

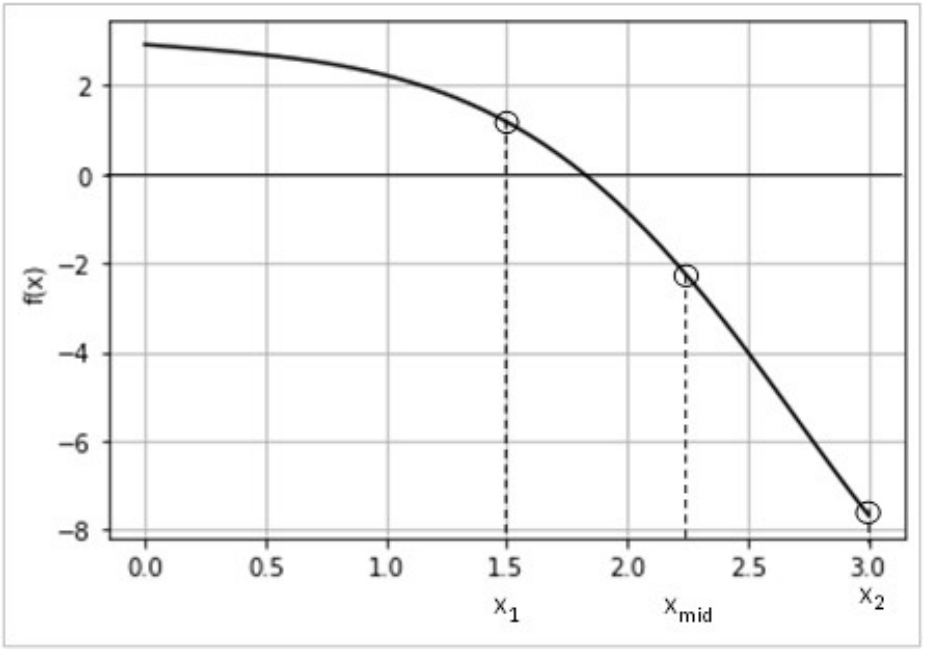
Examples Considered

- Single, nonlinear algebraic equation
  - water-gas shift equilibrium
- Set of linear algebraic equations
  - absorber column
- Set of nonlinear algebraic equations
  - steam/water equilibrium
- Single nonlinear ordinary differential equation
  - batch reactor, single reaction
- Set of nonlinear ordinary differential equations
  - batch reactor, multiple reactions
- Second-order ordinary differential equation
  - split boundary conditions
- Set of linear differential equations
  - split boundary conditions
  - countercurrent heat exchanger
- Optimization
  - single factor, humps equation
  - multiple factors with constraints
    grain bin design
- Linear Regression
  - polynomial
    density of MeOH-H2O solutions
  - general, NaCl solution density
- Nonlinear Regression
  - Antoine equation

Solving Single Algebraic Equations - Bisection



First iteration



Second iteration

Solving an Algebraic Equation with Bisection

```
function root = bisect(fun,x1,x2)
% uses the bisection method to estimate a root
% of fun(x)=0 between x1 and x2.
% the method is iterated 20 times to provide
% 1 part in 2^20 of the original x1-x2 interval.
if fun(x1)*fun(x2) > 0
    root = 'initial estimates do not bracket the solution';
else
    for i=1:20
        xm = (x1+x2)/2;
        if fun(xm)*fun(x1) > 0
            x1 = xm;
        else
            x2 = xm;
        end
    end
    root = xm;
end
```

bisect.m

Solving an Algebraic Equation with Bisection

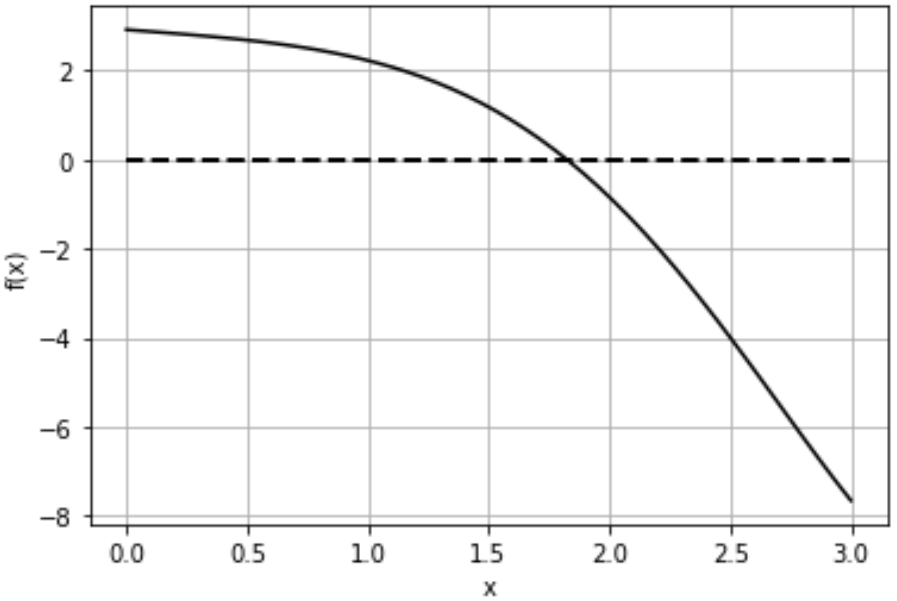
Example $f(x) = \sin(x+2) \cdot \cosh(x) + 2 = 0 \qquad x_1 = 0 \qquad x_2 = 3$

```
function fval = fn(x)
fval = sin(x+2)*cosh(x)+2.;
```

fn.m

```
x1 = 0.;
x2 = 3.;
xsoln = bisect(@fn,x1,x2);
disp(xsoln)

>> bisect_test
    1.8229
```

bisect_test.m




```
x1 = 4.;
x2 = 3.;
xsoln = bisect(@fn,x1,x2);
disp(xsoln)
```

```
>> bisect_test
initial estimates do not bracket the solution
```

Solving an Algebraic Equation with Bisection

Adding the number of iterations as an optional argument

bisect1.m
bisect1_test.m

```
function root = bisect1(fun,x1,x2,varargin)
% uses the bisection method to estimate a root
% of fun(x)=0 between x1 and x2.
% number of iterations is 20 by default
% 1 part in 2^20 of the original x1-x2 interval, but
% additional argument can specify iterations.
nargin = length(varargin);
if nargin == 0
    maxit = 20;  % here if no extra argument
else
    maxit = cell2mat(varargin(1));  % here with
end                                 % extra argument
if fun(x1)*fun(x2) > 0
    root = 'initial estimates do not bracket the solution';
else
    for i=1:maxit
        xm = (x1+x2)/2;
        if fun(xm)*fun(x1) > 0
            x1 = xm;
        else
            x2 = xm;
        end
    end
    root = xm;
end
```

- **varargin** indicates optional extra argument(s)
- **nargin** provides the number of arguments provided
- **varargin** is a cell array
- **cell2mat** converts first element to numerical type

```
clc
clear
x1 = 1.;
x2 = 3.;
xsoln = bisect1(@fn,x1,x2,25);
disp(xsoln)
```

```
1.8229
```

Solving an algebraic equation with MATLAB's fzero function

$$f(x) = 0$$

fzero_example.m

f.m

xs = fzero(f,x0)

f : function "handle", @name

x0 : initial estimate for solution

Example $f(x) = \sin(x) \cdot x - 1 = 0$ $x_0 = 0.5$

f.m

```
function ferr = f(x)
ferr = sin(x)*x-1;
```

fzero_example.m

```
% solve single algebraic equation
% using fzero function
x0 = 0.5;  % initial estimate
xsoln = fzero(@f,x0)
```

```
>> fzero_example

xsoln =

    1.1142
```

The **fzero** function uses a combination of the bisection, secant, and inverse quadratic interpolation methods, switching between the methods to preserve stability and increase efficiency of convergence. See MATLAB Help.

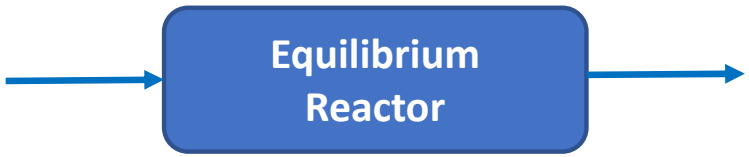
Example: water-gas shift equilibrium

$$CO + H_2O \Leftrightarrow H_2 + CO_2$$

$$\frac{[H_2]\cdot[CO_2]}{[H_2O]\cdot[CO]} = K_{eq}(T) \qquad ln\left[K_{eq}(T)\right] = -3.112 + \frac{3317}{T} \qquad T(K)$$

Feed

	kmol/hr
H_2	450
CO	500
CO_2	50
H_2O	1150



Product

T = 1200 °C
P = 1 atm

$$f(x) = \frac{\left[Feed_{H_2} + x\right]\cdot\left[Feed_{CO_2} + x\right]}{\left[Feed_{H_2O} - x\right]\cdot\left[Feed_{CO} - x\right]} - K_{eq}(T) = 0$$

where x is the shift to equilibrium in kmol/hr.

Solve for x and the reactor product flow rates for the given temperature.

Water-gas shift equilibrium

Setting up a function to compute f(x)

fwg.m

```
% function to compute f(x)
% for water-gas shift equilibrium
function ferr = f(x)
T=1200; % degC
TK = T+273.15; % K
Keq = exp(-3.112+3317/TK); % equilbrium constant
% feed specifications
FeedH2 = 450; % kmol/hr
FeedCO2 = 50;
FeedH2O = 1150;
FeedCO = 500;
% equation error -- allow for vector x
ferr = (FeedH2+x).*(FeedCO2+x)./(FeedH2O-x)./(FeedCO-x)-Keq;
```

```
>> fwg(150)

ans =

   -0.0802
```

```
>> fwg(175)

ans =

    0.0208
```

Solution apparently between 150 and 175 °C.

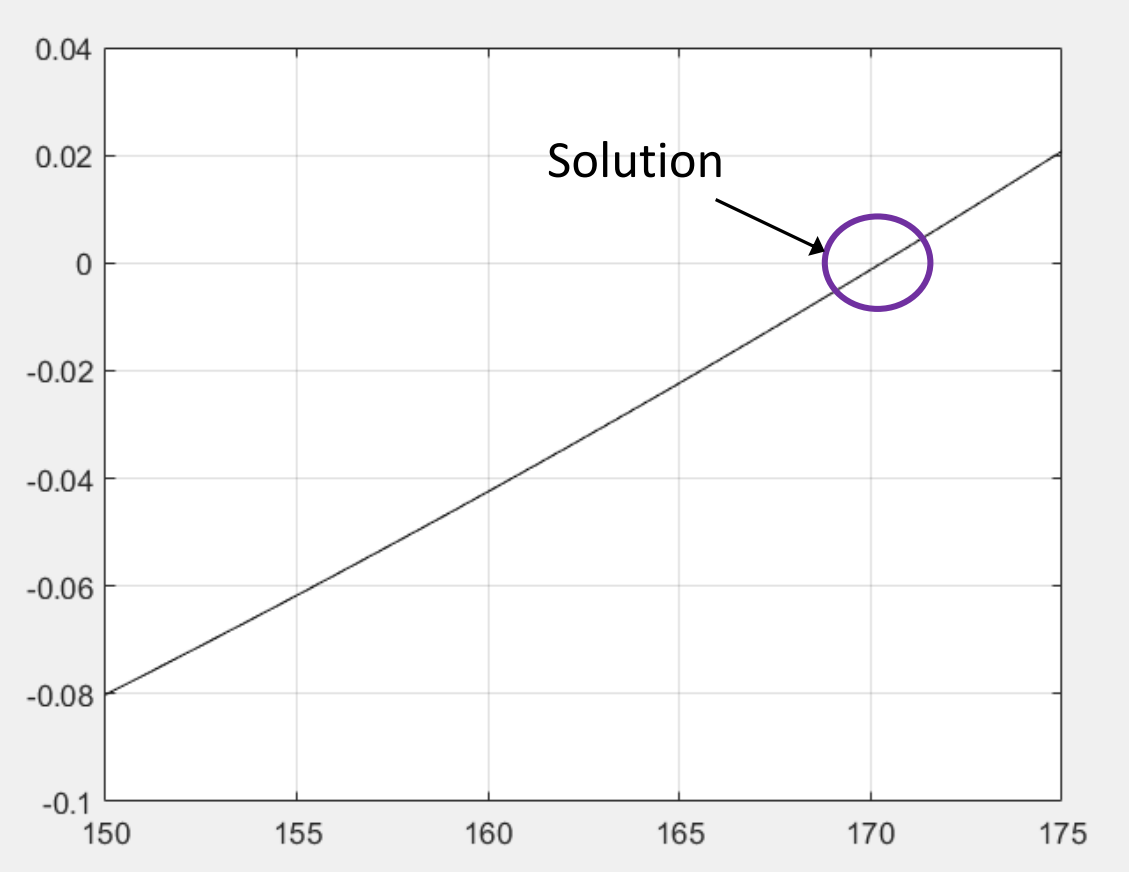
Water-gas shift equilibrium

Carrying out a case study of the equation error

fwg.m | WaterGasEqnCaseStudy.m

```
1  % case study
2  % water-gas shift equilibrium
3  % at 1200 degC
4  x = 150:175;
5  plot(x,fwg(x),'k-')
6  grid
```

WaterGasCaseStudy.m



Water-gas shift equilibrium

Including temperature and feed specs as arguments to the function

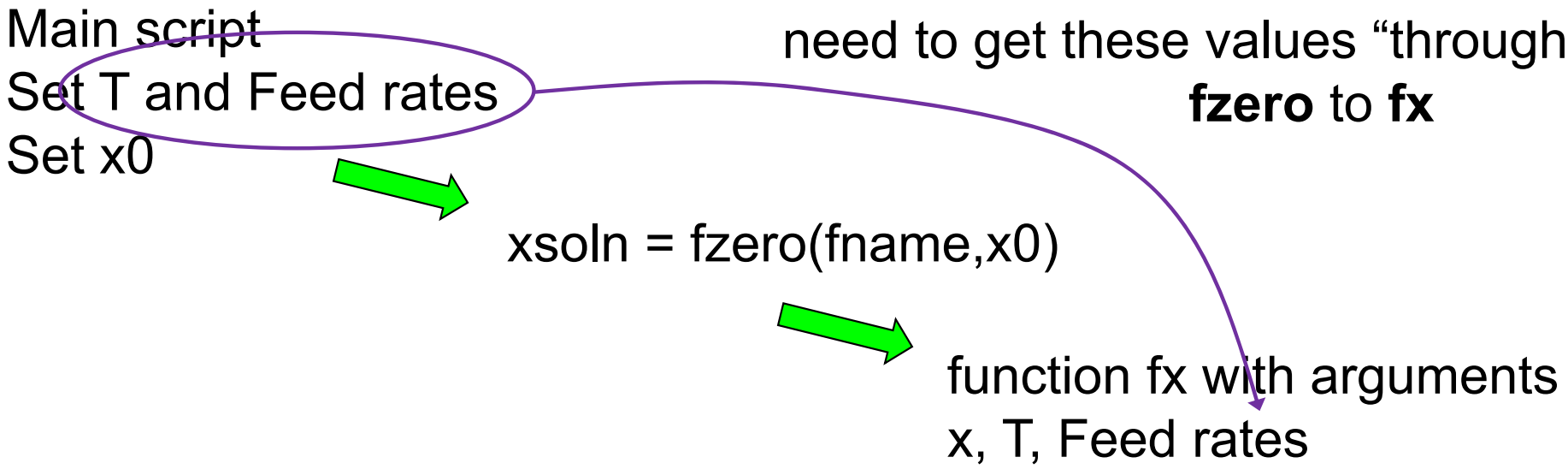
fx.m

```
% function to compute f(x)
% for water-gas shift equilibrium
% set up with external arguments
function ferr = fx(x,T,FeedH2,FeedCO2,FeedH2O,FeedCO)
TK = 273.15 + T; % K
Keq = exp(-3.112+3317/TK); % equilibrium constant
% equation error
ferr = (FeedH2+x).*(FeedCO2+x)./(FeedH2O-x)./(FeedCO-x)-Keq;
```

fx.m

Water-gas shift equilibrium

Use of an anonymous function




```
fanon = @(x) fx(x,T,FeedH2,FeedCO2,FeedH2O,FeedCO);
xsoln = fzero(fanon,[50,450]);
```

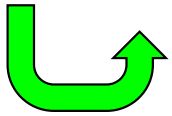
here, we use an interval for the initial estimates

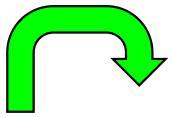
Water-gas shift equilibrium

Case study for a range of temperatures with a plot

WaterGasTempCaseStudyWithPlotUsingfzero.m | WaterGasTempCaseStudy.m

```
% script for case study
% of water-gas shift equilibrium
% over a range of temperatures
% set up with external arguments
% feed specifications
FeedH2 = 450; % kmol/h
FeedCO2 = 50;
FeedH2O = 1150;
FeedCO = 500;
T = 500:25:1500;  % array of temperatures
n = length(T);
```



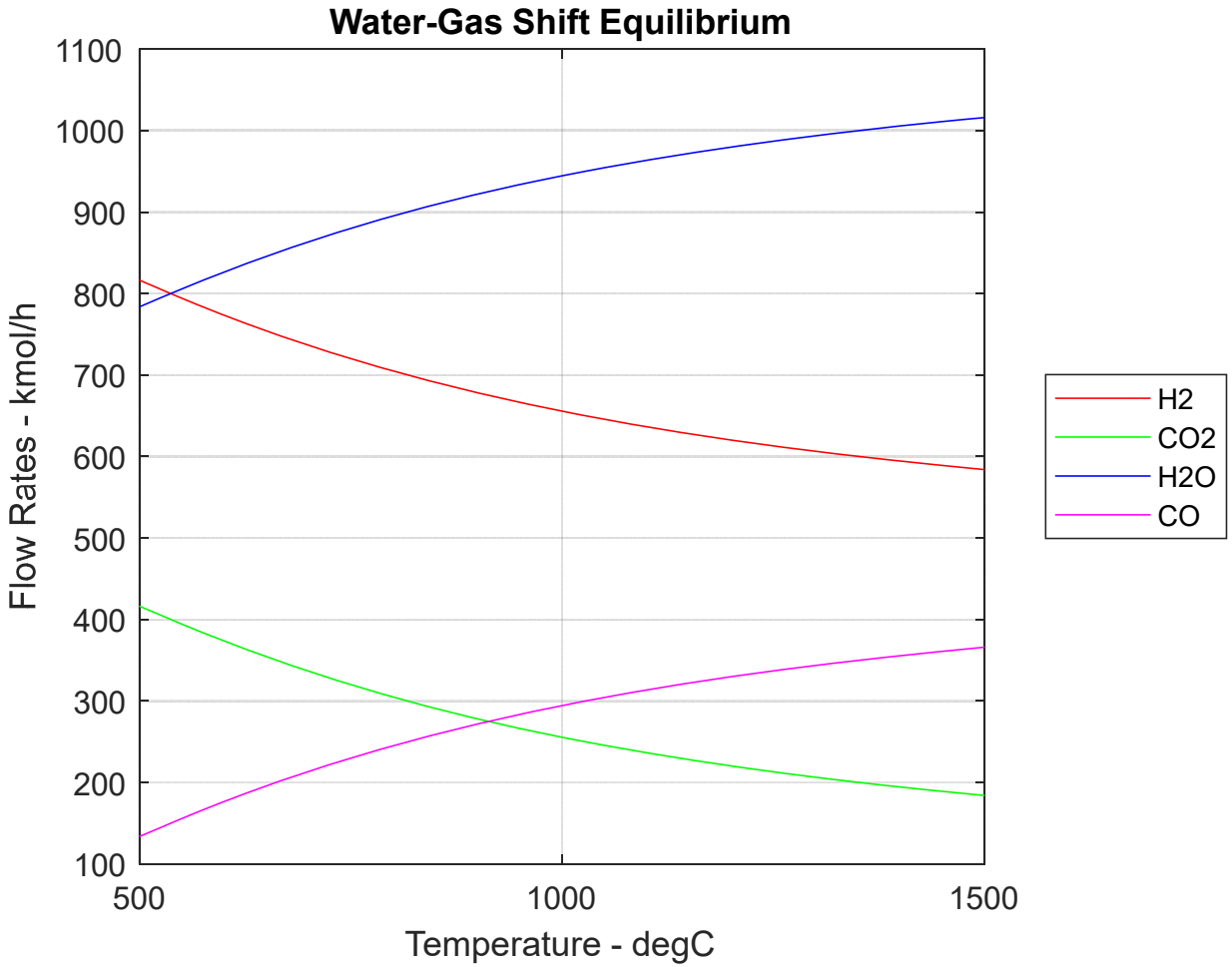


```
for i = 1:n
    fanon = @(x) fx(x,T(i),FeedH2,FeedCO2,FeedH2O,FeedCO)
    xsoln(i) = fzero(fanon,[50,450]);
    ProdH2(i) = FeedH2 + xsoln(i);
    ProdCO2(i) = FeedCO2 + xsoln(i);
    ProdH2O(i) = FeedH2O - xsoln(i);
    ProdCO(i) = FeedCO - xsoln(i);
end
plot(T,ProdH2,'c',T,ProdCO2,'g',T,ProdH2O,'b',T,ProdCO,'m')
grid; grid minor
xlabel('Temperature - degC')
ylabel('Flow Rate - kmol/h')
title('Water-Gas Shift Equilibrium')
legend('H2','CO2','H2O','CO','Location','eastoutside')
```

WaterGasShiftCaseStudyWithPlotWithfzero.m

Water-gas shift equilibrium

Case study for a range of temperatures with a plot



Solving Sets of Linear Algebraic Equations

n equations in *n* unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \qquad \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad \mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

Solving the equations:

1. matrix algebra and computations

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

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

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

compute the inverse of **A** and multiply it by **b**

2. more efficient numerical method
   - Gaussian elimination with enhancements
   - LU decomposition

Solving Sets of Linear Algebraic Equations

Example

$$3x + 2y - z = 10$$
$$-x + 3y + 2z = 5$$
$$x - y - z = -1$$

$$\begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

$A^{-1}*b$ ⇨ not recommended

```
A = [ 3 2 -1 ; -1 3 2 ; 1 -1 -1];
b = [10 5 -1]';
x = inv(A)*b
```

```
>> LinAlgEqsExample1

x =

   -2.0000
    5.0000
   -6.0000
```

```
x = A\b
```

Using "left divide" preferred also **mldivide**(A,b).
Versatile solver, typically uses LU decomposition.

```
x = linsolve(A,b)
```

Uses LU decomposition.

Solving Sets of Linear Algebraic Equations

Example problem: Six-stage absorber column

Equilibrium relationship on tray *i* $y_i = ax_i + b$

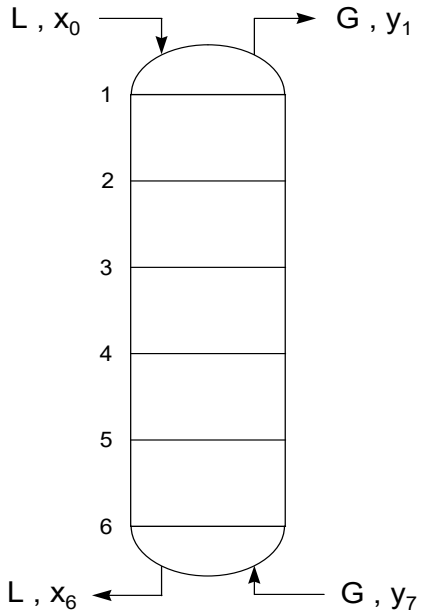
x_0 , y_7 , L and G specified

Component material balance on tray *i*

$$L \cdot x_{i-1} + G \cdot y_{i+1} = L \cdot x_i + G \cdot y_i$$

Incorporate equilibrium relationship

$$L \cdot x_{i-1} - \left(L + G \cdot a\right) \cdot x_i + G \cdot a \cdot x_{i+1} = 0$$



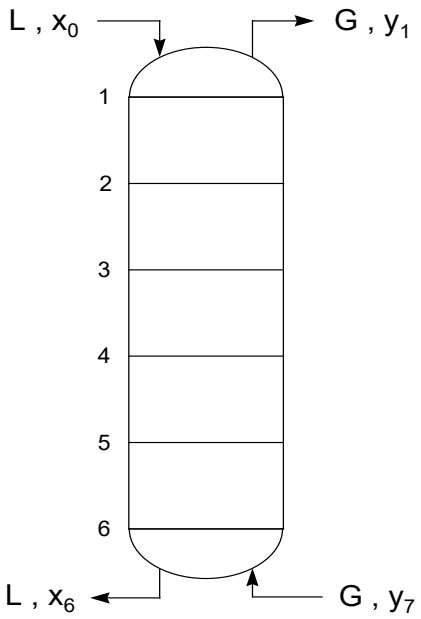
Solving Sets of Linear Algebraic Equations

Example problem: Six-stage absorber column

Write component material balances for each tray and rearrange with unknowns on the left and knowns on the right.

$$-(L+Ga)x_1 + Gax_2 = -Lx_0$$
$$Lx_1 - (L+Ga)x_2 + Gax_3 = 0$$
$$Lx_2 - (L+Ga)x_3 + Gax_4 = 0$$
$$Lx_3 - (L+Ga)x_4 + Gax_5 = 0$$
$$Lx_4 - (L+Ga)x_5 + Gax_6 = 0$$
$$Lx_5 - (L+Ga)x_6 = -G(y_7 - b)$$

This represents a set of six linear equations in the six unknown mass fractions.



Basic data: equilibrium model: a = 0.7, b = 0
Operating conditions: L = 20 mol/s, G = 12 mol/s
Inlet gas mole fraction: $y_7 = 0.1$
Inlet liquid mole fraction: $x_0 = 0$

Solving Sets of Linear Algebraic Equations

Example problem: Six-stage absorber column

```
% solve set of linear equations
% for six-stage absorber
a = 0.7; % equilibrium factor
yin = 0.1; % vapor entry mole fraction
L = 20; % mol/s
G = 12; % mol/s
n = 6;  % number of stages
% compose A matrix
for i = 1:n
    A(i,i) = -(L+G*a);
end
for i = 1:n-1
    A(i,i+1) = G*a;
end
for i = 2:n
    A(i,i-1) = L;
end
% constant vector b
b = zeros(6,1);
b(6) = -G*yin;
% solve system of equations
x = A\b;
disp(x)
```

```
>> Absorber
    0.0003
    0.0014
    0.0040
    0.0101
    0.0248
    0.0597
```

Absorber.m

Solving Sets of Nonlinear Algebraic Equations

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \quad \text{or} \quad \mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Common solution technique: Newton's Method

Start with an initial estimate of the solution: $\mathbf{x}^0$

Iterate with $\mathbf{x}^{i+1} = \mathbf{x}^i - \mathbf{J}^{-1}(\mathbf{x}^i) \cdot \mathbf{f}(\mathbf{x}^i)$ until a convergence criterion is met.

Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

or, where analytical derivatives are difficult:

$$\frac{\partial f_1}{\partial x_1}(\mathbf{x}^i) \cong \frac{f_1(x_1^i + \delta, x_2^i, \ldots, x_n^i) - f_1(x_1^i - \delta, x_2^i, \ldots, x_n^i)}{2 \cdot \delta}$$

and so forth.

Solving Sets of Nonlinear Algebraic Equations

Solution with Matlab's **fsolve** function

Example $x^2 + y^2 - 4 = 0$

$x \cdot y - 1 = 0$

$$\mathbf{J} = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}$$

x = fsolve(fun,x0)

without supplying **J**

solvemyeqns.m | myeqns.m

```
1   x0 = [0.6 0.4];
2   x = fsolve(@myeqns,x0)
```

myeqns.m | myeqns.m

```
function ferr = myeqns(x)
ferr(1) = x(1)^2 + x(2)^2-4;
ferr(2) = x(1)*x(2)-1;
```

```
x =

    1.9319    0.5176
```

supplying **J**

```
x0 = [0.6 0.4];
x = fsolve(@myeqnsJ,x0)
```

```
function [ferr,J] = myeqns(x)
ferr(1) = x(1)^2 + x(2)^2-4;
ferr(2) = x(1)*x(2)-1;
J = [ 2*x(1) 2*x(2); x(2) x(1)];
```

Solving Sets of Nonlinear Algebraic Equations

Example problem: steam/water equilibrium

$$P \cdot V = \frac{m}{MW} \cdot R \cdot (T + 273.15) \qquad log_{10} P = A - \frac{B}{T + C}$$

ideal gas law — Antoine equation

P : absolute pressure, Pa

V : vapor volume, m^3

m : mass of vapor, kg

MW: H_2O molecular weight, ≅ 18.02 kg/kgmol

R : gas law constant, 8314 $(Pa \cdot m^3)/(kmol \cdot K)$

T : temperature, °C

A, B, C : Antoine constants for H_2O

$A = 11.21 \quad B = 2354.7 \quad C = 280.71$

Operating conditions: $m = 3.755\ kg \quad V = 3.142\ m^3$

Solve for P and T .

Solving Sets of Nonlinear Algebraic Equations

Example problem: steam/water equilibrium

Formulating the problem for solution

$$f_1(T,P) = P \cdot V - \frac{m}{MW} \cdot R \cdot (T + 273.15)$$

$$f_2(T,P) = log_{10} P - A + \frac{B}{T + C}$$

$$\mathbf{J}\left(\begin{bmatrix} P \\ T \end{bmatrix}\right) = \begin{bmatrix} V & -\frac{m}{MW} \cdot R \\ \frac{1}{ln(10) \cdot P} & -\frac{B}{(C+T)^2} \end{bmatrix}$$

analytical Jacobian practical in this case

A possible issue here is the comparative scaling of the two equations. Typical values for the PV term could be of magnitude 10^6; whereas, terms in the second equation are closer to unity. A practical approach to this is to scale the first equation by dividing it by, e.g., 100,000.

$$f_1(T,P) = \left(P \cdot V - \frac{m}{MW} \cdot R \cdot (T + 273.15) \right) \Big/ 100000$$

$$f_2(T,P) = log_{10} P - A + \frac{B}{T + C}$$

$$\mathbf{J}\left(\begin{bmatrix} P \\ T \end{bmatrix}\right) = \begin{bmatrix} V/1e5 & -\frac{m}{MW} \cdot R \Big/ 1e5 \\ \frac{1}{ln(10) \cdot P} & -\frac{B}{(C+T)^2} \end{bmatrix}$$

Solving Sets of Nonlinear Algebraic Equations

Example problem: steam/water equilibrium

SolveStmEquil.m

```
% solve steam equilibrium problem
% with analytical Jacobian
clear;clc
% specify mass and volume of vapor
m = 3.755; % kg
V = 3.142; % m3
% initial estimates for pressure and temperature
x0 = [ 200000 110 ];
% create anonymous function to pass m and V
stmanon = @(x) SteamEq(x,m,V);
% set option for analytical Jacobian and to suppres display
options = optimoptions('fsolve','SpecifyObjectiveGradient',true ...
    ,'Display','none');
% call fsolve function to solve the equations
xsoln = fsolve(stmanon,x0,options);
% display the solution
fprintf('Pressure = %8.1f Pa\nTemperature = %6.2f degC\n',xsoln)
```

SteamEq.m

```
% function to compute equation errors
% and Jacobian matrix
function [ferr,J] = SteamEq(x,m,V)
R = 8314. ; % Pa*m3/(kmol*K)
MW = 18.02; % kg/kmol
A = 11.21 ; % Antoine coefficients
B = 2354.7 ;
C = 280.7 ;
P = x(1) ;
T = x(2);
% ideal gas law scaled by 100000
ferr(1) = (P*V - m/MW*R*(T+273.15))/1e5 ;
% Antoine equation
ferr(2) = log10(P) - A + B/(T+C);
% Jacobian matrix
J(1,1) = V/1.e5; J(1,2) = -m/MW*R/1.e5;
J(2,1) = 1/log(10)/P ; J(2,2) = -B/(C+T)^2;
```

```
Pressure = 216878.2 Pa
Temperature = 120.18 degC
```

Solving Single Differential Equations

Two types

generally, analytical solutions are not feasible

$$\frac{dy}{dt} = f(t) \quad \Longrightarrow \quad \int_{y_0}^{y_f} dy = y_f - y_0 = \int_{t_0}^{t_f} f(t)\,dt$$

finding the area under the curve or quadrature

$$\frac{dy}{dt} = f(t, y)$$

numerical methods used to solve

Solving Single Differential Equations

Quadrature – Analytical function

quadrature.m

$$\frac{dy}{dt} = t \cdot \cos(t) \quad \Rightarrow \quad y = \int_0^{\pi/2} t \cdot \cos(t) \cdot dt$$

MATLAB **integral** function

integral(fun,tmin,tmax)

```
intfun = @(t) t.*cos(t);
y=integral(intfun,0,pi/2)
```

using an anonymous function here

```
>> testintegral

y =

    0.5708
```

Note the use of the dot (**.**) operator since **integral** may call **intfun** with multiple values.

Solving Single Differential Equations

Quadrature – Analytical function

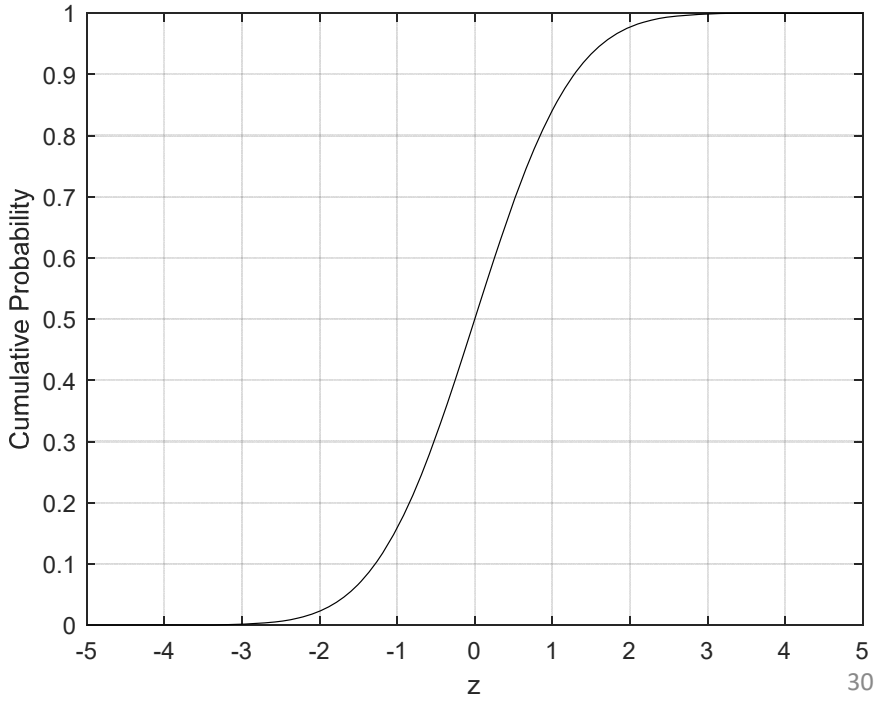
stdnormcumprob.m
stdnormdens.m

Standard normal distribution – cumulative probability

$$f(z)=\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2} \quad \Rightarrow \quad \frac{dP}{dz}=f(z) \quad \Rightarrow \quad P[-\infty \le z \le a]=\int_{-\infty}^{a} f(z)\,dz$$

```
function fz = stdnormdens(z)
fz = 1/sqrt(2*pi)*exp(-1/2*z.^2);
```

```
a = linspace(-5,5);
n = length(a);
for i = 1:n
    P(i) = integral(@stdnormdens,-Inf,a(i));
end
plot(a,P,'k-') ; grid
xlabel('z')
ylabel('Cumulative Probability')
```



Note: MATLAB has a **cdf** function in the Statistics toolbox.

Solving Single Differential Equations

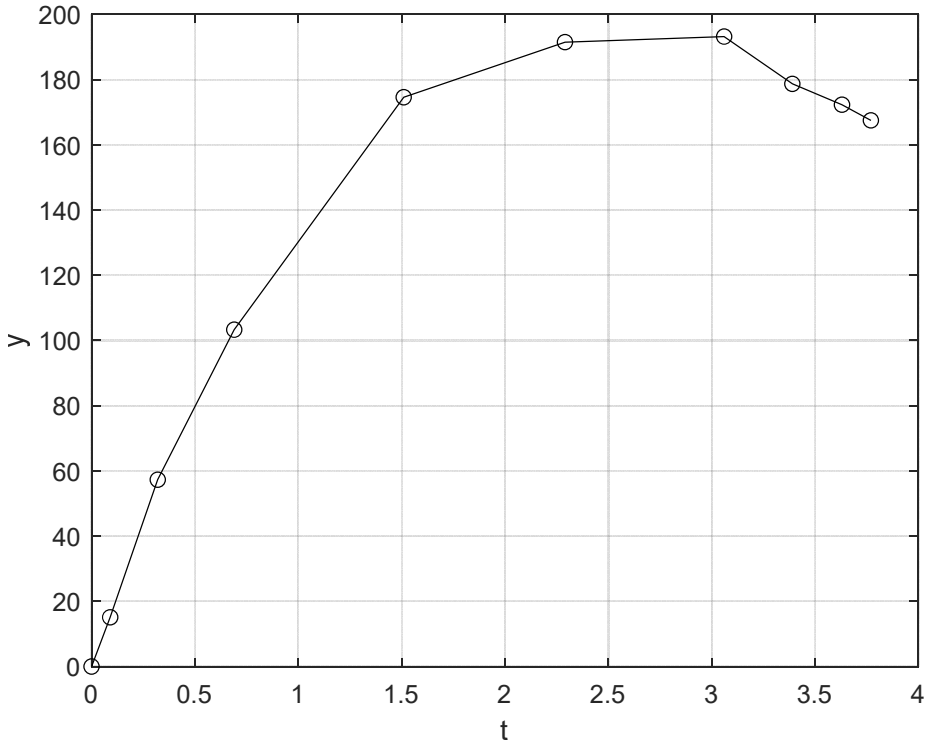
Quadrature – Data

MATLAB **trapz** function trapz(ydata,tdata) trapezoidal rule integration

```
t = [0,0.09,0.32,0.69,1.51,2.29,3.06,3.39,3.63,3.77];
y = [0,15.1,57.3,103.3,174.6,191.5,193.2,178.7,172.3,167.5];
plot(t,y,'ko-');grid
xlabel('t');ylabel('y')
inty = trapz(t,y)
```

```
>> testtrapz

inty =

   570.8135
```

testtrapz.m



Solving Single Differential Equations

Initial Value Problem

$$\frac{dy}{dt} = f(t, y) \qquad y(0) = y_0$$

Example

$$\frac{dy}{dt} = 5\left(y - t^2\right) \qquad y(0) = 0.08 \qquad 0 \le t \le 5$$

There is an analytical solution:

$$y = t^2 + 0.4t + 0.08$$

MATLAB has a family of functions for solving ODEs. Two commonly used functions are
ode45 for most ODEs
ode15s for "stiff" ODEs
A "stiff" ODE is one where the derivative changes dramatically over the solution domain.
For sets of ODEs, it is where one or more equations are dramatically "faster" than others.
ode15s is also required when solving systems which combine ODEs with nonlinear algebraic eqns.
See "Choose an ODE Solver" in Help.

Solving Single Differential Equations

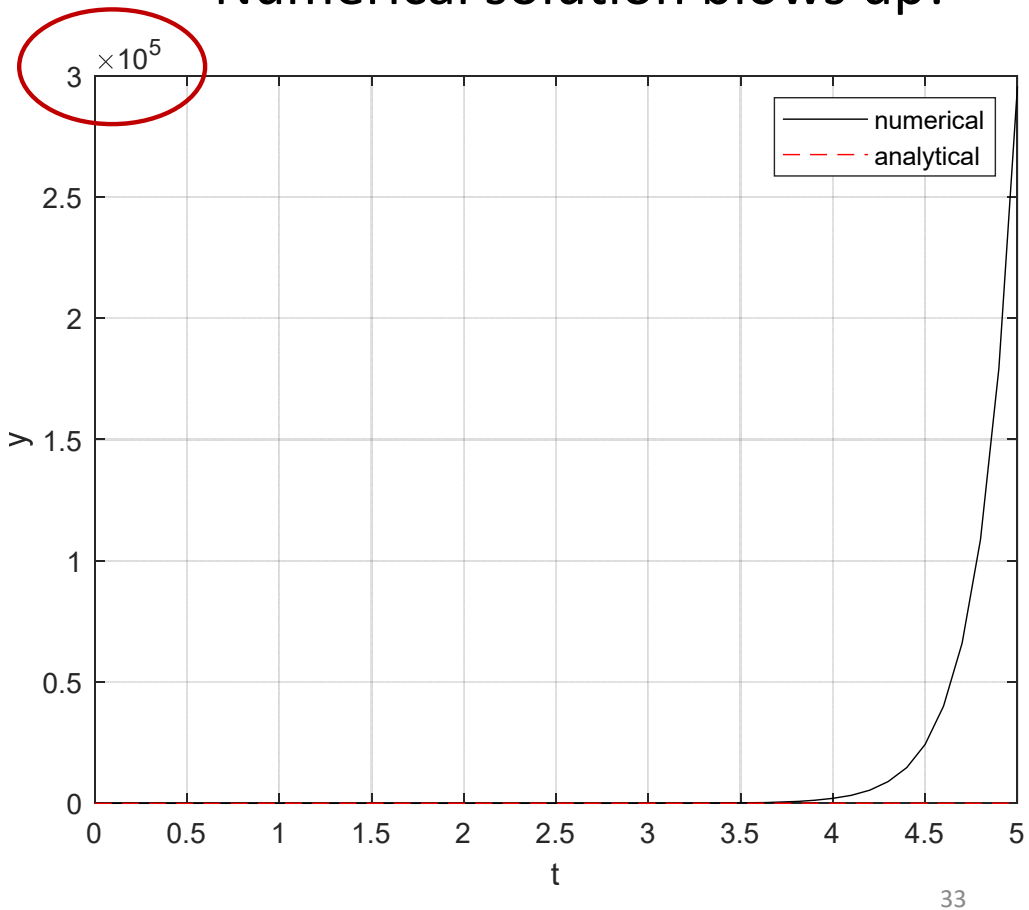
Initial Value Problem

Example using **ode45**

```
y0 = 0.08;
tout = 0:0.1:5;
anfun = @(t,y) 5*(y-t.^2);
[t,y] = ode45(anfun,tout,y0);
tanalyt = tout.^2+0.4*tout+0.08;
plot(t,y,'k-',tout,tanalyt,'r--');grid
xlabel('t')
ylabel('y')
legend('numerical','analytical')
```

parasite.m

Numerical solution blows up!



Solving Single Differential Equations

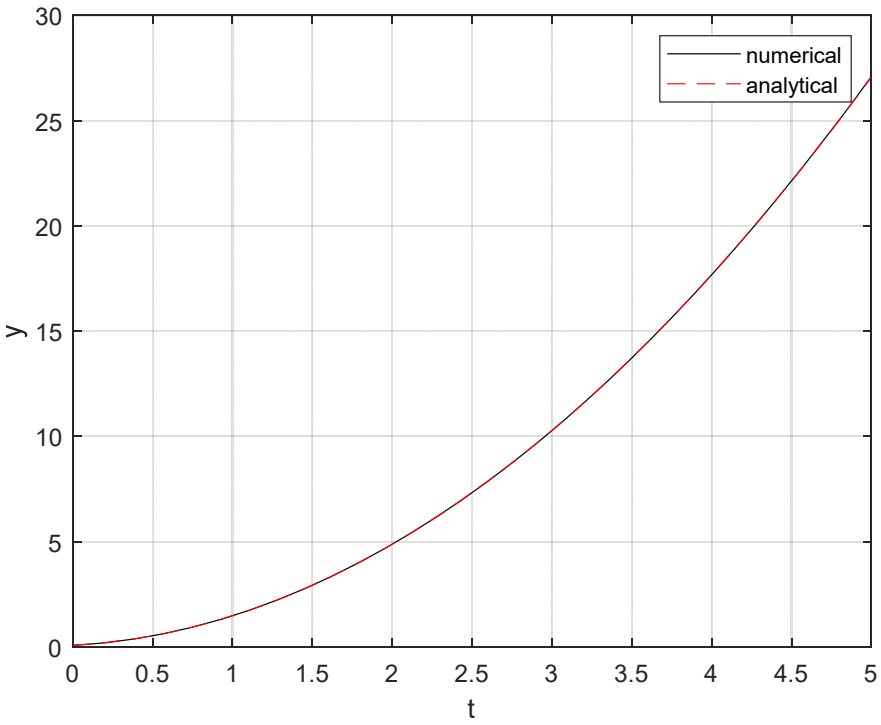
Initial Value Problem

Example using **ode45** with tightened tolerances

```
y0 = 0.08;
tout = 0:0.1:5;
anfun = @(t,y) 5*(y-t.^2);
options = odeset('RelTol',1.e-12,'AbsTol',1.e-12)
[t,y] = ode45(anfun,tout,y0,options);
tanalyt = tout.^2+0.4*tout+0.08;
plot(t,y,'k-',tout,tanalyt,'r--');grid
xlabel('t')
ylabel('y')
legend('numerical','analytical')
```

parasite1.m

numerical and analytical solutions coincide



Solving Single Differential Equations

Single Equation Example – Isothermal Batch Reactor $A + B \overset{k}{\Rightarrow} C$

Rate of disappearance of A: $\dfrac{dC_A}{dt} = -k \cdot C_A \cdot C_B$

Initial conditions: $C_A(0) = C_{A0} \quad C_B(0) = C_{B0} \quad C_C(0) = C_{C0}$

Basic data: $k = 14.7 \dfrac{1}{mol/L} \cdot \dfrac{1}{min}$

Initial conditions: $C_{A0} = 0.0209 \dfrac{mol}{L} \quad C_{B0} = C_{A0}/3 \quad C_{C0} = 0$

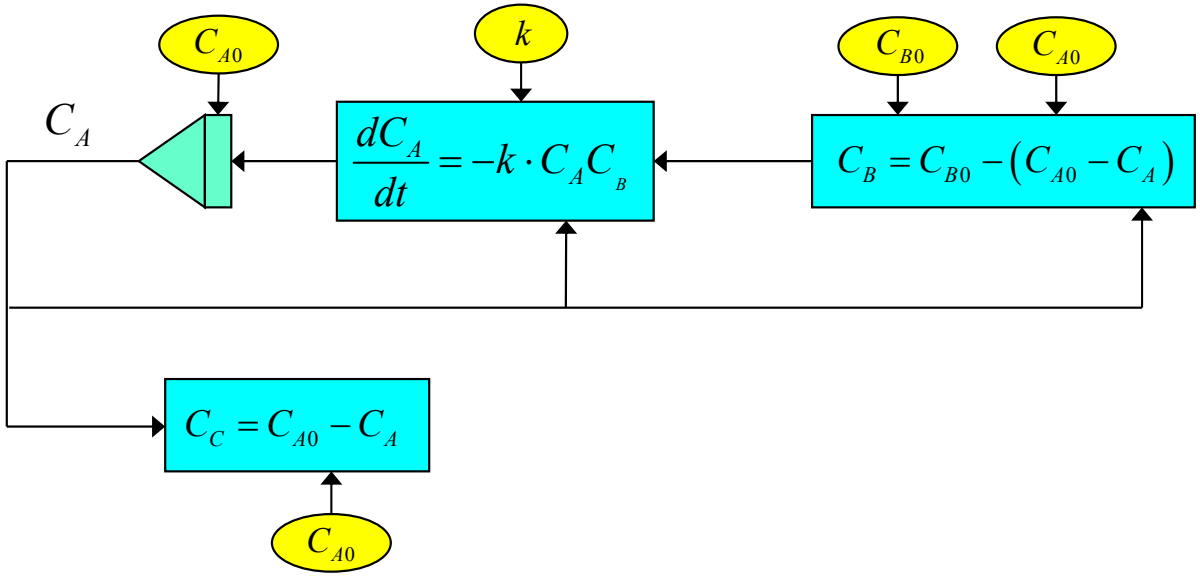
Stoichiometric relationships: $C_B(t) = C_{B0} - \left(C_{A0} - C_A(t)\right)$

$C_C(t) = C_{C0} + \left(C_{A0} - C_A(t)\right)$

Solving Single Differential Equations

Single Equation Example – Isothermal Batch Reactor

Information Flow Diagram



Integrator

Solving Single Differential Equations

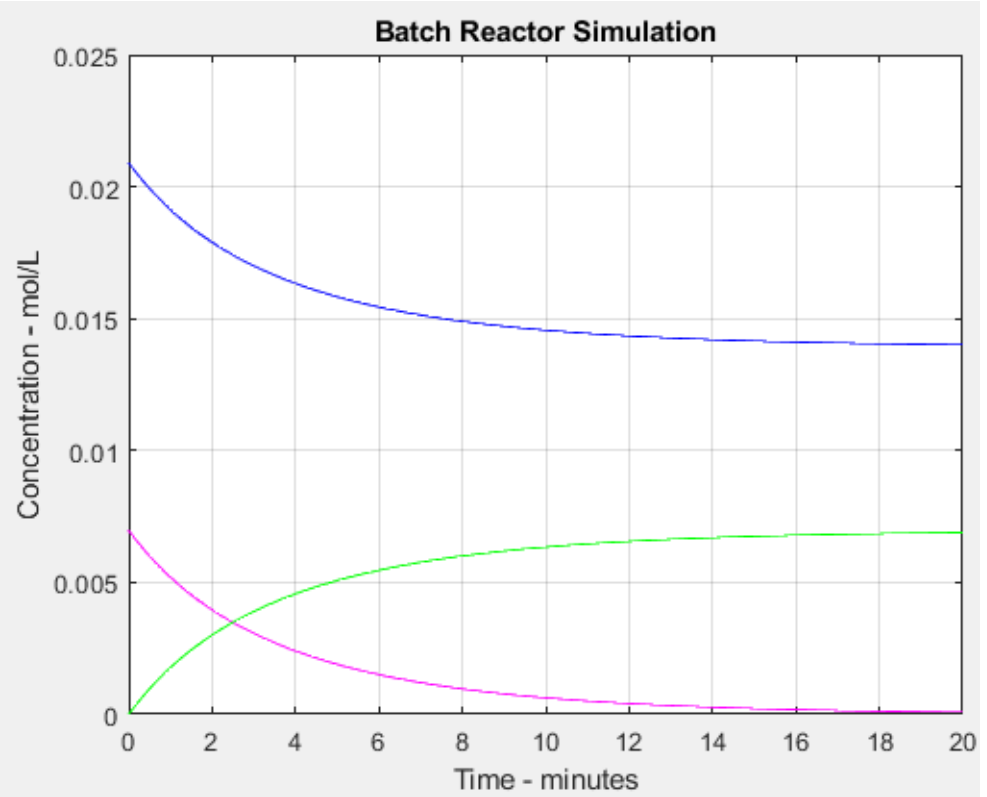
Single Equation Example – Isothermal Batch Reactor

batch.m*

```
% function to compute derivative
% for isothermal batch reactor
function dCa = batch(t,Ca,k,Ca0,Cb0)
Cb = Cb0 - (Ca0-Ca) ;
dCa = -k * Ca * Cb ;
```

SolveBatchReactor.m*

```
% solve differential equation
% for isothermal batch reactor
% using ode45
k = 14.7; % 1/(mol/L)/min
Ca0 = 0.0209; % mol/L
Cb0 = Ca0/3;
tspan = 0:0.1:20;
batchanon = @(t,Ca) batch(t,Ca,k,Ca0,Cb0);
[ t , Ca_soln ] = ode45(batchanon,tspan,Ca0);
Cb_soln = Cb0 - (Ca0-Ca_soln);
Cc_soln = Ca0-Ca_soln;
plot(t,Ca_soln,'b-',t,Cb_soln,'m-',t,Cc_soln,'g-')
grid
xlabel('Time - minutes')
ylabel('Concentration - mol/L')
title('Batch Reactor Simulation')
```



batch.m

SolveBatchReactor.m

Solving Single Differential Equations

Single Equation Example – Isothermal Batch Reactor

Matlab solution using the **ode15s** function

Differential Algebraic Equations (DAE) Approach

SolveBatchReactorDAE.m | batchDAE.m*

```
1     % function to compute derivative
2     % and algebraic relationships
3     % for isothermal batch reactor
4     % DAE format
5     function out = batchDAE(t,y,k,Ca0,Cb0)
6     Ca = y(1);
7     Cb = y(2);
8     Cc = y(3);
9     out(1) = -k * Ca * Cb ;
10    out(2) = Cb - (Cb0 - (Ca0-Ca)) ;
11    out(3) = Cc - (Ca0-Ca) ;
12    out = out';
```

two algebraic equations set equal to zero

batchDAE.m

Solving Single Differential Equations

Single Equation Example – Isothermal Batch Reactor

Matlab solution using the **ode15s** function

Differential Algebraic Equations (DAE) Approach

```
% solve differential equation
% for isothermal batch reactor
% using ode15s as a DAE system
k = 14.7 ; % 1/(mol/L)/min
Ca0 = 0.0209 ; % mol/L
Cb0 = Ca0/3 ;
Cc0 = 0
tspan = 0:0.1:20 ; % solution times
% set mass matrix
M = [ 1 0 0 ; % 1 on diagonal identifies ODE
      0 0 0 ; % 0 on diagonal identifies algebraic
      0 0 0 ];
y0 = [ Ca0 ; Cb0 ; Cc0 ]; % initial conditions
options = odeset('Mass',M); % set for DAE
% anonymous function to include extra arguments
batchanon = @(t,y) batchDAE(t,y,k,Ca0,Cb0);
% solve system with ode15s
[ t, ysoln ] = ode15s(batchanon,tspan,y0,options);
```

SolveBatchReactorDAE.m

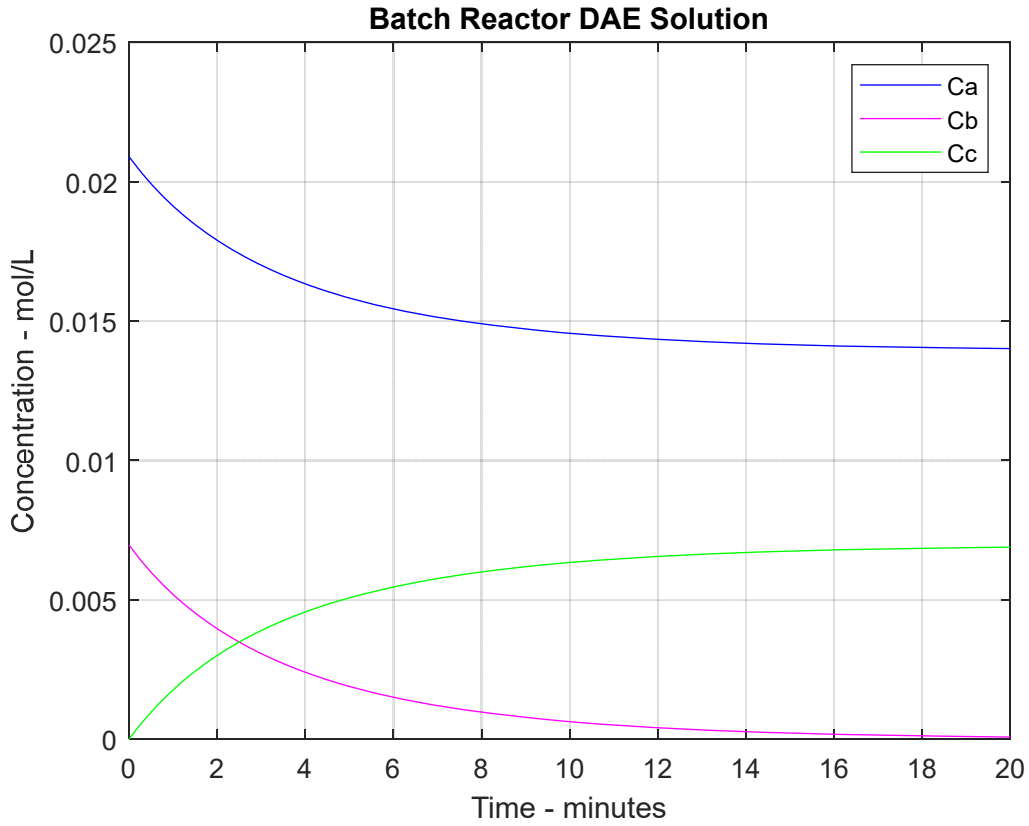
```
% unpack solution
Ca = ysoln(:,1); Cb = ysoln(:,2) ; Cc = ysoln(:,3);
% create plot
plot(t,Ca,'b',t,Cb,'m',t,Cc,'g')
grid
xlabel('Time - minutes')
ylabel('Concentration - mol/L')
title('Batch Reactor DAE Solution')
legend('Ca','Cb','Cc')
```

Solving Single Differential Equations

Single Equation Example – Isothermal Batch Reactor

Matlab solution using the **ode15s** function

Differential Algebraic Equations (DAE) Approach



Solving Multiple Differential Equations

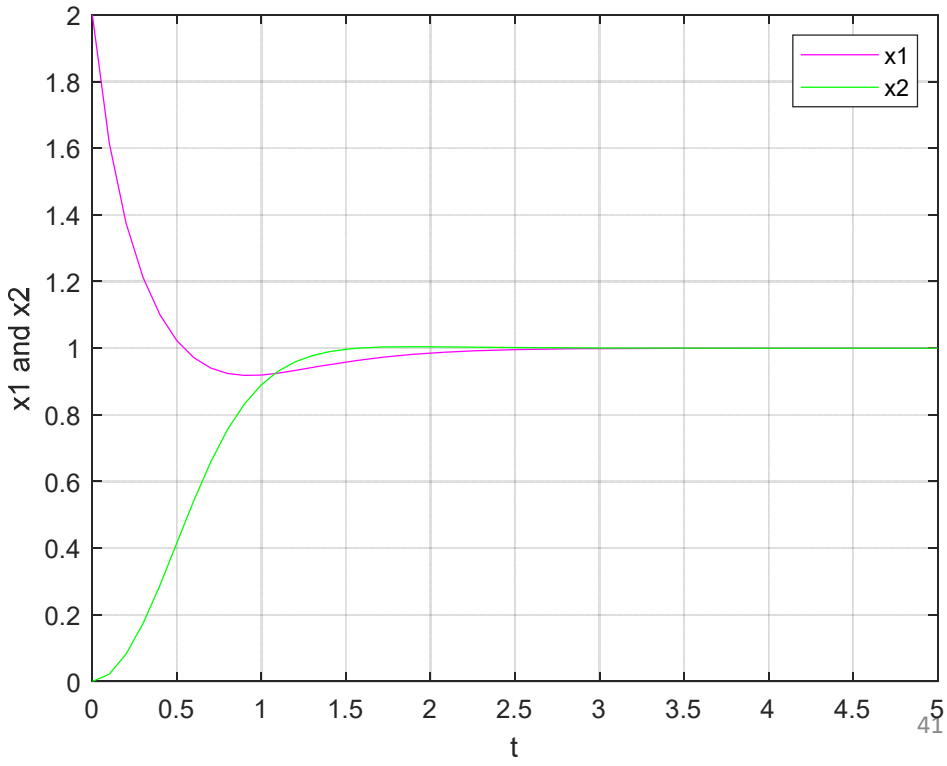
Example

$$\frac{dx_1}{dt} = -2x_1^2 + 2x_1 + x_2 - 1 \qquad x_1(0) = 2$$

$$\frac{dx_2}{dt} = -x_1 - 3x_2^2 + 2x_2 + 2 \qquad x_2(0) = 0$$

solvemultipleODEs.m
derivs.m

```
function dx = derivs(t,x)
dx(1) = -2*x(1).^2 + 2*x(1) + x(2) -1;
dx(2) = -x(1) - 3*x(2).^2 + 2*x(2) + 2;
dx = dx';

x0 = [2 0]';
tout = 0:0.1:5;
[t x] = ode45(@derivs,tout,x0);
plot(t,x(:,1),'m-',t,x(:,2),'g-') ; grid
xlabel('t')
ylabel('x1 and x2')
legend('x1','x2')
```



Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

$$A + B \xrightarrow{k_1} C + F$$
$$A + C \xrightarrow{k_2} D + F$$
$$A + D \xrightarrow{k_3} E + F$$

$$\frac{dA}{dt} = -k_1 AB - k_2 AC - k_3 AD \qquad A(0) = 0.0209\ \frac{mol}{L}$$

$$\frac{dB}{dt} = -k_1 AB \qquad B(0) = \frac{A(0)}{3}$$

$$\frac{dC}{dt} = k_1 AB - k_2 AC \qquad C(0) = 0$$

$$\frac{dD}{dt} = k_2 AC - k_3 AD \qquad D(0) = 0$$

$$k_1 = 14.7\ \frac{1}{mol/L} \cdot \frac{1}{min}$$

$$k_2 = 1.53\ \frac{1}{mol/L} \cdot \frac{1}{min}$$

$$k_3 = 0.294\ \frac{1}{mol/L} \cdot \frac{1}{min}$$

From stoichiometry: $E = \frac{A(0) - A - C - 2D}{3}$ and $F = A(0) - A$

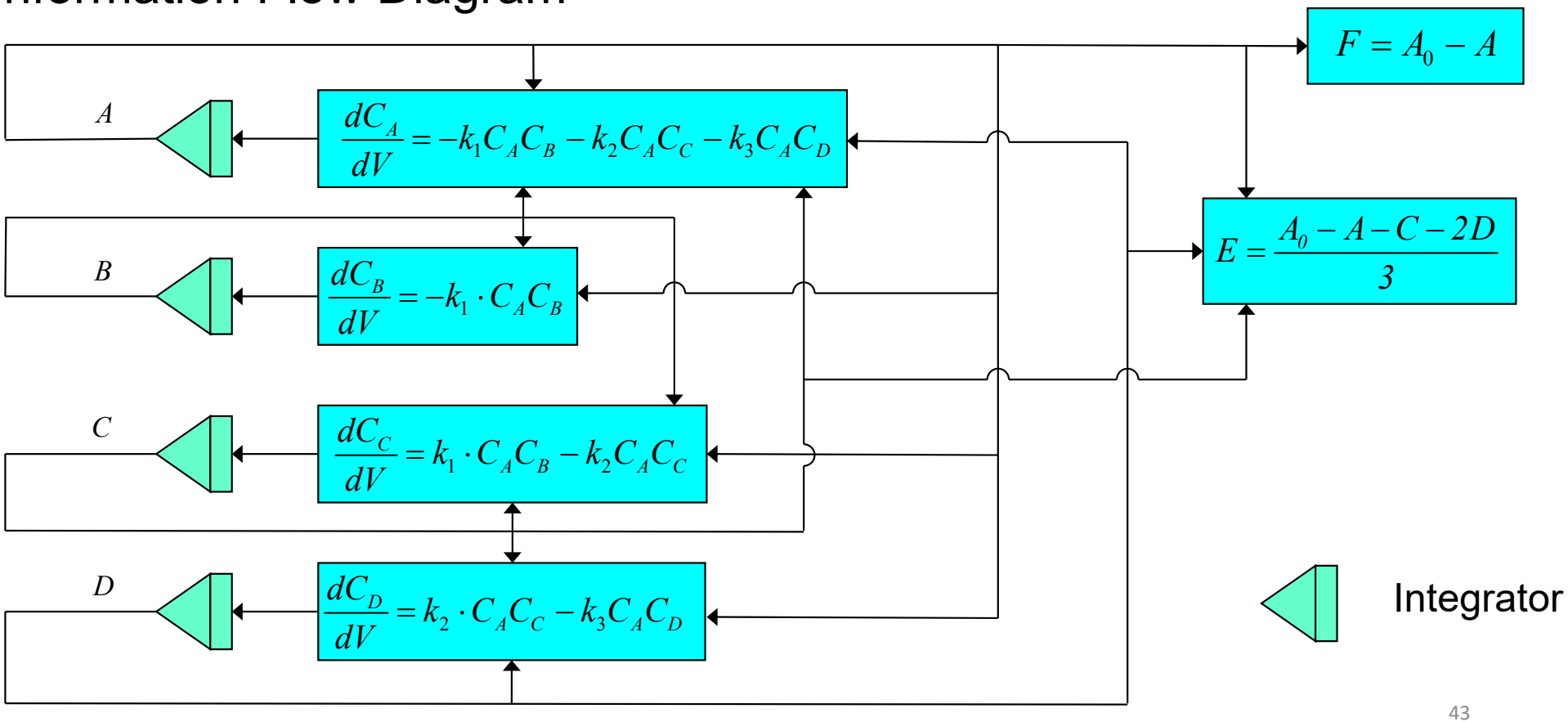
Svirbely, W.J., and J.A. Blauer, *The Kinetics of Three-step Competitive Consecutive Second-order Reactions*, **J. Amer. Chem. Soc.**, **83**, 4115, 1961.

Svirbely, W.J., and J.A. Blauer, *The Kinetics of the Alkaline Hydrolysis of 1,3,5,Tricarbomethoxybenzene*, **J. Amer. Chem. Soc.**, **83**, 4118, 1961.

Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

Information Flow Diagram



Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

```
% function to specify derivatives
% and algebraic equations
% for isothermal batch reactor
% with multiple reactions
function out = multibatch(t,y,k1,k2,k3,A0)
A = y(1);
B = y(2);
C = y(3);
D = y(4);
E = y(5);
F = y(6);
out(1) = -k1*A*B - k2*A*C - k3*A*D;
out(2) = -k1*A*B;
out(3) = k1*A*B - k2*A*C;
out(4) = k2*A*C - k3*A*D;
out(5) = E-(A0-A-C-2*D)/3;
out(6) = F-(A0-A);
out=out';
```

additional arguments

unpack **y** into familiar variables

4 ODEs
2 algebraic equations

multibatch.m

Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

MATLAB solution as a DAE system

```
% Multiple reactions simulation
% using ode15s and a DAE format
% rate constants
k1 = 14.7; % 1/(mol/L)/min
k2 = 1.53;
k3 = 0.294;
% initial conditions
A0 = 0.0209; % mol/L
B0 = A0/3;
C0 = 0;
D0 = 0;
E0 = 0;
F0 = 0;
y0 = [ A0 B0 C0 D0 E0 F0 ]';
% time span
tspan = 0:500;
```

```
% mass matrix
M = [ 1 0 0 0 0 0 ;
      0 1 0 0 0 0 ;
      0 0 1 0 0 0 ;
      0 0 0 1 0 0 ;
      0 0 0 0 0 0 ;
      0 0 0 0 0 0 ];
% set options for DAE
options = odeset('Mass',M);
% anonymous function for the DAE system
multibatchanon = @(t,y) multibatch(t,y,k1,k2,k3,A0);
% solve system
[ t,y ] = ode15s(multibatchanon,tspan,y0,options);
```

SolveMultiBatchDAE.m

Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

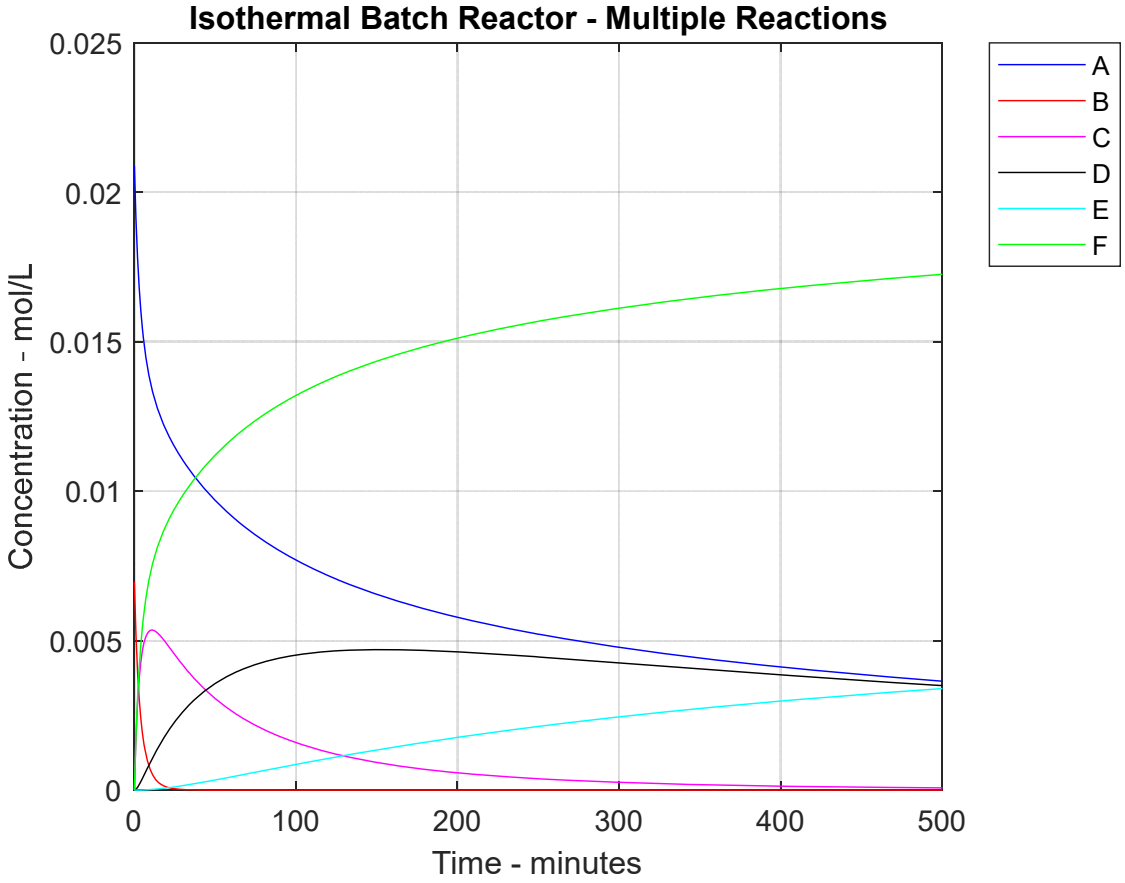
MATLAB solution as a DAE system

```
% unpack solution
A = y(:,1);
B = y(:,2);
C = y(:,3);
D = y(:,4);
E = y(:,5);
F = y(:,6);
% create plot
plot(t,A,'b-',t,B,'r-',t,C,'m-',t,D,'k-',t,E,'c-',t,F,'g-');
grid
axis([ 0 500 0 0.025 ]);
legend('A','B','C','D','E','F','Location','northeastoutside')
xlabel('Time - minutes')
ylabel('Concentration - mol/L')
title('Isothermal Batch Reactor - Multiple Reactions')
```

Solving Multiple Differential Equations

Multiple Equation Models – Isothermal Batch Reactor

MATLAB solution as a DAE system



Solving Ordinary Differential Equations

Second-order differential equation with split boundary conditions

$$\frac{d^2y}{dt^2} = \frac{1}{4}\frac{dy}{dt} + y \qquad y(0) = 5 \qquad y(10) = 8 \qquad 0 \le t \le 10$$

Decompose into two first-order ODEs

$$\frac{dy}{dt} = y_1 \qquad y(0) = 5 \qquad y(10) = 8$$

$$\frac{dy_1}{dt} = \frac{1}{4}y_1 + y$$

"Shooting" Strategy

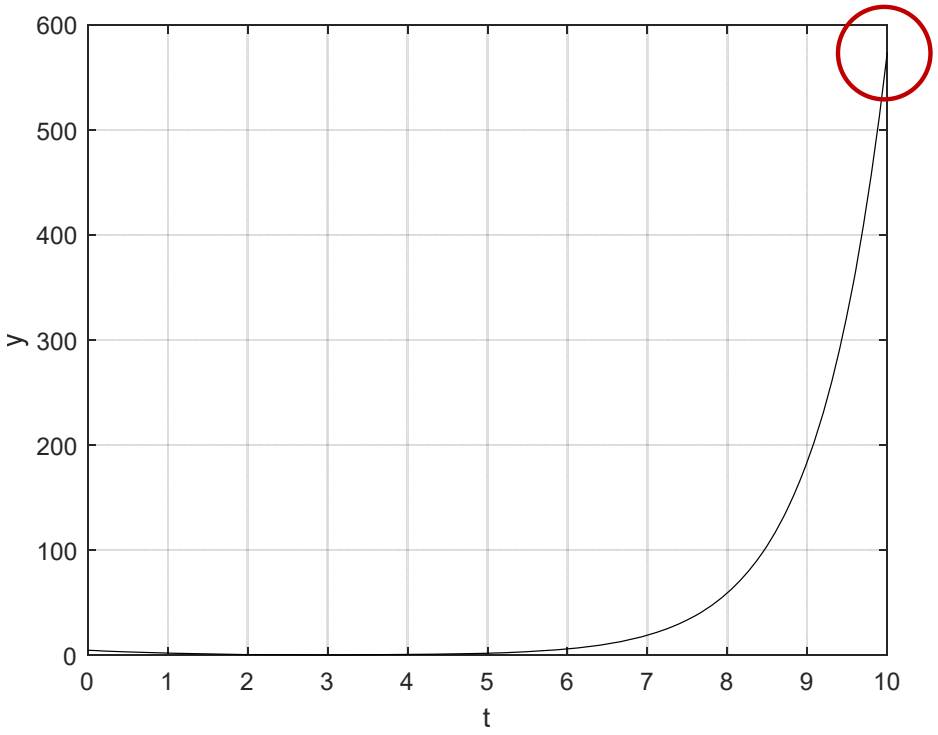
1. Estimate a value for y1 (dy/dt) at t = 0.
2. Solve the ODEs to t = 10
3. Check y(10) versus the required value, 8.
4. Adjust the y1(0) value and repeat steps 2 and 3 until the desired y(10)=8 value is obtained.

Solving Ordinary Differential Equations

Second-order differential equation with split boundary conditions

```
y0 = [ 5 -4.4 ]';
tspan = 0:0.1:10;
[ t y ] = ode45(@diffeqs,tspan,y0);
plot(t,y(:,1),'k-'); grid
xlabel('t')
ylabel('y')
```

```
function dy = diffeqs(t,y)
dy(1) = y(2);
dy(2) = y(2)/4 + y(1);
dy = dy';
```



Equations solved with an estimate for y1(0). Clearly doesn't meet the required final condition y(10)=8.

splitboundary.m
diffeqs.m

Solving Ordinary Differential Equations

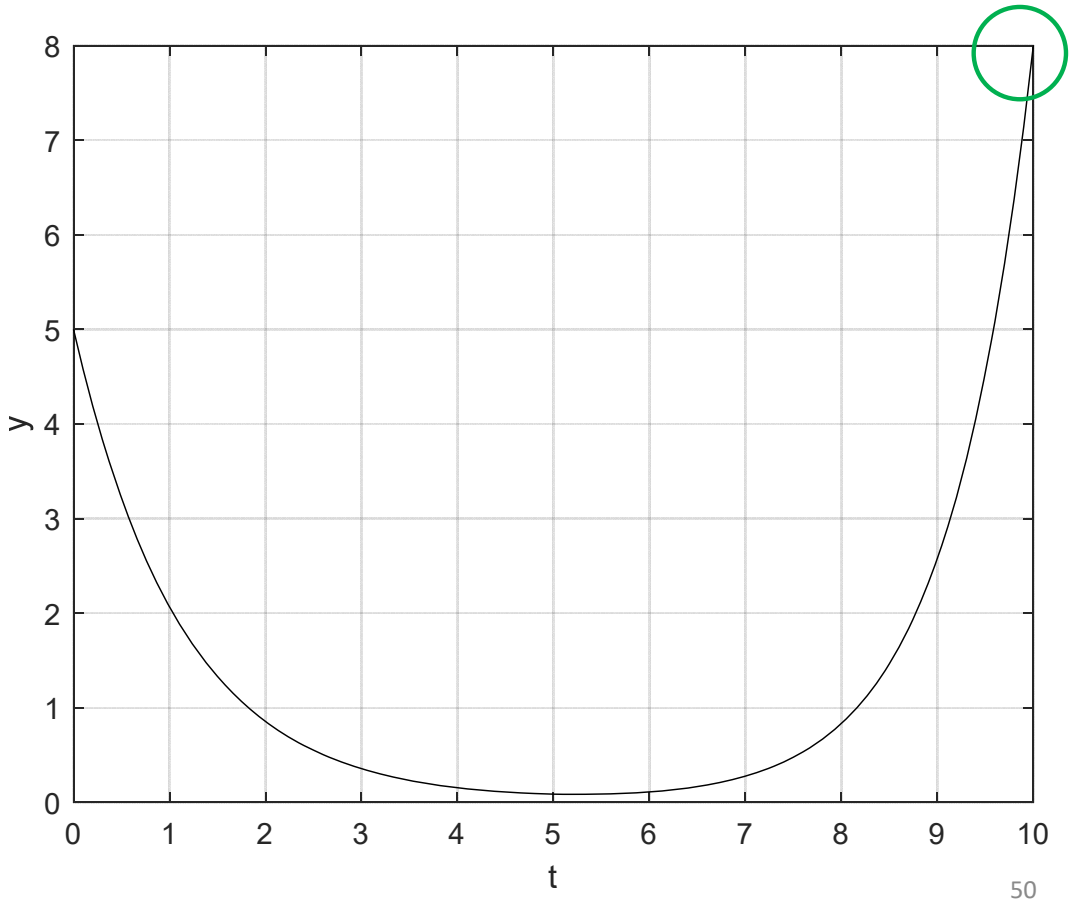
Second-order differential equation with split boundary conditions

Employ **fzero** to satisfy the final boundary condition

```
y10 = -4.4;
y10soln = fzero(@splitboundary1,y10)
y0 = [ 5 y10soln ]';
tspan = 0:0.1:10;
[ t y ] = ode45(@diffeqs,tspan,y0);
plot(t,y(:,1),'k-'); grid
xlabel('t')
ylabel('y')
```

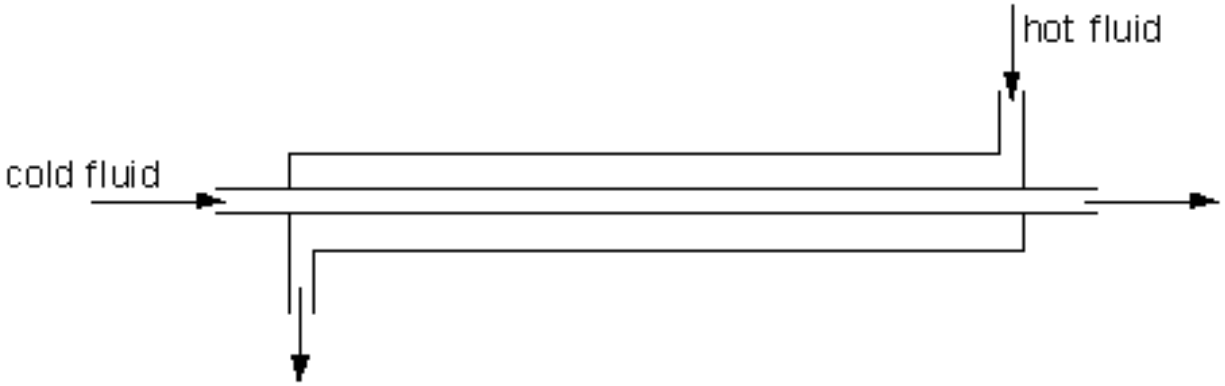
```
function ferr = solveODEs(y10)
y0 = [ 5 y10 ]';
tspan = [0 10];
[ t y ] = ode45(@diffeqs,tspan,y0);
ferr = y(end,1) - 8;
```

splitboundary1.m
solvesplitboundary.m



Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger



$$\frac{dT_c}{dz} = \frac{h_i A_i}{w_c C_c}\left(T_h - T_c\right) \qquad T_c\left(0\right) = T_{ci}$$

$$\frac{dT_h}{dz} = \frac{h_o A_o}{w_h C_h}\left(T_h - T_c\right) \qquad T_h\left(L\right) = T_{hi}$$

$$h_o = \frac{h_i \cdot D_i}{D_o}$$

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

z: distance down the heat exchanger from the cold fluid inlet (on the left)

L: length of the heat exchanger

T_c: temperature of the cold fluid, a function of z

T_{ci}: cold water inlet temperature, at $z=0$

T_{hi}: hot water inlet temperature, at $z=L$

T_h: temperature of the hot fluid, a function of z

w_c: mass flow rate of cold fluid

w_h: mass flow rate of hot fluid

C_c: heat capacity of cold fluid

C_h: heat capacity of hot fluid

A_i: inside area for heat transfer (cold fluid) per unit length

A_o: outside area for heat transfer (hot fluid) per unit length

h_i: inside heat transfer coefficient (cold fluid)

h_o: outside heat transfer coefficient (hot fluid)

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

$$\frac{dT_c}{dz} = \frac{h_i A_i}{w_c C_c}\left(T_h - T_c\right) \qquad T_c\left(0\right) = T_{ci}$$

$$\frac{dT_h}{dz} = \frac{h_o A_o}{w_h C_h}\left(T_h - T_c\right) \qquad T_h\left(L\right) = T_{hi}$$

The issue we have with solving these equations is that the cold stream boundary condition is at z = 0 and the hot stream boundary condition is at z = L, the other end of the heat exchanger. A practical way to handle this is to estimate the hot stream temperature at z = 0, proceed with the solution, and adjust that estimate later on to meet the condition at z = L.

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

Basic data and operating conditions

Outer tube
11 BWG
OD 2 in, ID 1.76 in

Inner tube
11 BWG
OD 1 in, ID 0.76 in

Length 5 m

Inlet temperatures
Hot stream 50 °C
Cold stream 10 °C

Fluid density (H_2O)
988 kg/m^3

Heat capacity (H_2O)
4187 J/(kg·°C)

Hot stream flow rate 1 L/s
Cold stream 0.3 L/s

Heat transfer coefficient
h_i = 14,000 W/(m^2·°C)

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

```
% tube-in-tube countercurrent heat exchanger
% basic data
% tubes
doin = 1 ; % in
do = doin * 0.0254 ; % m
Ao = pi*do ; % m2/m
diin = 0.76 ; % in
di = diin * 0.0254 ; % m
Ai = pi*di ; % m2/m
L = 5 ; % m
% fluid properties
den = 998 ; % kg/m3
cP = 4187 ; % J/(kg*degC)
% heat transfer coefficients
hi = 14000 ; % W/(m2*degC)
ho = hi*di/do ; % W/(m2*degC)
% flow rates
qcL = 0.3 ; % L/s
qc = qcL/1000 ; % m3/s
wc = qc*den ; % kg/s
qhL = 1 ; % L/s
qh = qhL/1000 ; % m3/s
wh = qh*den ; % kg/s
% inlet temperatures
Tci = 10 ; % degC
Thi = 50 ; % degC
% z values
zspan = linspace(0,L) ;
% estimate hot outlet temperature
Tho = 40 ; %degC
```

tube_in_tube_heat_exchanger.m

Solve the model first with an estimate for the hot stream outlet temperature.

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

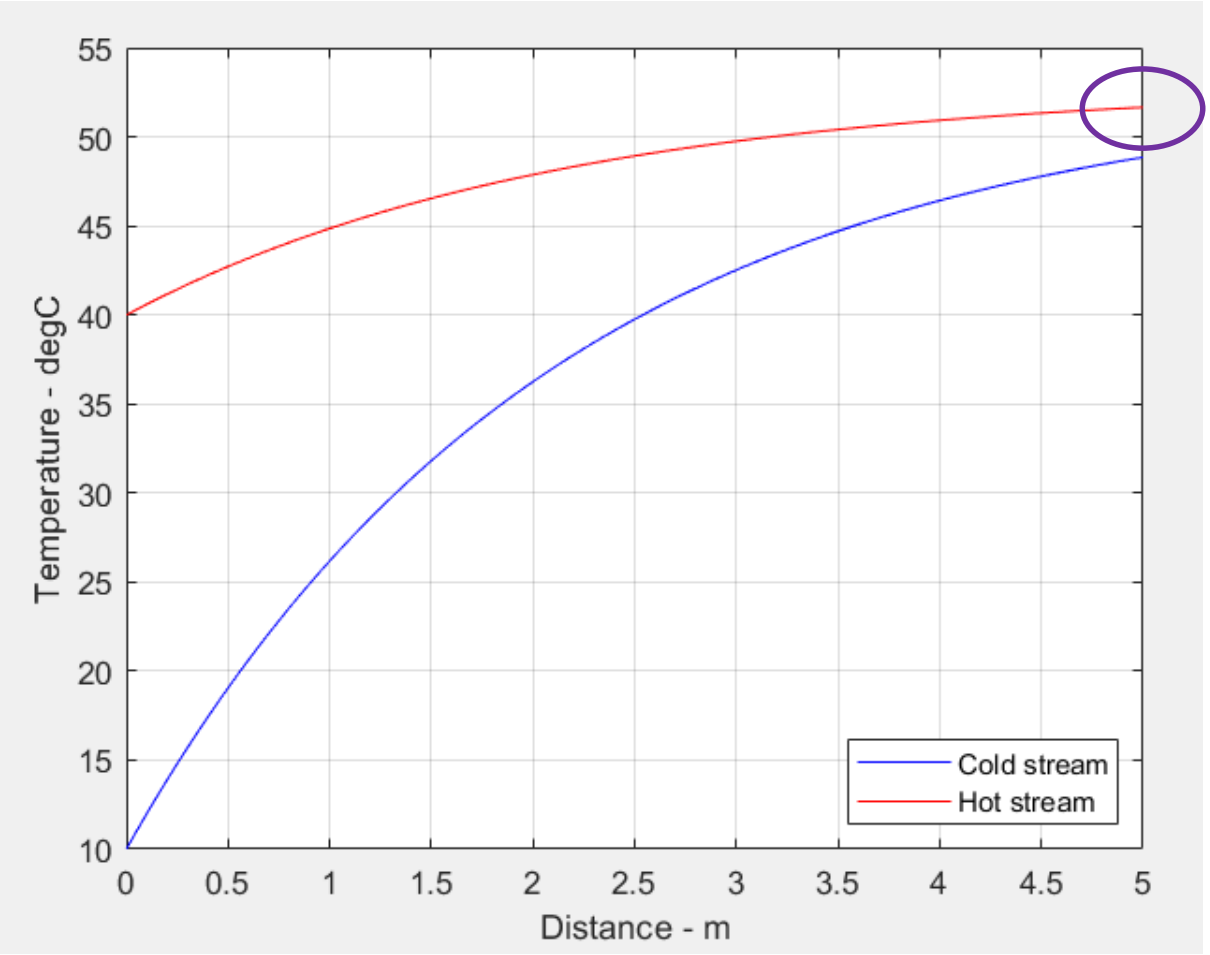
```
31    % initial conditions
32    T0 = [ Tci Tho ]';
33    % anonymous function
34    htexranon = @(z,T) htexr(z,T,hi,ho,Ai,Ao,wc,wh,cP);
35    % solve ODEs
36    [ zsoln, Tsoln ] = ode45(htexranon,zspan,T0);
37    % unpack temperatures
38    Tcsoln = Tsoln(:,1);
39    Thsoln = Tsoln(:,2);
40    % create plot
41    plot(zsoln,Tcsoln,'b-',zsoln,Thsoln,'r-')
42    grid
43    xlabel('Distance - m')
44    ylabel('Temperature - degC')
45    legend('Cold stream','Hot stream','Location','southeast')
```

```
1   function dTout = htexr(z,T,hi,ho,Ai,Ao,wc,wh,cP)
2   Tc = T(1);
3   Th = T(2);
4   dTc = hi*Ai/wc/cP*(Th-Tc);
5   dTh = ho*Ao/wh/cP*(Th-Tc);
6   dTout = [ dTc dTh]';
```

htexr.m

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger



50 °C not quite met

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

Use the **fzero** function to adjust the hot stream outlet temperature until the inlet hot stream condition is met.

```
% tube-in-tube countercurrent heat exchanger
% basic data
% tubes
doin = 1 ; % in
do = doin * 0.0254 ; % m
Ao = pi*do ; % m2/m
diin = 0.76 ; % in
di = diin * 0.0254 ; % m
Ai = pi*di ; % m2/m
L = 5 ; % m
% fluid properties
den = 998 ; % kg/m3
cP = 4187 ; % J/(kg*degC)
% heat transfer coefficients
hi = 14000 ; % W/(m2*degC)
ho = hi*di/do ; % W/(m2*degC)
```

tube_in_tube_heat_exchanger_fzero.m

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

```
17  % flow rates
18  qcL = 0.3 ; % L/s
19  qc = qcL/1000 ; % m3/s
20  wc = qc*den ; % kg/s
21  qhL = 1 ; % L/s
22  qh = qhL/1000 ; % m3/s
23  wh = qh*den ; % kg/s
24  % inlet temperatures
25  Tci = 10 ; % degC
26  Thi = 50 ; % degC
27  % initial estimate
28  Tho = 40 ; % degC
29  % anonymous function
30  solveODEsanon = @(Tho) solveODEs(Tho,hi,ho,Ai,Ao,L,wc,wh,cP,Tci,Thi);
31  % call fzero
32  Tho = fzero(solveODEsanon,Tho)
```

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

```
% solve model again
% z values
zspan = linspace(0,L) ;
% initial conditions
T0 = [ Tci Tho ]';
% anonymous function
htexranon = @(z,T) htexr(z,T,hi,ho,Ai,Ao,wc,wh,cP);
% solve ODEs
[ zsoln, Tsoln ] = ode45(htexranon,zspan,T0);
% unpack temperatures
Tcsoln = Tsoln(:,1);
Thsoln = Tsoln(:,2);
% create plot
plot(zsoln,Tcsoln,'b-',zsoln,Thsoln,'r-')
grid
xlabel('Distance - m')
ylabel('Temperature - degC')
legend('Cold stream','Hot stream','Location','southeast')
```

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger

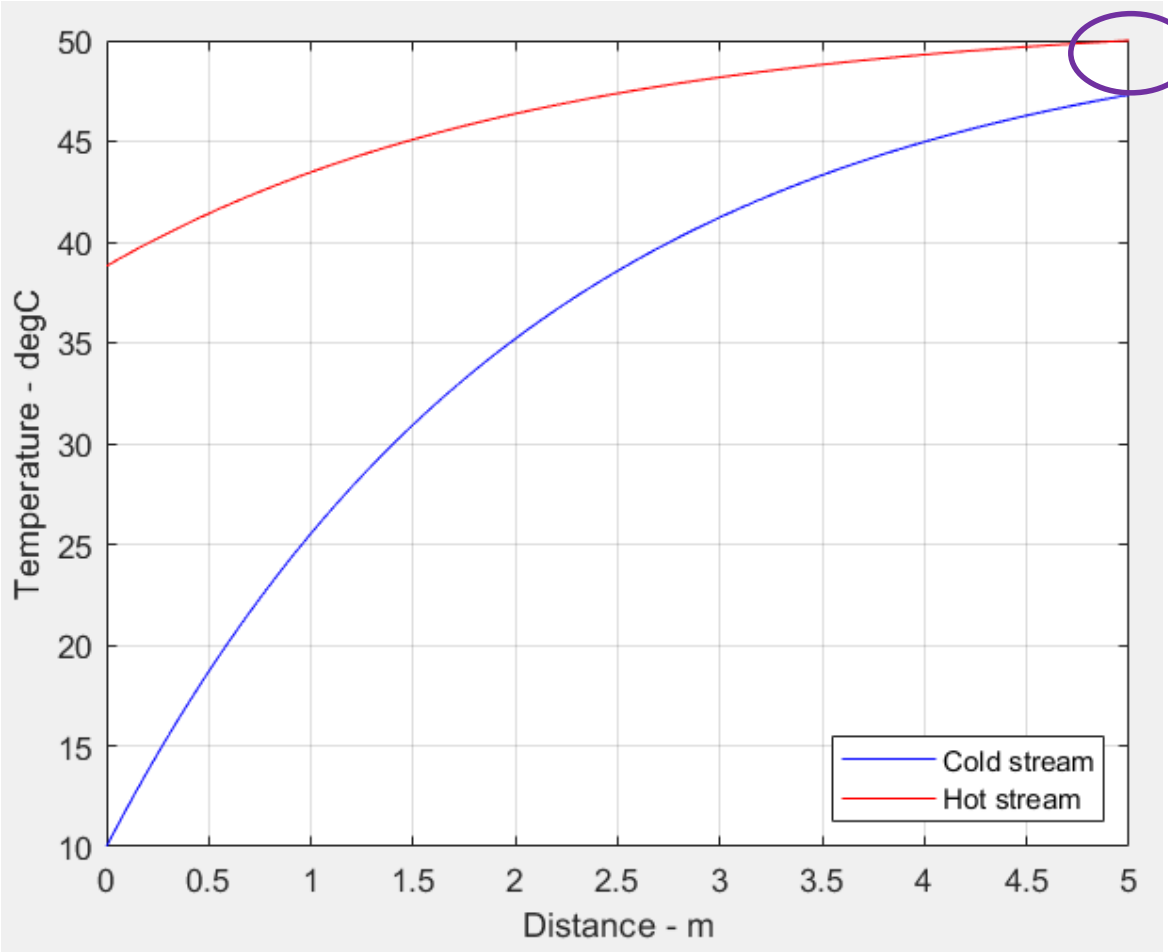
```
1  function ferr = solveODEs(Tho,hi,ho,Ai,Ao,L,wc,wh,cP,Tci,Thi)
2  % z values
3  zspan = linspace(0,L) ;
4  % initial conditions
5  T0 = [ Tci Tho ]';
6  % anonymous function
7  htexranon = @(z,T) htexr(z,T,hi,ho,Ai,Ao,wc,wh,cP);
8  % solve ODEs
9  [ zsoln, Tsoln ] = ode45(htexranon,zspan,T0);
10 % compute error as computed hot stream inlet
11 % temperature minus the specified value
12 ferr = Tsoln(end,2) - Thi;
```

fzero adjusts Tho until Thi computed meets the spec, that is, ferr = 0

SolveODEs.m

Solving Ordinary Differential Equations

Example: tube-in-tube, countercurrent heat exchanger



50 °C met

```
>> tube_in_tube_heat_exchanger_fzero

Tho =

   38.8075
```

Optimization

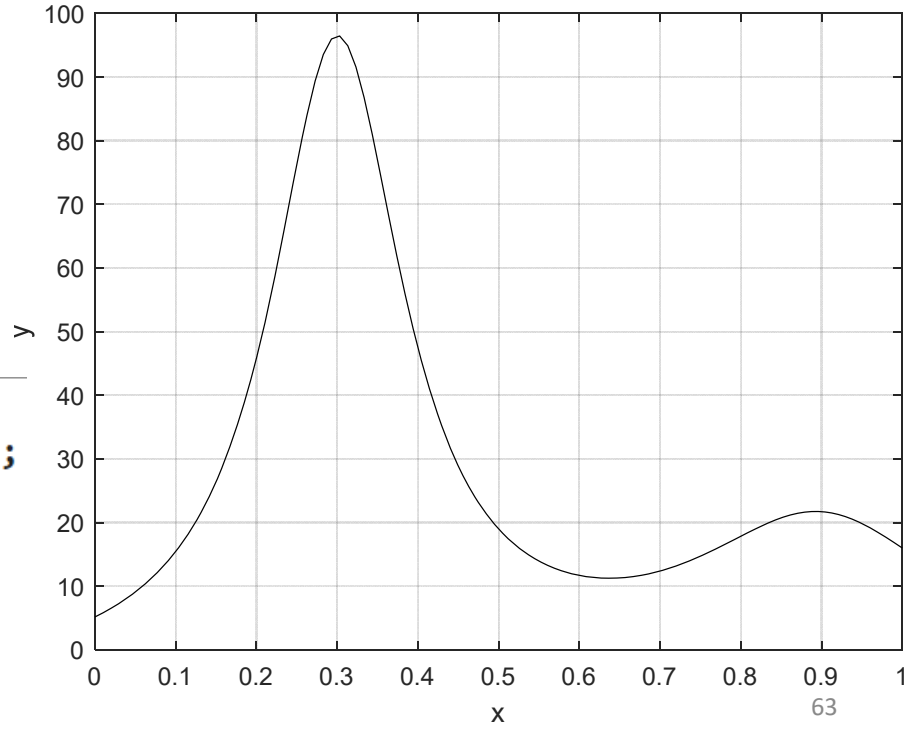
Finding a maximum or minimum of a function with a single adjustable variable

Example $f(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$

```
x = linspace(0,1);
y = humps(x);
plot(x,y,'k-');grid
xlabel('x')
ylabel('y')
```

```
function y = humps(x)
y = 1 ./((x-0.3).^2+0.01) + 1 ./((x-0.9).^2+0.04) -6;
```

humps.m
plothumps.m



Optimization

Finding a maximum or minimum of a function with a single adjustable variable

Using MATLAB function **fminbnd**

xmin = fminbnd(fun,x1,x2)

```
x1 = 0.;
x2 = 1.;
xmin = fminbnd(@humps,x1,x2)
```

```
>> findmin

xmin =

    0.6370
```

findmin.m
humps1.m
findmax.m

Find a maximum using - fun

```
x1 = 0.;
x2 = 1.;
xmax = fminbnd(@humps1,x1,x2)
```

```
function y = humps1(x)
y = - ( 1 ./((x-0.3).^2+0.01) + 1 ./((x-0.9).^2+0.04) -6 );
```

```
>> findmax

xmax =

    0.3004
```

Optimization

Finding a maximum or minimum of a function with multiple adjustable variables and one or more constraints

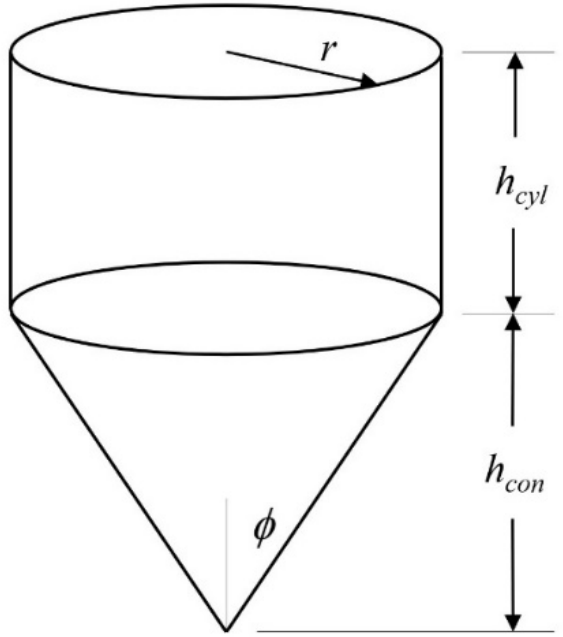
Example:
Optimal grain bin design

Minimize surface area not including the top

Constraints

$$V = V_{cyl} + V_{con} = 10 \text{ m}^3$$

$$\phi_{\max} = 20.4°$$



$$V_{cyl} = \pi r^2 h_{cyl}$$

$$S_{cyl} = 2\pi r h_{cyl}$$

$$V_{con} = \frac{1}{3}\pi r^2 h_{con}$$

$$S_{con} = \pi r \sqrt{r^2 + h_{con}^2}$$

$$\phi = \tan^{-1}\left(\frac{r}{h_{con}}\right)$$

Optimization

Finding a maximum or minimum of a function with multiple adjustable variables and one or more constraints

Using MATLAB function **fmincon**

xmin = fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options)

fun	function that returns value of performance criterion
xmin, x0	vectors of initial variables
A, b	linear inequality constraints, $A \cdot x \leq b$
Aeq, beq	linear equality constraints, $Aeq \cdot x = beq$
lb, ub	bounds on x
nonlcon	nonlinear constraints
options	various options

Optimization

Example:
Optimal grain bin design

```
function area = S(x)
r = x(1);
hcyl = x(2);
hcon = x(3);
Scyl = 2*pi*r*hcyl;
Scon = pi*r*sqrt(r^2+hcon^2);
area = Scyl+Scon;
```

```
function vol = V(x)
r = x(1);
hcyl = x(2);
hcon = x(3);
Vcyl = pi*r^2*hcyl;
Vcon = pi*r^2*hcon/3;
vol = Vcyl + Vcon;
```

```
function [c,ceq] = bincon(x)
r = x(1);
hcyl = x(2);
hcon = x(3);
angdeg = 20.4;
angrad = angdeg/180*pi;
c = atan(r/hcon)-angrad;
volc = 10;
ceq = V(x)-volc;
```

S.m
V.m
bincon.m

Optimization

Example:
Optimal grain bin design

```
clc; clear
x0 = [ 1 1 1]';
A = []; b = []; Aeq = []; beq = []; lb = []; ub = [];
xmin = fmincon(@S,x0,A,b,Aeq,beq,lb,ub,@bincon);
r = xmin(1);
hcyl = xmin(2);
hcon = xmin(3);
fprintf('\nradius = %4.2f m\n',r);
fprintf('cylinder height = %4.2f m\n',hcyl);
fprintf('cone height = %4.2f m\n',hcon);
phi = atan(r/hcon);
phid = rad2deg(phi);
fprintf('cone angle = %4.1f deg\n',phid);
fprintf('surface area = %6.1f m2\n',S(xmin));
fprintf('volume = %6.1f m3\n',V(xmin));
```

```
radius = 1.44 m
cylinder height = 0.26 m
cone height = 3.86 m
cone angle = 20.4 deg
surface area =   20.9 m2
volume =   10.0 m3
```

optbin.m

Curve-Fitting

Polynomial regression

coeff = polyfit(x,y,order)

y = polyval(x,coeff)

Example:

Density of Methanol-Water Solutions at 20 degC

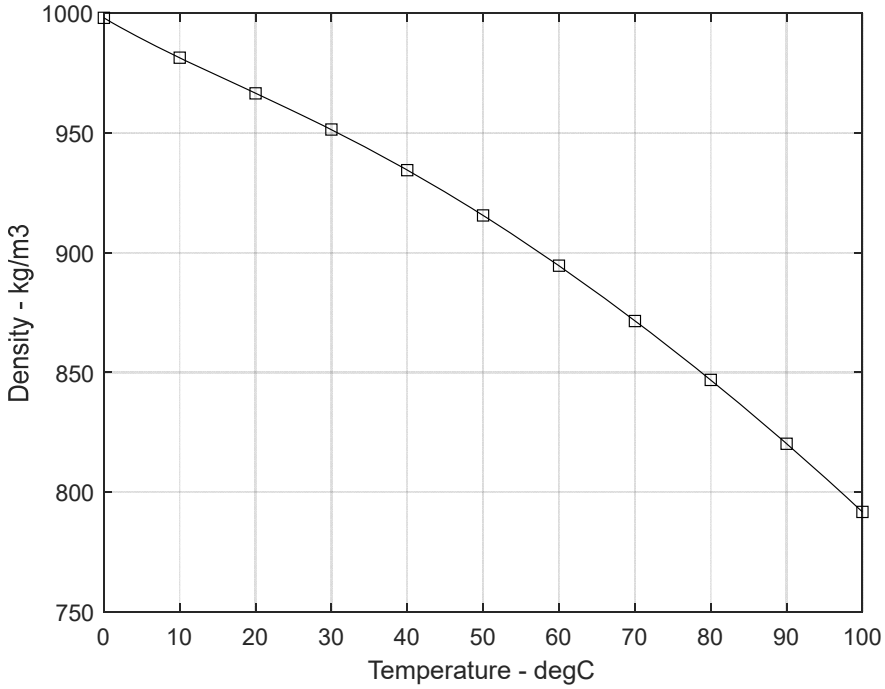
Wt% Methanol	Density (kg/m^3)
0	998.2
10	981.5
20	966.6
30	951.5
40	934.5
50	915.6
60	894.6
70	871.5
80	846.9
90	820.2
100	791.7

DensityModel.m

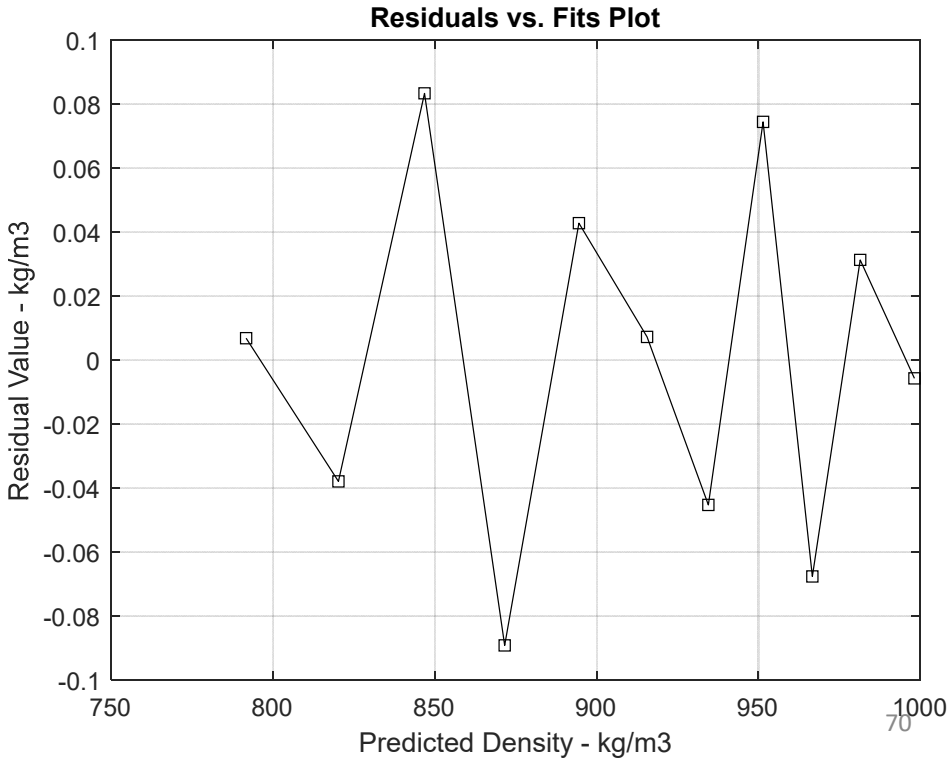
```
clc;clear
format short e
wp = 0:10:100; % wt% methanol
Den = [998.2,981.5,966.6,951.5,934.5,915.6,894.6 ...
      ,871.5,846.9,820.2,791.7];
coeff = polyfit(wp,Den,6)
wpP = linspace(0,100);
DenP = polyval(coeff,wpP);
plot(wp,Den,'sk',wpP,DenP,'k-')
grid
xlabel('Temperature - degC')
ylabel('Density - kg/m3')
DenP1 = polyval(coeff,wp);
resid = Den - DenP1;
figure % new figure
plot(DenP1,resid,'sk-')
grid
xlabel('Predicted Density - kg/m3')
ylabel('Residual Value - kg/m3')
title('Residuals vs. Fits Plot')
```

Curve-Fitting

Polynomial regression




```
coeff =

  Columns 1 through 4

   1.0784e-10  -5.0943e-08   9.3103e-06  -8.3716e-04

  Columns 5 through 7

   2.9008e-02  -1.8889e+00   9.9821e+02
```



Curve-Fitting

Multilinear regression

Model $$y = \beta_0 + \beta_1 f_1\left(x_j, j = 1,\ldots,m\right) + \beta_2 f_2\left(x_j, j = 1,\ldots,m\right) + \cdots + \beta_k f_k\left(x_j, j = 1,\ldots,m\right)$$

measurement responses

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

realization matrix of input levels

$$\mathbf{X} = \begin{bmatrix} 1 & f_{11} & f_{21} & \cdots & f_{m1} \\ 1 & f_{12} & f_{22} & \cdots & f_{m2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & f_{1n} & f_{2n} & \cdots & f_{mn} \end{bmatrix}$$

intercept

parameter estimates

$$\hat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_k \end{bmatrix}$$

model predictions

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

residuals

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \qquad \mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

sum of squares criterion

$$V = \sum_{i=1}^{n} e_i^2 = \mathbf{e}^T \cdot \mathbf{e}$$

Dataset

$$\left\{y_i, x_{1i}, \ldots, x_{mi}, i = 1,\ldots,n\right\}$$

Minimize V by choice of $\hat{\boldsymbol{\beta}}$ via calculus

Normal equations

$$\left(\mathbf{X}^T\mathbf{X}\right)\mathbf{b} = \mathbf{X}^T\mathbf{y}$$

Fitted model parameters

$$\mathbf{b} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Curve-Fitting

Multilinear regression

Example

Density of NaCl Aqueous Solutions

		Temperature			
		0 °C	10 °C	25 °C	40 °C
Wt % NaCl	1	1.00747	1.00707	1.00409	0.99908
	2	1.01509	1.01442	1.01112	1.00593
	4	1.03038	1.02920	1.02530	1.01977
	8	1.06121	1.05907	1.05412	1.04798
	12	1.09244	1.08946	1.08365	1.07699
	16	1.12419	1.12056	1.11401	1.10688
	20	1.15663	1.15254	1.14533	1.13774
	24	1.18999	1.18557	1.17776	1.16971
	26	1.20709	1.20254	1.19443	1.18614

from *Perry's Chemical Engineer's Handbook*, Green and Southard, Ed., 9th Ed., p. 2-103.

Model

$$\rho = \beta_0 + \beta_1 w + \beta_2 T + \beta_3 w^2 + \beta_4 T^2 + \beta_5 wT$$

Curve-Fitting

Multilinear regression using vector-matrix calculations

NaClDensityRegressionMatrixCalcs.m

```
clc;clear
format short e
w = [ 1 2 4 8 12 16 20 24 26 ];
T = [ 0 10 25 40 ];
rho = [ 1.00747, 1.00707, 1.00409, 0.99908 ; ...
        1.01509, 1.01442, 1.01112, 1.00593 ; ...
        1.03038, 1.02920, 1.02530, 1.01977 ; ...
        1.06121, 1.05907, 1.05412, 1.04798 ; ...
        1.09244, 1.08946, 1.08365, 1.07699 ; ...
        1.12419, 1.12056, 1.11401, 1.10688 ; ...
        1.15663, 1.15254, 1.14533, 1.13774 ; ...
        1.18999, 1.18577, 1.17776, 1.16971 ; ...
        1.20709, 1.20254, 1.19443, 1.18614 ] ;
n1 = length(w);
n2 = length(T);
for i = 1:n1 % create stacked vectors of independent variables
    for j = 1:n2
        wn((i-1)*n2+j) = w(i);
        Tn((i-1)*n2+j) = T(j);
    end
end
```

Curve-Fitting

Multilinear regression using vector-matrix calculations

```
wn = wn';
Tn = Tn';
X = [ ones(n1*n2,1) wn Tn wn.^2 Tn.^2 wn.*Tn];  % create the X matrix
y = reshape(rho',[n1*n2,1]);  % create the y vector
A = X'*X;
rs = X'*y;
b = A\rs  % model coefficients
rhop = X*b;  % predicted densities
resid = y - rhop;  % residuals
plot(y,rhop,'ok')
grid
hold on
plot(rhop,rhop,'k--')
xlabel('Density Data - g/cm3')
ylabel('Predicted Density - g/cm3')
figure
plot(rhop,resid,'k.')
grid
xlabel('Predicted Density')
ylabel('Residual')
title('Residuals vs. Fits')
```

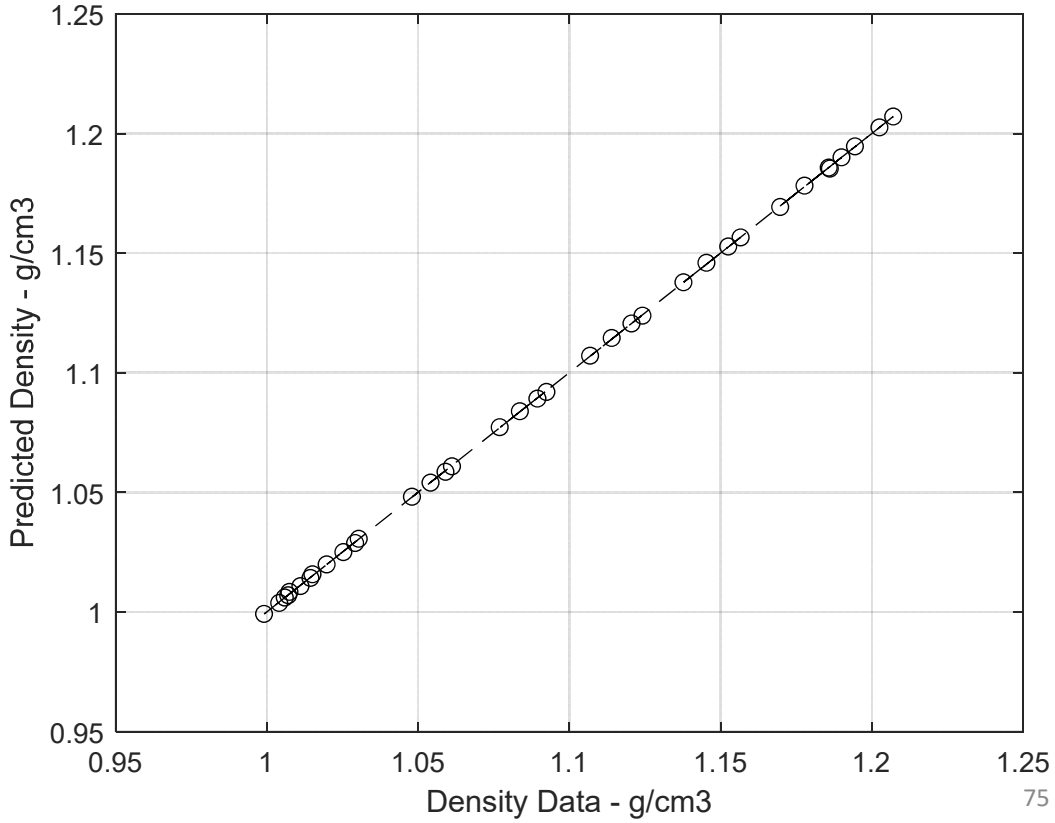
Curve-Fitting

Multilinear regression using vector-matrix calculations

Model coefficients

```
>> NaClDensityRegressionMatrixCalcs

b =

   1.0011e+00
   7.2739e-03
  -9.6642e-05
   2.5015e-05
  -3.0211e-06
  -1.2572e-05
```

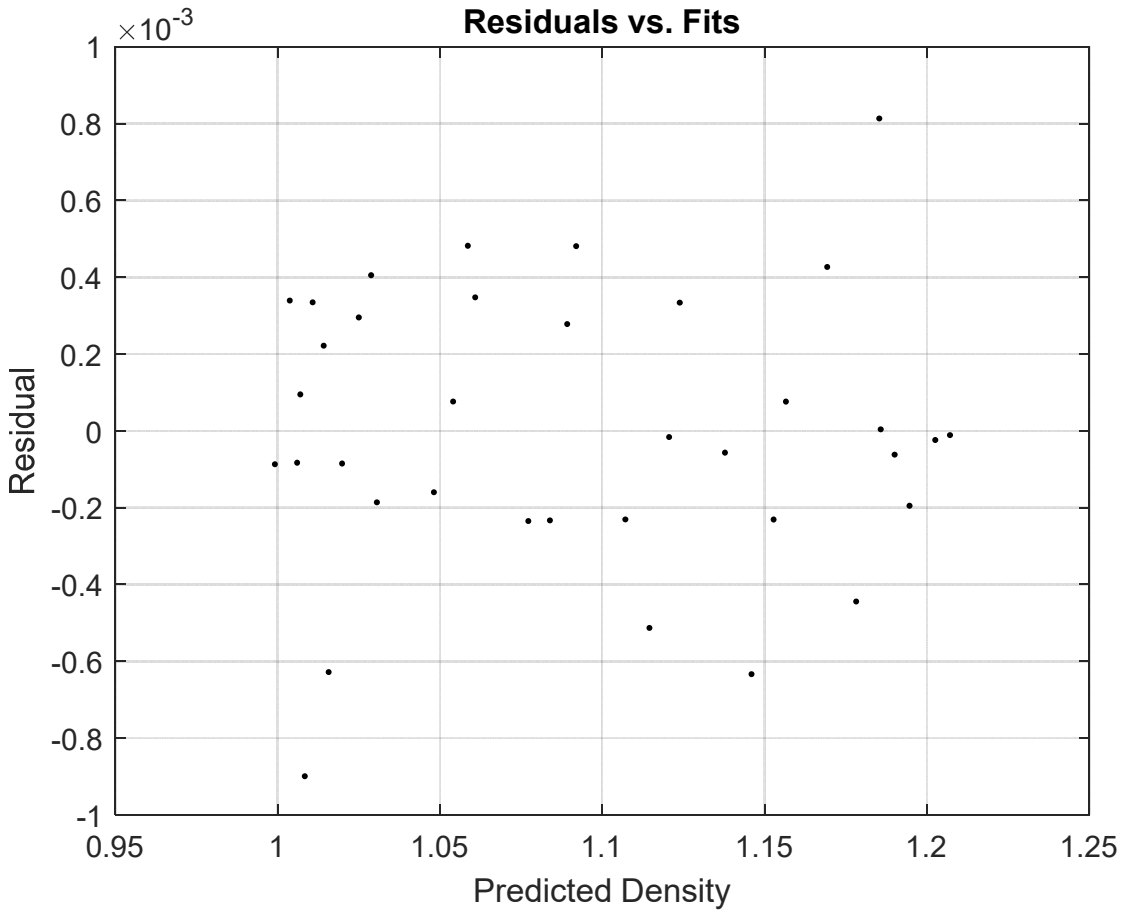
Agreement between predicted and measured values



Curve-Fitting

Multilinear regression using vector-matrix calculations

Errors < 0.001
No significant pattern
Model appears to
be adequate



Curve-Fitting

Multilinear regression using the MATLAB **fitlm** function

```
clear
w = [ 1 2 4 8 12 16 20 24 26 ];
T = [ 0 10 25 40 ];
rho = [ 1.00747, 1.00707, 1.00409, 0.99908 ; ...
        1.01509, 1.01442, 1.01112, 1.00593 ; ...
        1.03038, 1.02920, 1.02530, 1.01977 ; ...
        1.06121, 1.05907, 1.05412, 1.04798 ; ...
        1.09244, 1.08946, 1.08365, 1.07699 ; ...
        1.12419, 1.12056, 1.11401, 1.10688 ; ...
        1.15663, 1.15254, 1.14533, 1.13774 ; ...
        1.18999, 1.18577, 1.17776, 1.16971 ; ...
        1.20709, 1.20254, 1.19443, 1.18614 ] ;
n1 = length(w);
n2 = length(T);
```

```
for i = 1:n1 % create vectors of independent variables
    for j = 1:n2
        wn((i-1)*n2+j) = w(i);
        Tn((i-1)*n2+j) = T(j);
    end
end
wn = wn';
Tn = Tn';
X = [ wn Tn wn.^2 Tn.^2 wn.*Tn];  % create the X matrix
y = reshape(rho',[n1*n2,1]);  % create the y vector
mdl = fitlm(X,y)
```

NaClDensityRegression.m

Curve-Fitting

Multilinear regression using the MATLAB **fitlm** function

```
>> NaClDensityRegression

mdl =


Linear regression model:
    y ~ 1 + x1 + x2 + x3 + x4 + x5

Estimated Coefficients:
                    Estimate          SE          tStat          pValue
                   ___________    ___________    ___________    ___________

    (Intercept)     1.0011e+00    2.1778e-04     4.5967e+03    2.7744e-89
    x1              7.2739e-03    3.1725e-05     2.2928e+02    3.1758e-50
    x2             -9.6642e-05    1.7055e-05    -5.6665e+00    3.5531e-06
    x3              2.5015e-05    1.1176e-06     2.2382e+01    2.8365e-20
    x4             -3.0211e-06    3.8114e-07    -7.9265e+00    7.5816e-09
    x5             -1.2572e-05    4.8072e-07    -2.6153e+01    3.3132e-22


Number of observations: 36, Error degrees of freedom: 30
Root Mean Squared Error: 0.00039
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 2.25e+05, p-value = 1.2e-67
```

Can also perform step-wise regression with function **stepwiselm**

Curve-Fitting

Nonlinear regression

Model: $y = f(\mathbf{x}, \boldsymbol{\beta})$ Dataset: $\{y_i, x_{1i}, \ldots, x_{mi}, i = 1, \ldots, n\}$

$$\mathbf{f}(\mathbf{x}, \boldsymbol{\beta}) = \begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix}$$

$\mathbf{e} = \mathbf{y} - \mathbf{f}(\mathbf{x}, \hat{\boldsymbol{\beta}})$ $\min_{\hat{\boldsymbol{\beta}}} \mathbf{e}^T \mathbf{e}$ using an optimization routine

Curve-Fitting Nonlinear regression

Example: fitting the Antoine equation to vapor pressure data

$$\log_{10} P_V = A - \frac{B}{C+T}$$

Vapor Pressure of 95%(wt) Sulfuric Acid Aqueous Solution

Temperature (degC)	Vapor Pressure (torr)
35	0.0015
40	0.00235
45	0.0037
50	0.0058
55	0.00877
60	0.0133
65	0.0196
70	0.0288
75	0.0415
80	0.0606
85	0.0879
90	0.123
95	0.172
100	0.237
105	0.321
110	0.437
115	0.59
120	0.788
125	1.07
130	1.42
135	1.87
140	2.4
145	3.11
150	4.02
155	5.13
160	6.47
165	8.39
170	10.3
175	12.9
180	15.9
185	20.2
190	24.8
195	30.7
200	36.7
205	45.3
210	55
215	66.9
220	79.8
225	95.5
230	115
235	137
240	164
245	193
250	229
255	268
260	314
265	363
270	430
275	500
280	580
285	682
290	790

Curve-Fitting Nonlinear regression

```
clear
sizeH2SO4 = [2 Inf];
formatSpec = '%f';
fileID = fopen('H2SO4VaporPressure.txt','r');
TVP = fscanf(fileID,formatSpec,sizeH2SO4);
TVP = TVP';
T = TVP(:,1);
VP = TVP(:,2);
LVP = log10(VP);
A = 10;
B = 2000;
C = 250;
```

```
x0 = [ A B C];
Antanon = @(x) Antoine(x,T,LVP);
Params = fminunc(Antanon,x0)
Aopt = Params(1);
Bopt = Params(2);
Copt = Params(3);
LVPM = Aopt - Bopt ./ (Copt + T);
plot(T,LVP,'ko',T,LVPM,'k-')
grid
xlabel('Temperature - degC')
ylabel('Log10(Vapor Pressure)')
title('Vapor Pressure of Concentrated H2SO4 Solutions')
```

H2SO4AntoineEqn.m

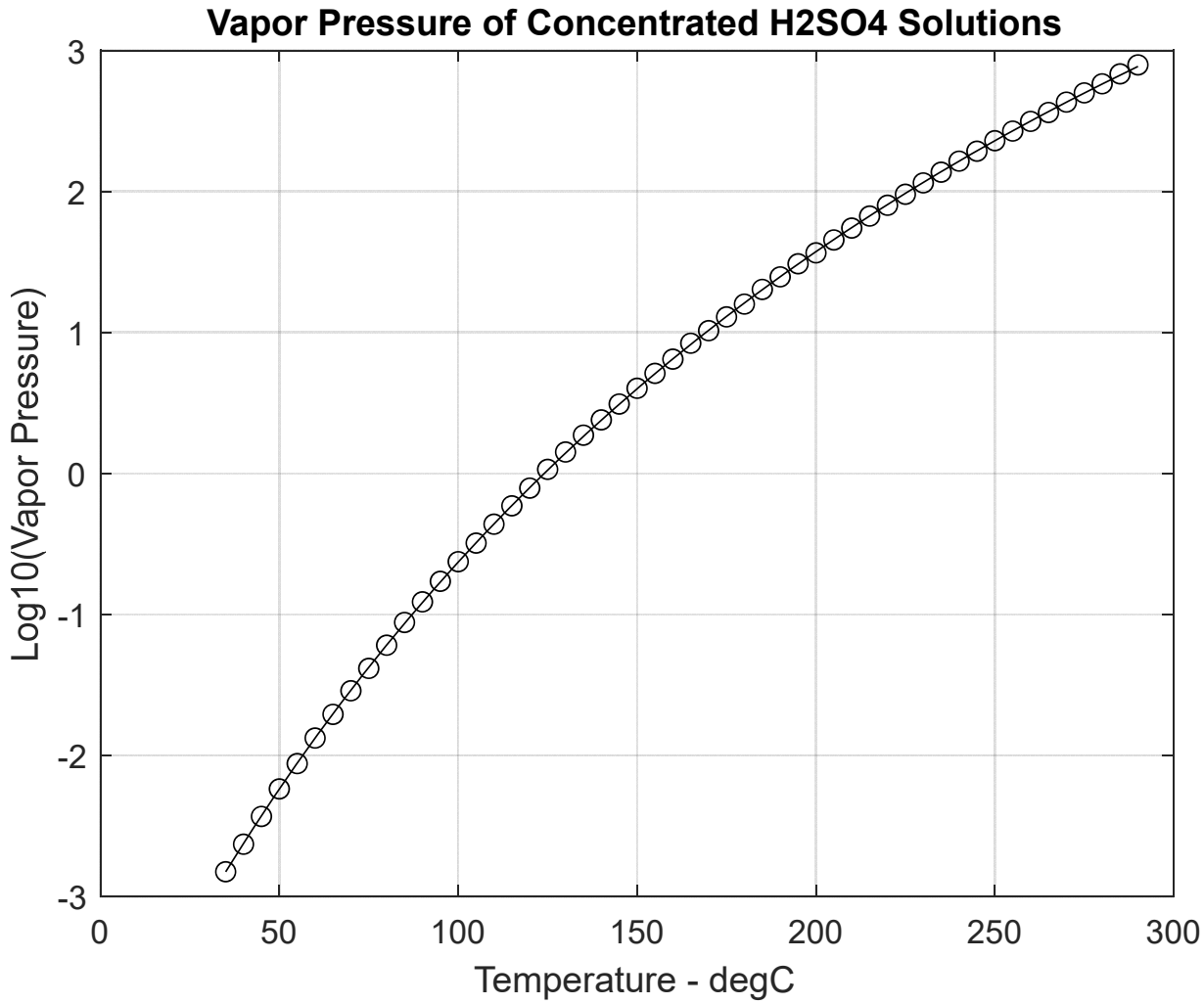
Antoine.m

```
function SSE = Antoine(x,T,LVP)
A = x(1);
B = x(2);
C = x(3);
LVPM = A - B ./ (C + T);
VPerr = LVP - LVPM;
SSE = VPerr'*VPerr;
```

Curve-Fitting Nonlinear regression

```
>> H2SO4AntoineEqn

Local minimum found.

Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.

<stopping criteria details>

Params =

   9.8053e+00   3.9014e+03   2.7391e+02
```

Curve-Fitting Nonlinear regression



Reference

Chapra, Steven C.
Applied Numerical Methods
with MATLAB for Engineers and Scientists
5th Edition, McGraw-Hill, 2022.

"Prof. Clough, may I be excused? My brain is full."

MATLAB – What's Next?

Bootcamp 3

- ✓ 1: Getting up to speed (or back up to speed) with MATLAB
- ✓ 2: Learning to use MATLAB to solve typical problem scenarios
- • 3: Detailed modeling of packed-bed and plug-flow reactors