Applications of MATLAB/Simulink for Process Dynamics and Control

(This lecture was modified from slides provided by Professor Kirk Dolan and Wei Liao at MSU and Venkat Subramanian at WashU)
Outline

- Introduction of Matlab/simulink
  - Background
  - Matlab fundamentals
- Matlab Applications
  - Parameter estimation (Creep compliance using excel and Toolbox)
  - Kinetics modeling (ode45)
  - Simulink
    - Simulink tutorials: Blocks and links; Building a simple system; Running simulations
    - Simulink examples
- Parameter fitting and Process Optimization (fmin function and nlinfit)
1. MATLAB Introduction

- Matlab is short for “MATrix LABoratory”
- High-performance technical computing environment
  - Comprehensive math
  - Graphic functions
  - Powerful high-level language
- Simulink is a platform for multidomain simulation and model-based design of dynamic systems
  - Process control dynamics (ChE462 teaches design of feedback and feedforward control using Simulink)
  - Bioprocess Engineering (Solve multiple kinetic models)
Background

Structure of Matlab/Simulink

MATLAB based software…

- COMSOL
- Formerly FEMLAB
- A finite element analysis, solver and simulation
- Simulink
- Matlab Platform
- Toolboxes
- Curvefitting toolbox
- Optimization toolbox
- Differential equation toolbox
- Other toolboxes……
- M. File
- Systems biology toolbox (new)
Background

History of Matlab/Simulink

- Later 1970s Cleve Moler invented the Matlab
- 1984 The Mathworks founded by Cleve Moler, Jack Litter, and Steve Bangert in Portola Valley, CA
- 1985 The first order from MIT
- 1986 MATLAB 2 released, moved to Massachusetts
- 1987 MATLAB 3 released

- 1990 Simulink 1.0 released
- 1992 MATLAB 4 with 2- and 3-D color graphics and sparse matrices
- 1993 MATLAB for Microsoft Windows
- 1995 MATLAB for Linux
- 1996 MATLAB 5 with advanced visualization, debugger, and a GUI builder
- 2000 MATLAB 6 with new MATLAB desktop and faster Fourier transform
- 2004 MATLAB 7 released with large-scale modeling and more sophisticated Simulink
- 2009 MATLAB 7.9 added support for the 64-bit Mac platform, and for processing large image files in Image Processing Toolbox
MATLAB fundamentals

MATLAB uses Command window Graphics window(s) Edit window. The MATLAB command widow can be used as a calculator:

For symbolic processing (using the “syms” command)
>> syms a n x y z t k s Note: a n x y z t k s are symbolic objects.

Laplace Transform >> laplace (exp(-a*t)) >> ans =1/(s+a)

Example 1: Using MATLAB to find the roots of \( (S^4 + 6S^2 + 11S + 6=0) \): “roots” command
>> n=[1 0 6 11 6]
>> roots (n)
Note: vector n contains coefficients of the polynomial equation

Example 2: Solve symbolic solution of algebraic equations (“solve”) \[ \begin{align*}
    &x^2 + xy + y = 3 \\
    &x^2 - 4x + 3 = 0
\end{align*} \]
>> [x, y] = solve('x^2 + x*y + y = 3','x^2 - 4*x + 3 = 0')
Note: you will get two solutions

Example 3: Solution of linear differential equation (“dsolve”, Dx is first derivative, D2x is second derivative, etc.) \[ \begin{align*}
    &\frac{d^3 x}{dt^3} + 2 \frac{d^2 x}{dt^2} - \frac{dx}{dt} - 2x = 4 + e^{2t} \\
    &x(0) = 1 \quad x'(0) = 0 \quad x''(0) = -1
\end{align*} \]
>> dsolve('D3x+2*D2x-Dx-2*x=4+exp(2*t)','x(0)=1','Dx(0)=0','D2x(0)=-1')
ans = -2+1/12*exp(2*t)+2/3*exp(t)+11/3*exp(-t)-17/12*exp(-2*t)
MATLAB fundamentals

• MATLAB can automatically handle rectangular arrays of data - one-dimensional arrays are called **vectors** and *multi-dimensional* arrays are called **matrices**.
• Arrays are set off using square brackets [ ] in MATLAB.
• Entries within a row are separated by spaces or commas.
• Rows are separated by semicolons.
• The transpose operator (apostrophe) can be used to flip an array over its own diagonal. For example, if b is a row vector, b’ is a column vector.

**Examples:**

```
>> a = [1 2 3 4 5 ]
a =
    1     2     3     4     5
>> b = [2;4;6;8;10]
b =
    2
    4
    6
    8
   10
```

Q: What is a’=?

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
    1     2     3
    4     5     6
    7     8     9
```

Assuming some matrix C:

```
C =
    2     4     9
    3     3    16
    3     0     8
   10    13    17
```

C (2) would report 3
C (4) would report 10
C (13) would report an error!

Entries can also be accessed using the row and column:

C (2, 1) would report 3
C (3, 2) would report 0
C (5, 1) would report an error!
MATLAB fundamentals

Array Creation - Colon Operator

The colon operator : is useful in several contexts. It can be used to create a linearly spaced array of points using the notation

\[ \text{start:diffval:limit} \]

where \( \text{start} \) is the first value in the array, \( \text{diffval} \) is the difference between successive values in the array, and \( \text{limit} \) is the boundary for the last value (though not necessarily the last value).

\[ \gg 1:0.6:3 \]
\[ \text{ans} = \]
\[ 1.0000 \quad 1.6000 \quad 2.2000 \quad 2.8000 \]

Colon Operator - Notes

If \( \text{diffval} \) is omitted, the default value is 1:

\[ \gg 3:6 \]
\[ \text{ans} = \]
\[ 3 \quad 4 \quad 5 \quad 6 \]

To create a decreasing series, \( \text{diffval} \) must be negative:

\[ \gg 5:-1.2:2 \]
\[ \text{ans} = \]
\[ 5.0000 \quad 3.8000 \quad 2.6000 \]

If \( \text{start}+\text{diffval}<\text{limit} \) for an increasing series or \( \text{start}+\text{diffval}>\text{limit} \) for a decreasing series, an empty matrix is returned:

\[ \gg 5:2 \]
\[ \text{ans} = \]
\[ \text{Empty matrix: 1-by-0} \]

To create a column, transpose the output of the colon operator, not the limit value; that is, \( (3:6)' \), not \( 3:6' \)

You can use help check more commands:

logspace  linspace

Clearing Commands

- When running a program many times, the command window may become cluttered. Clear the command window with —clc. (clear command).
- Good programming practice: At the beginning of the program, clear all variables: —clear: removes all variables from workspace (“clear all” clears all objects in workspace, plus resets all assumptions.)
MATLAB fundamentals

Graphics

- MATLAB has a powerful suite of built-in graphics functions.
- Two of the primary functions are `plot` (for plotting 2-D data) and `plot3` (for plotting 3-D data).
- In addition to the plotting commands, MATLAB allows you to label and annotate your graphs using the `title`, `xlabel`, `ylabel`, and `legend` commands.

```
t = [0:2:20]';
g = 9.81; m = 68.1; cd = 0.25;
v = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);
plot(t, v)
```

- `subplot(m, n, p)` — `subplot` splits the figure window into an \( m \times n \) array of small axes and makes the \( p \)th one active. Note - the first subplot is at the top left, then the numbering continues across the row. This is different from how elements are numbered within a matrix!

Hold on and hold off
- `hold on` keep the current data plotted and add the results of any further plot commands to the graph. This continues until the `hold off` command, which clear the graph and start over if another plotting command is given. `hold on` should be used after the first plot in a series is made.
There are two main kinds of M-file
– **Script files:** A *script file is merely a set of MATLAB commands* that are saved on a file - when MATLAB runs a script file, it is as if you typed the characters stored in the file on the command window. Scripts can be executed either by typing their name (without the .m) in the command window, by selecting the Debug, or Run (or Save and Run) command in the editing window.

– **Function files:** *Function files serve an entirely different* purpose from script files. Function files can accept input arguments from and return outputs to the command window, but variables created and manipulated within the function do not impact the command window. A function file can contain a single function, but it can also contain a primary function and one or more subfunctions. The primary function is whatever function is listed first in the M-file - its function name should be the same as the file name. Subfunctions are only accessible by the main function.
The general syntax for a function is:

```matlab
function outvar = funcname(arglist)
    %help comments
    statements
    outvar = value;
end
```

where

- `outvar`: output variable name
- `funcname`: function name
- `arglist`: input argument list

- `helpcomments`: text to show with `help funcname`
- `statements`: MATLAB commands for the function

---

**Script and function files Debug**

- Put breakpoints after the line you want to see.
- The program will stop and allow seeing the variable values.
- Can move through program step-by-step.
- Only works in the main program. For functions, can use `pause` and `print` for output values.
- `Ctrl+C` will stop any program and get you out.
MATLAB fundamentals (Input and Output)

- The easiest way to get a value from the user is the input command:
  
  \[ n = \text{input('promptstring')} \]
  
  MATLAB will display the characters in `promptstring`, and whatever value is typed is stored in \( n \). For example, if you type `pi`, \( n \) will store 3.1416...

- \[ n = \text{input('promptstring', 's')} \]
  
  MATLAB will display the characters in `promptstring`, and whatever characters are typed will be stored as a string in \( n \). For example, if you type `pi`, \( n \) will store the letters `p` and `i` in a 2x1 char array.

- For numbers:
  
  \[
  \begin{align*}
  \text{>> } m &= \text{input('Mass (kg):')} \\
  \text{Mass (kg):} &= 22 \\
  m &= 22
  \end{align*}
  \]

- For strings:
  
  \[
  \begin{align*}
  \text{>> } n &= \text{input('What is your name? ','s')} \\
  \text{What is your name?} &= \text{Kirk Dolan} \\
  n &= \text{Kirk Dolan}
  \end{align*}
  \]

Data can be read from Excel files

- Data are often exchanged among people by storing in Excel files and sending via email.

  \[
  \text{num} = \text{xlsread(filename)}
  \]

- Reads in the data, and ignores headers and any other outer non-numeric data. Inner non-numeric data are not ignored.

- To be more specific within the excel sheet, use

  \[
  \text{num} = \text{xlsread(filename, sheet, range)}
  \]

Where `sheet` is the name of the excel worksheet, and `range` is the rectangular region of the sheet (cell number(s))
MATLAB fundamentals (Input and Output)

- The easiest way to display the value of a matrix is to type its name, but that will not work in function or script files. Instead, use the `disp` command
  
  ```matlab
  disp(value)
  ```

  will show the value on the screen, and if it is a string, will enclose it in single quotes.

Creating and Accessing Files

- MATLAB has a built-in file format that may be used to save and load the values in variables.
- `save filename var1 var2 ... varn` saves the listed variables into a file named `filename.mat`. If no variable is listed, all variables are saved.
- `load filename var1 var2 ... varn` loads the listed variables from a file named `filename.mat`. If no variable is listed, all variables in the file are loaded.

- Note - these are not text files!

Example saving .mat files

```matlab
>> g=9.81; m= 80; t=5;
>> cd=[.25 .267 .245 .28 .273]';
>> v=sqrt(g*m./cd).*tanh(sqrt(g*cd/m)*t);
save veldrag v cd
>> clear
>> v
??? Undefined function or variable 'v'.
load veldrag
>> v
v =
    39.4514
    38.9587
    39.5993
    38.5921
    38.7884
```

Excel Files--output

```matlab
xlswrite(filename, M, sheet, range)
```

`M` is the matrix to be written.

If `filename` does not exist, `xlswrite` creates it.
Example reading from one Excel file, doing computations, and writing to another Excel file

data = xlsread('veldat.xls');
time = data(:,1);
velocity = data(:,2);
veltime=time;
veltime(:,2)=velocity
columnHeader ={"time (s)";"velocity (m/s)"}
xlswrite('results.xls',columnHeader,'Sheet1','A1');
>> xlswrite('results.xls', veltime , 'Sheet1', 'A2');

Will create an excel file with the vector velocity written starting in cell A2

More options: See also xlswrite, csvread, csvwrite, dlmread, dlmwrite, textscan.
MATLAB fundamentals (programming)

Structured Programming

- Structured programming allows MATLAB to make decisions or selections based on conditions of the program.
- Decisions in MATLAB are based on the result of logical and relational operations and are implemented with if, if...else, and if...elseif structures.

Rational Operators

<table>
<thead>
<tr>
<th>Example</th>
<th>Operator</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>x == 0</td>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>unit ~= 'm'</td>
<td>~=</td>
<td>Not equal</td>
</tr>
<tr>
<td>a &lt; 0</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>s &gt; t</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>3.9 &lt;= a/3</td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>r &gt;= 0</td>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

Example

```matlab
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```
MATLAB fundamentals (programming, Loops)

Another programming structure involves loops, where the same lines of code are run several times. There are two types of loop:

- A for loop ends after a specified number of repetitions established by the number of columns given to an index variable.
- A while loop ends on the basis of a logical condition.

for Loops

One common way to use a for...end structure is:

```
for index = start:step:finish
    statements
end
```

where the `index` variable takes on successive values in the vector created using the `:` operator.

while Loops

- A while loop is fundamentally different from a for loop since while loops can run an indeterminate number of times. The general syntax is:

```
while condition
    statements
end
```

where the `condition` is a logical expression. If the `condition` is true, the `statements` will run and when that is finished, the loop will again check on the `condition`.

- Note - though the `condition` may become false as the `statements` are running, the only time it matters is after all the statements have run.
2. Matlab Applications

a) Parameter estimation (curve fitting toolbox)*

- Different ways to estimate parameters
  - excel “solver”
  - A graphical user interface (curve fitting toolbox)
  - Matlab command (nlinfit or fmincon)

- Example
  - Creep compliance of a wheat protein film (determination of retardation time and free dashpot viscosity in the Jefferys model)

\[ J = J_1 (1 - \exp\left(\frac{-t}{\lambda_{ret}}\right)) + \frac{t}{\mu_0} \]

Note: creep is the tendency of a solid material to slowly move or deform permanently under the influence of stresses.

Where \( J \) is the strain, \( J_1 \) is the retarded compliance (Pa\(^{-1}\)); \( \lambda_{ret} = \mu_1 / G_1 \) is retardation time constant (s); \( \mu_0 \) is the free dashpot viscosity (Pa s); \( t \) is the time.
2. Matlab Applications

Parameter estimation - Creep compliance Experimental Data

<table>
<thead>
<tr>
<th>t, Time (s)</th>
<th>J, Strain (Mpa^-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>0.17</td>
</tr>
<tr>
<td>600</td>
<td>0.265</td>
</tr>
<tr>
<td>900</td>
<td>0.318</td>
</tr>
<tr>
<td>1200</td>
<td>0.348</td>
</tr>
<tr>
<td>1500</td>
<td>0.366</td>
</tr>
<tr>
<td>1800</td>
<td>0.376</td>
</tr>
<tr>
<td>2100</td>
<td>0.382</td>
</tr>
<tr>
<td>2400</td>
<td>0.386</td>
</tr>
<tr>
<td>2700</td>
<td>0.388</td>
</tr>
<tr>
<td>3000</td>
<td>0.39</td>
</tr>
<tr>
<td>3300</td>
<td>0.392</td>
</tr>
<tr>
<td>3600</td>
<td>0.393</td>
</tr>
<tr>
<td>3900</td>
<td>0.395</td>
</tr>
<tr>
<td>4200</td>
<td>0.396</td>
</tr>
<tr>
<td>4500</td>
<td>0.397</td>
</tr>
<tr>
<td>4800</td>
<td>0.398</td>
</tr>
<tr>
<td>5100</td>
<td>0.4</td>
</tr>
<tr>
<td>5400</td>
<td>0.401</td>
</tr>
<tr>
<td>5700</td>
<td>0.402</td>
</tr>
<tr>
<td>6000</td>
<td>0.403</td>
</tr>
<tr>
<td>6300</td>
<td>0.404</td>
</tr>
<tr>
<td>6600</td>
<td>0.405</td>
</tr>
<tr>
<td>6900</td>
<td>0.406</td>
</tr>
<tr>
<td>7200</td>
<td>0.408</td>
</tr>
</tbody>
</table>

Note: Unlike brittle fracture, creep deformation does not occur suddenly upon the application of stress. Instead, strain accumulates as a result of long-term stress. Creep is a "time-dependent" deformation.

*Chewing gum*
Method 1) Using Excel Solver

\[ J = J_1 \left( 1 - \exp\left( \frac{-t}{\lambda_{ret}} \right) \right) + \frac{t}{\mu_0} \]
Method 2) CFTtoolbox

Parameter estimation - Creep compliance

❖ Use the command of  >>cftool
Method 2: Parameter estimation - Creep compliance

1. Input data
2. Select Data

3. Define Model

4. Initial Guess

Initial guesses and boundaries are important for a global solution
Parameter estimation - Creep compliance

Results

Initial value (start point): $a = 0.1$, $b = 1000$, $c = 100000$

The results:

$$J_1 = a = 0.38 \, \text{Mpa}^{-1}$$
$$\lambda_{ret} = b = 510.6 \, \text{s}$$
$$\mu_0 = c = 260800 \, \text{Mpa} \, \text{s}$$

The procedure:

http://www.swarthmore.edu/NatSci/echeeve1/Ref/MatlabCurveFit/MatlabCftool.html
2. Matlab ODE45 function (solve kinetic models)

- Ordinary Differential Equation (ODE): The dependent variable is a function of only one independent variable.
- Partial Differential Equation (PDE): The dependent variable is a function of more than one dependent variable.
- The order of a differential equation is the highest derivative in the expression.
- A differential equation is linear if the unknown function and its derivatives appear to the power one without products, and nonlinear otherwise.
Initial-Value Problem

ODE equations
\[ \frac{dy}{dt} = f(t, y) \]
y(0) = initial value; Find y(t)

- Methods described here are for solving differential equations of the form:
  \[ \frac{dy}{dt} = f(t, y) \]
- The methods in this chapter are all one-step methods and have the general format:
  \[ y_{i+1} = y_i + \phi h \]
  where \( \phi \) is called an increment function, and is used to extrapolate from an old value \( y_i \) to a new value \( y_{i+1} \).

Use Excel to Solve

Euler’s Method
- The first derivative provides a direct estimate of the slope at \( t_i \):
  \[ \frac{dy}{dt} \bigg|_{t_i} = f(t_i, y_i) \]
  and the Euler method uses that estimate as the increment function:
  \[ \phi = f(t_i, y_i) \]
  \[ y_{i+1} = y_i + f(t_i, y_i) h \]

Use MATLAB ode45

Classical Fourth-Order Runge-Kutta Method
- The most popular RK methods are fourth-order, and the most commonly used form is:
  \[ y_{i+1} = y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) h \]
  where:
  \[ k_1 = f(t_i, y_i) \]
  \[ k_2 = f \left( t_i + \frac{1}{2} h, y_i + \frac{1}{2} hk_1 \right) \]
  \[ k_3 = f \left( t_i + \frac{1}{2} h, y_i + \frac{1}{2} hk_2 \right) \]
  \[ k_4 = f \left( t_i + h, y_i + hk_3 \right) \]
1. The solutions to some ODE problems exhibit multiple time scales - for some parts of the solution the variable changes slowly, while for others there are abrupt changes. Constant step-size algorithms would have to apply a small stepsize to the entire computation, wasting many more calculations on regions of gradual change.

2. MATLAB’s ode23 function uses second- and third order functions to solve the ODE. ode45 function uses fourth- and fifth order RK functions. Ode23 is better than ode45 for stiff problems.

MATLAB has a number of built-in functions for solving stiff systems of ODEs, including ode15s, ode23s, ode23t, and ode23tb
Solve high order ODEs
(Express an \( n \)th order ODE as a system of \( n \) first-order ODEs)

\[ y_1(0) = \frac{dy_1}{dt} = 1. \]

\[ \frac{d^2 y_1}{dt^2} - \mu (1 - y_1^2) \frac{dy_1}{dt} + y_1 = 0 \]

Convert the second-order ODE into a pair of first order ODEs by defining

\[ \frac{dy_1}{dt} = y_2 \]

\[ \frac{dy_2}{dt} = \mu (1 - y_1^2) y_2 - y_1 = 0 \]
2. Matlab Application (b kinetic models)

b) kinetic models

Kinetic model 1: Enzyme kinetics

\[
\begin{align*}
\frac{d[S]}{dt} &= -k_1 [S][E] + k_{-1} [ES] \\
\frac{d[E]}{dt} &= -k_1 [S][E] + (k_{-1} + k_2) [ES] \\
\frac{d[ES]}{dt} &= k_1 [S][E] - (k_{-1} + k_2) [ES] \\
\frac{d[P]}{dt} &= k_2 [ES]
\end{align*}
\]

Matlab solution to equations using ode45
clear
clf
global  k1 k2 k3;  % define rate constant, k3 is k1-minus.
k1=100; k2=0.1; k3=100;

% initial conditions
S0=1; E0=1; ES0=0; P0=0; tend=100;
y0=[S0 E0 ES0 P0];
[t,y]=ode45('MMmodel',[0 tend], y0);

% results
S=y(:,1); E=y(:,2); ES=y(:,3); P=y(:,4);

subplot(211)
plot (t, S, 'k', t, P, 'y');
title('Substrate vs Products'), xlabel('time'), ylabel('amount'),
legend('substrate','product', 1);

subplot(212)
plot (t, E, 'k', t, ES, 'y');
title('Enzyme vs Enzyme complex'), xlabel('time'), ylabel('amount'),
legend('Enzyme','Enzyme complex', 1);

function ydot=MMmodel(t,y)
global  k1 k2 k3;
ydot(1)= -k1*y(1)*y(2)+k3*y(3);  %substrate
ydot(2)= -k1*y(1)*y(2)+(k3+k2)*y(3);  %enzyme
ydot(3)= k1*y(1)*y(2)-(k3+k2)*y(3);  %enzyme complex
ydot(4)= k2*y(3);  % product
ydot=ydot';

Enzyme kinetic model
Process dynamics can be unstable (no steady state)

\[
\frac{dy_1}{dt} = k_1 \cdot y_1 - k_{d1} \cdot y_1 - k_2 \cdot y_1 \cdot y_2
\]

\[
\frac{dy_2}{dt} = -k_{d2} \cdot y_2 + k_3 \cdot y_1 \cdot y_2
\]
Predator-Prey Population Cycles

Predator and prey populations exhibit fluctuations described as the predator "tracking" the prey. The classic example is the rabbit and cat populations.

![Graph showing the oscillations of substrate and products over time, with peaks and troughs indicating the predator-prey cycles.](image-url)
3. Simulink

Basic elements

❖ Blocks
  ❖ Blocks: generate, modify, combine, and display signals
  ❖ Typical blocks
    ❖ Continuous: Linear, continuous-time system elements (integrators, transfer functions, state-space models, etc.)
    ❖ Discrete: Linear, discrete-time system elements (integrators, transfer functions, state-space models, etc.)
    ❖ Functions & Tables: User-defined functions and tables for interpolating function values
    ❖ Math: Mathematical operators (sum, gain, dot product, etc.)
    ❖ Nonlinear: Nonlinear operators (coulomb/viscous friction, switches, relays, etc.)
    ❖ Signals: Blocks for controlling/monitoring signals
    ❖ Sinks: Used to output or display signals (displays, scopes, graphs, etc.)
    ❖ Sources: Used to generate various signals (step, ramp, sinusoidal, etc.)

❖ Lines
  ❖ transfer signals from one block to another
3. Simulink

Type “simulink” in the command window

The graphic interfaces
Tutorial example

\[
\frac{dy}{dt} + 2y = 3x + 1
\]  

(4)

where \( y \) is a dependent function of time, and \( x \) is independent. To form a Simulink model we first rearrange (4) to solve for the derivative,

\[
\frac{dy}{dt} = 3x + 1 - 2y
\]  

(5)

Then we arrange Simulink blocks to compute the \( y \) derivative, and integrate it to calculate \( y \). (We need \( y \) to compute the derivative, so the model must include a “recycle.”) Figure 10 shows one way to set it up (I have added the signal labels to help you understand it – they aren’t part of the normal display).

\[ x = \sin (t) \]
Creep compliance of a wheat protein film
Using formaldehyde cross-linker

- Creep compliance of a wheat protein film (determination of retardation time and free dashpot viscosity in the Jefferys model)

\[ J = J_1 (1 - \exp\left(\frac{-t}{\lambda_{ret}}\right)) + \frac{t}{\mu_0} \]

Where \( J \) is the strain, \( J_1 \) is the retarded compliance (Pa\(^{-1}\)); \( \lambda_{ret} = \mu_1/G_1 \) is retardation time (s); \( \mu_0 \) is the free dashpot viscosity (Pa s); \( t \) is the time.

- The recovery of the compliance is following the equation (\( t > t_1 \)):

\[ J = J_1 \exp\left(-\frac{t-t_1}{\lambda_{ret}}\right) \]

Where \( t_1 \) is the time the stress was released.
4. Simulink examples

Creep compliance of a wheat protein film

Simulink model

Parameters: \( J_1 = a = 0.38 \text{ Mpa}^{-1} \); \( \lambda_{\text{ret}} = b = 510.6 \text{ s} \); \( \mu_0 = c = 260800 \text{ Mpa s} \)

\[
J = J_1 (1 - \exp\left(\frac{-t}{\lambda_{\text{ret}}}\right)) + \frac{t}{\mu_0}
\]

\[
J = J_1 \exp\left(-\frac{t-t_1}{\lambda_{\text{ret}}}\right)
\]
4. Simulink examples

Creep compliance of a wheat protein film

Simulation result
4. The control (a) and control (b) are in the block diagram below. Discuss which control responds faster and which control causes higher frequency of oscillation? Please show your calculations to support your conclusion. (30 pts)

Control (a)

\[
\begin{align*}
Y_{SP} & \xrightarrow{+/-} Kc \xrightarrow{1/(s+1)^2} \xrightarrow{1/(5s+1)} Y \\
\end{align*}
\]

Control (b)

\[
\begin{align*}
Y_{SP} & \xrightarrow{+/-} Kc \xrightarrow{10} \xrightarrow{1/(s+1)^2} \xrightarrow{1/(5s+1)} Y \\
\end{align*}
\]