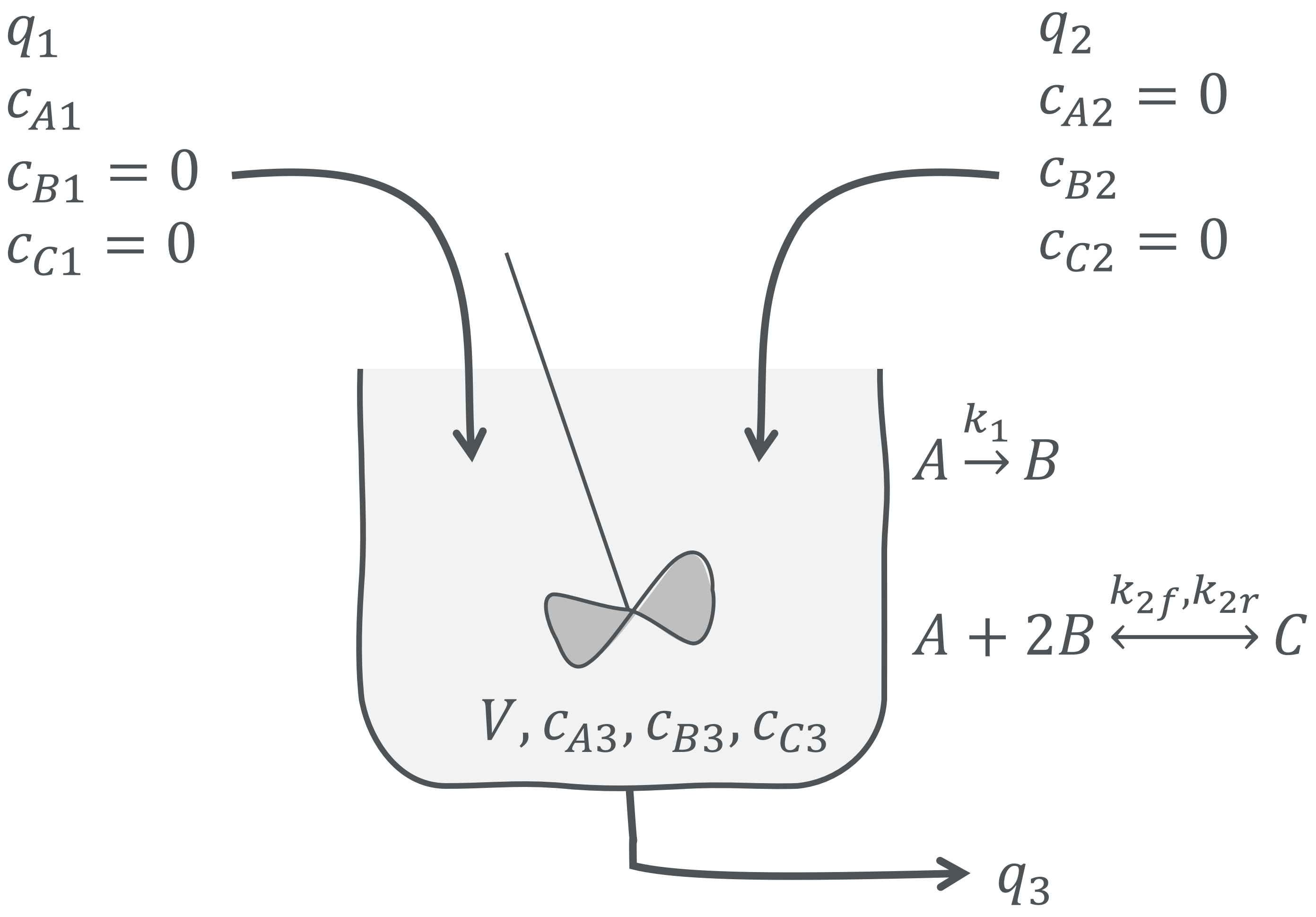
Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

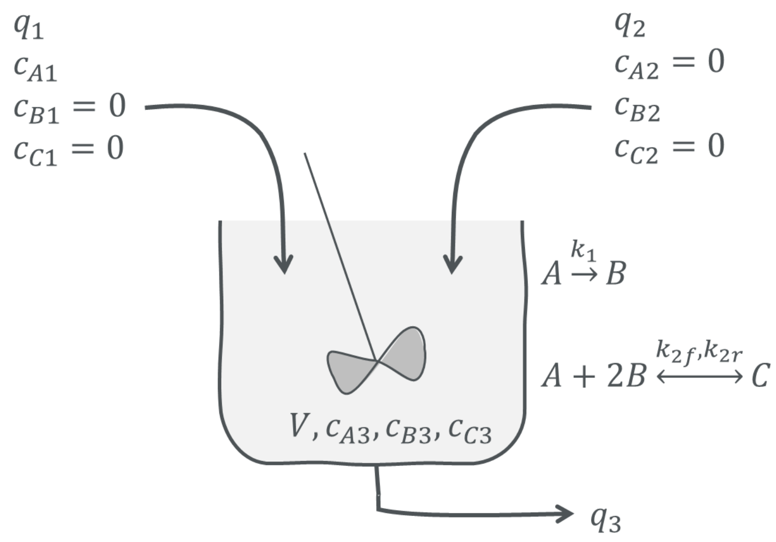
Developing and implementing systems of ODEs

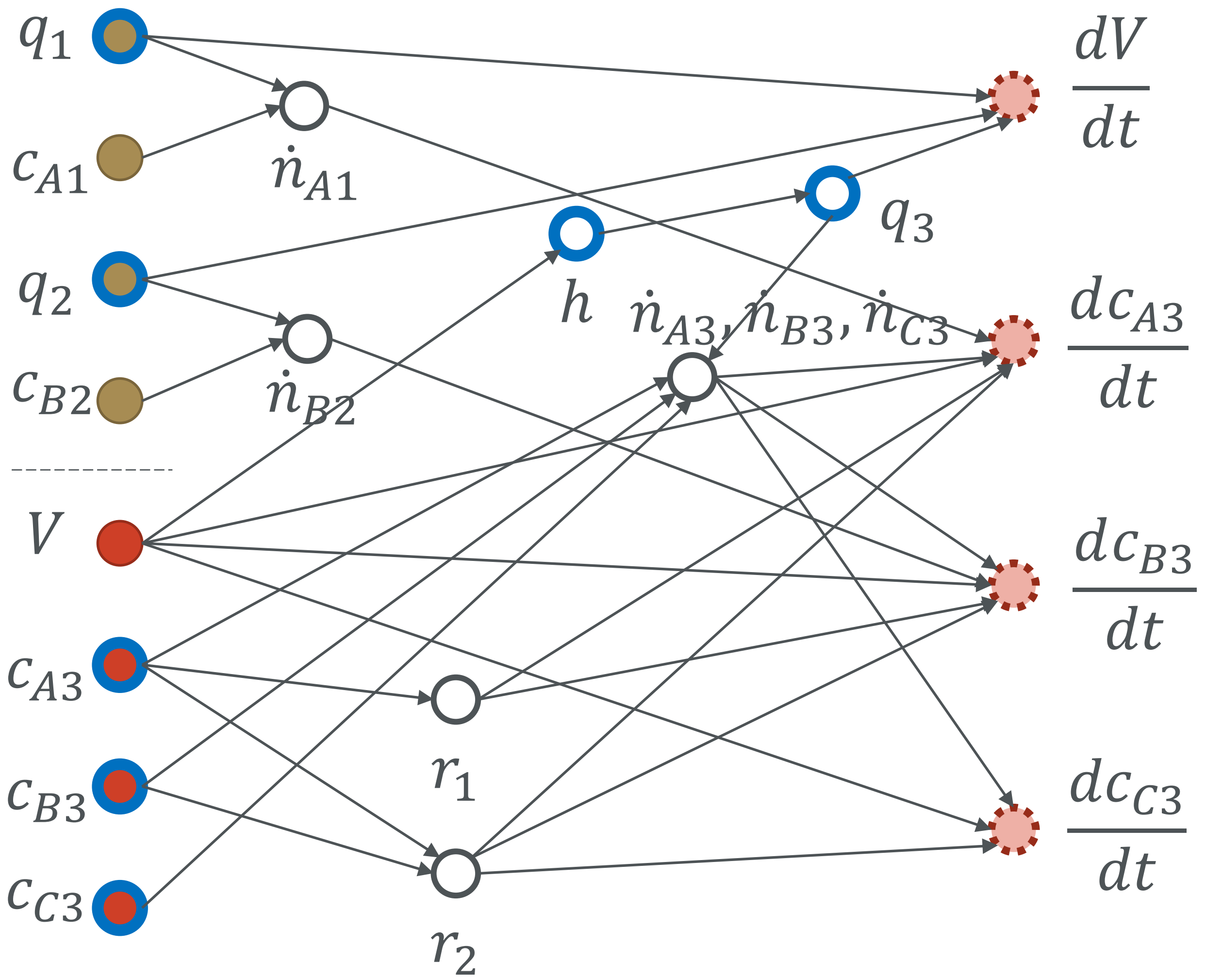
Mathematical modelling and machine learning, 2^{nd} of March 2022

Consider the following system

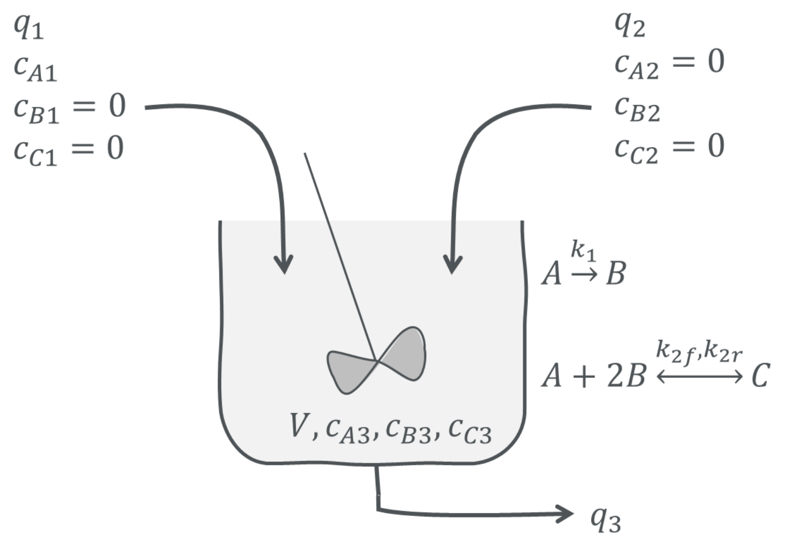


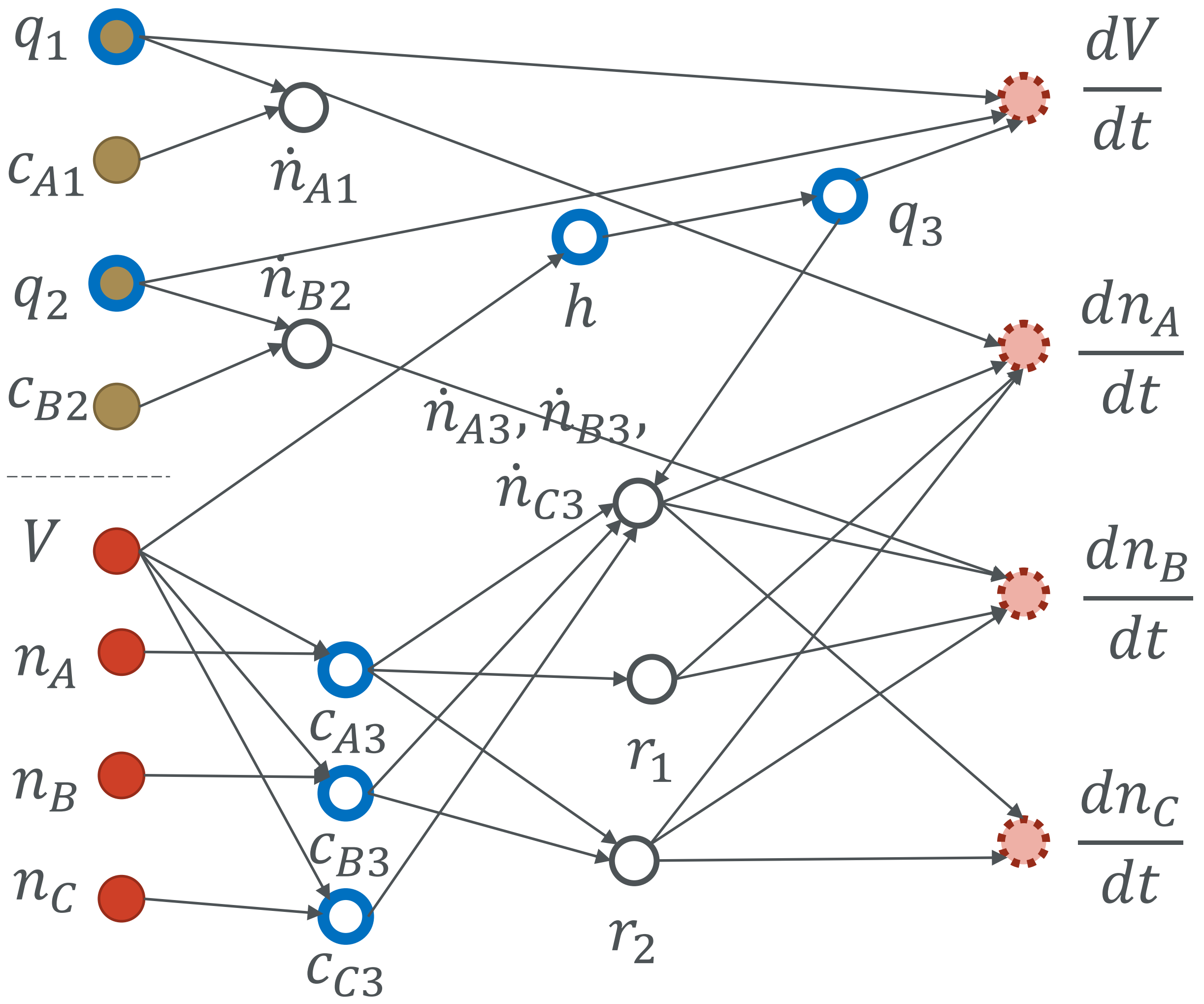
Consider the following system



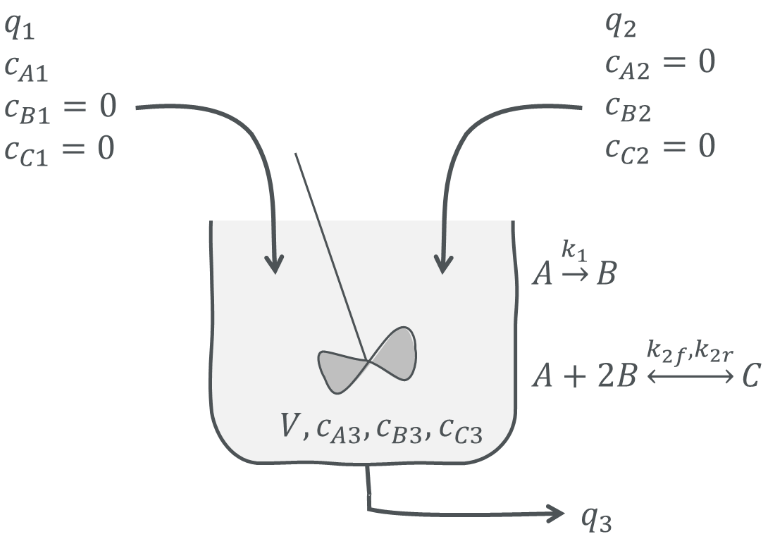


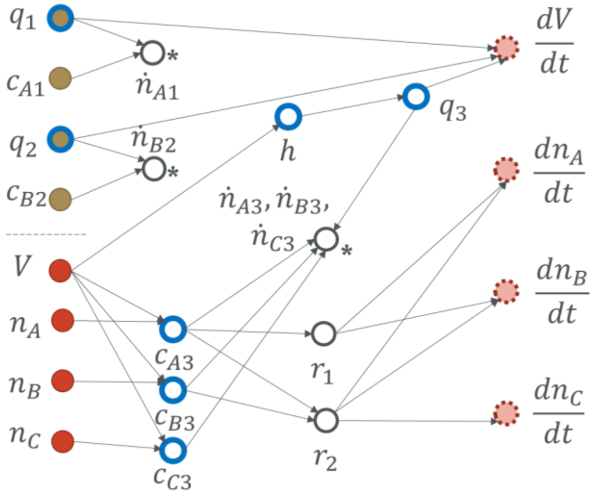
Consider the following system





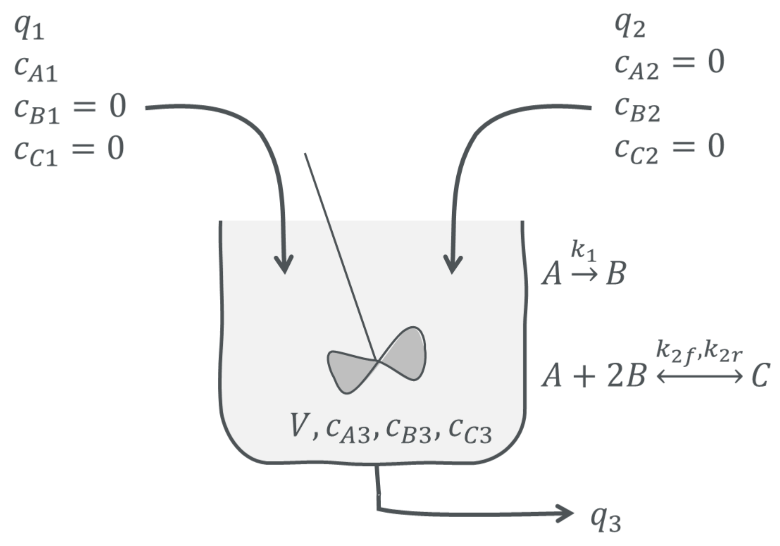
Consider the following system

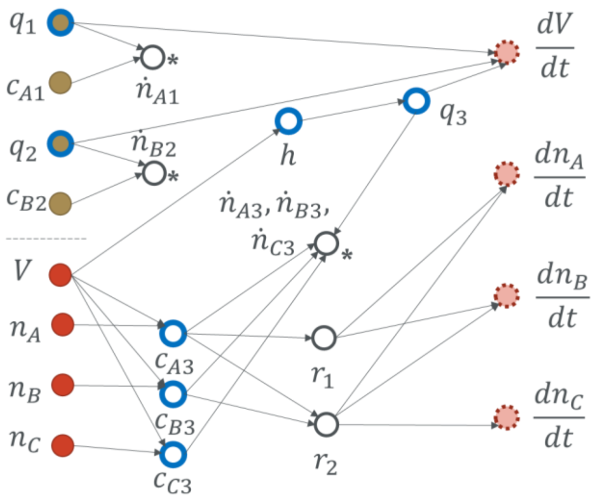




Equation	$\boldsymbol{x}, \boldsymbol{v}$	$\boldsymbol{u}$	$\boldsymbol{p}$
$\frac{dV}{dt} = q_1 + q_2 - q_3$	V	q_1, q_2	
$\frac{dn_A}{dt} = \dot{n}_{A1} - \dot{n}_{A3} + \nu_{A1} r_1 V + \nu_{A2} r_2 V$	n_A		ν_{A1}, ν_{A2}
$\frac{dn_B}{dt} = \dot{n}_{B2} - \dot{n}_{B3} + \nu_{B1} r_1 V + \nu_{B2} r_2 V$	n_B		ν_{B1}, ν_{B2}
$\frac{dn_C}{dt} = -\dot{n}_{C3} + \nu_{C1} r_1 V + \nu_{C2} r_2 V$	n_C		ν_{C1}, ν_{C2}
$q_3 = c_V \sqrt{h}$	q_3		c_V
$h = V/A$	h		A
$\dot{n}_{A1} = c_{A1} q_1$	$\dot{n}_{A1}$	c_{A1}	
$\dot{n}_{B2} = c_{B2} q_2$	$\dot{n}_{B2}$	c_{B2}	
$\dot{n}_{j3} = c_{j3} q_3,\ j = A, B, C$	$\dot{n}_{j3}$		
$c_{j3} = n_{j3}/V,\ \ j = A, B, C$	c_{j3}		
$r_1 = k_1 c_{A3}$	r_1		k_1
$r_2 = k_{2f} c_A c_B^2 - k_{2r} c_C$	r_2		k_{2f}, k_{2r}

Consider the following system



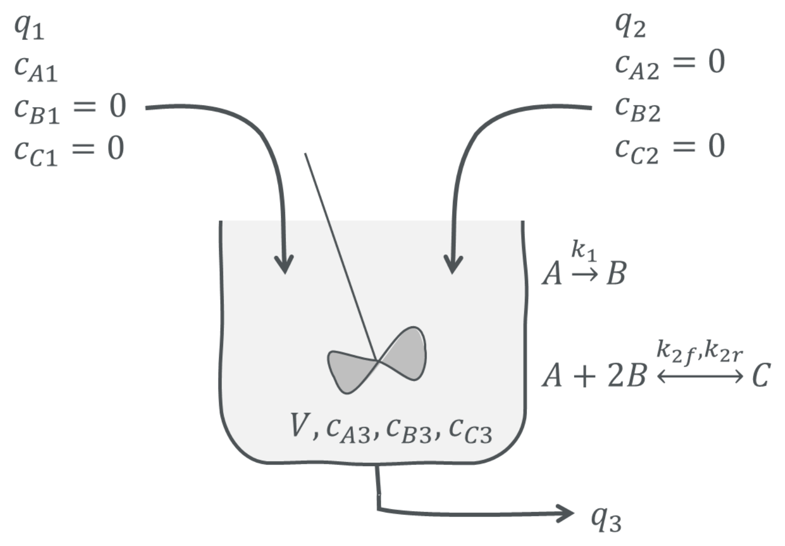


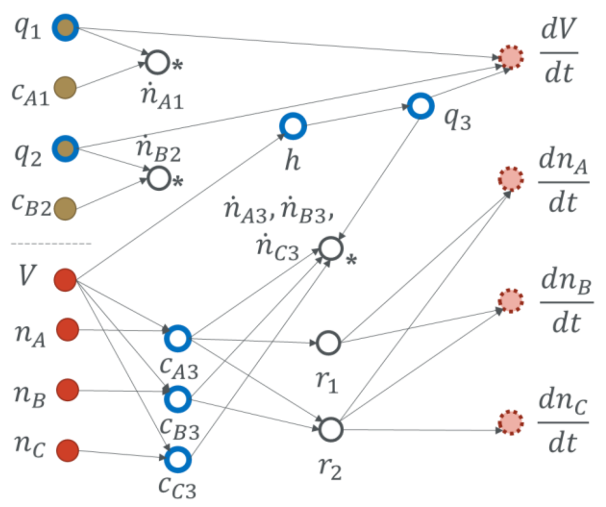
Reactions can be written as follows:

$$
\dot{\mathbf{s}} = \begin{bmatrix} \text{Generation}|\text{consumption of } A \\ \text{Generation}|\text{consumption of } B \\ \text{Generation}|\text{consumption of } C \end{bmatrix}
= \begin{bmatrix} -r_1 - r_2 \\ +r_1 - 2r_2 \\ 0 + 1r_2 \end{bmatrix}
= \begin{bmatrix} \nu_{A1} & \nu_{B1} & \nu_{C1} \\ \nu_{A2} & \nu_{B2} & \nu_{C2} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}
$$

$$
= \mathrm{N}\mathbf{r}
$$

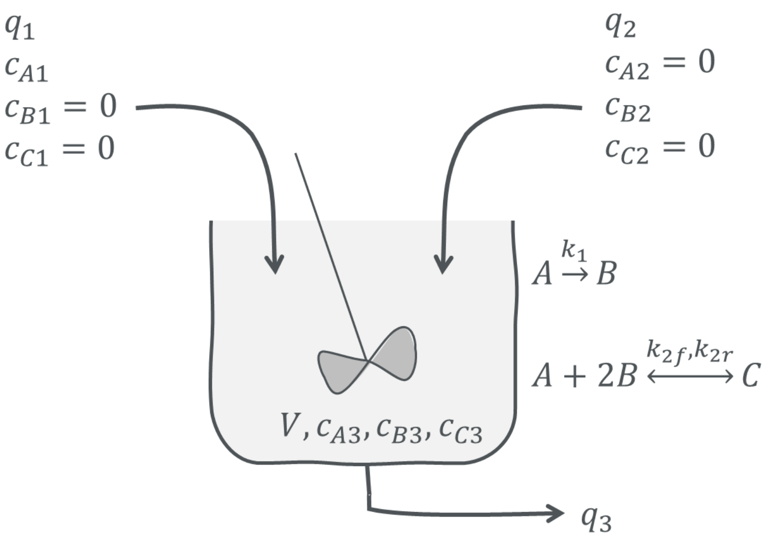
Consider the fol

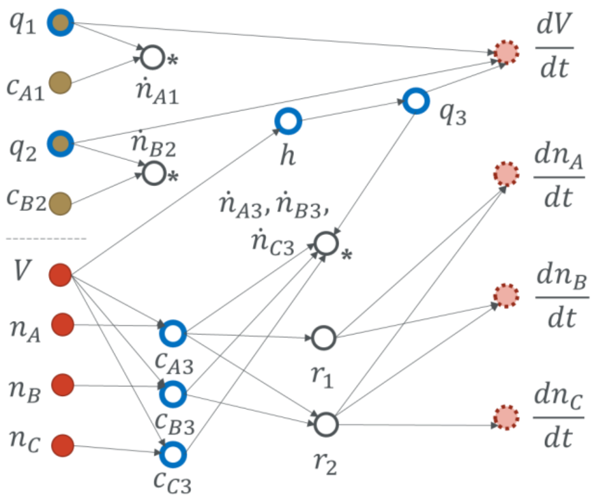




Equation	$\boldsymbol{x}$	$\boldsymbol{u}$	$\boldsymbol{p}$
$\frac{dV}{dt} = q_1 + q_2 - q_3 + \dot{s}_1$	V	q_1, q_2	
$\frac{dn_A}{dt} = \dot{n}_{A1} - \dot{n}_{A3} + \dot{s}_2$	n_A		
$\frac{dn_B}{dt} = \dot{n}_{B2} - \dot{n}_{B3} + \dot{s}_3$	n_B		
$\frac{dn_C}{dt} = -\dot{n}_{C3} + \dot{s}_4$	n_C		
$q_3 = c_V\sqrt{h}$	q_3		c_V
$h = V/A$	h		A
$\dot{n}_{A1} = c_{A1}q_1$	$\dot{n}_{A1}$	c_{A1}	
$\dot{n}_{B2} = c_{B2}q_2$	$\dot{n}_{B2}$	c_{B2}	
$\dot{n}_{j3} = c_{j3}q_3,\ j = A,B,C$	$\dot{n}_{j3}$		
$c_{j3} = n_{j3}/V,\ \ j = A,B,C$	c_{j3}		
$r_1 = k_1 c_{A3}$	r_1		k_1
$r_2 = k_{2f}c_A c_B^2 - k_{2r}c_C$	r_2		k_{2f}, k_{2r}
$\dot{\mathbf{s}} = \mathrm{N}\mathbf{r}$	$\dot{\mathbf{s}}$		N

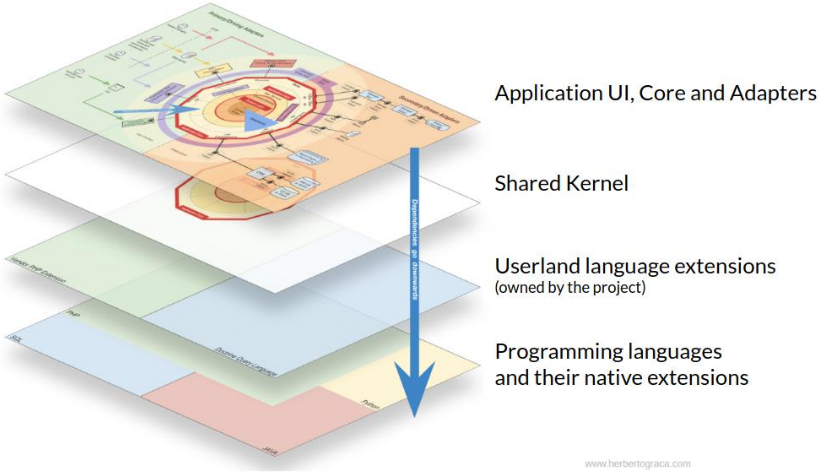
Consider the fol





Equation	x	u	p
$\frac{dV}{dt} = q_1 + q_2 - q_3 + \dot{s}_1$	V	q_1, q_2	Measured
$\frac{dn_A}{dt} = \dot{n}_{A1} - \dot{n}_{A3} + \dot{s}_2$	n_A		
$\frac{dn_B}{dt} = \dot{n}_{B2} - \dot{n}_{B3} + \dot{s}_3$	n_B		
$\frac{dn_C}{dt} = -\dot{n}_{C3} + \dot{s}_4$	n_C		
$q_3 = c_V\sqrt{h}$	q_3	Measured	c_V
$h = V/A$	h		A
$\dot{n}_{A1} = c_{A1}q_1$	$\dot{n}_{A1}$	c_{A1}	
$\dot{n}_{B2} = c_{B2}q_2$	$\dot{n}_{B2}$	c_{B2}	
$\dot{n}_{j3} = c_{j3}q_3,\ j = A, B, C$	$\dot{n}_{j3}$		
$c_{j3} = n_{j3}/V,\ \ j = A, B, C$ Measured	c_{j3}		
$r_1 = k_1 c_{A3}$	r_1	Unknown	k_1
$r_2 = k_{2f} c_A c_B^2 - k_{2r} c_C$	r_2		k_{2f}, k_{2r}
$\dot{\mathbf{s}} = \mathrm{N}\mathbf{r}$	$\dot{\mathbf{s}}$		N

Correct level of code complexity



https://herbertograca.com/2019/06/05/reflecting-architecture-and-domain-in-code/

Correct level of code complexity

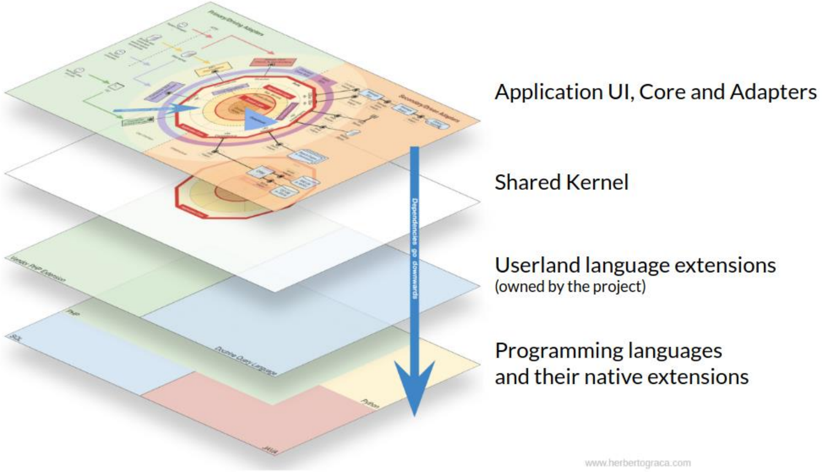
Spaghetti code

Middle ground

Code must be

- Easy to read
- Easy to reuse and modify

Deployed application



https://herbertograca.com/2019/06/05/reflecting-architecture-and-domain-in-code/

Implementing the model

- Create file “MAIN_System_of_ODEs.m”
- Describe and initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure

Implementing the model

```
%% System of ODEs: CSTR with two reactions
% Tobi Louw, 2022-03-02
% This code is used to illustrate the implementation of ODEs in MATLAB
% The example system is a CSTR facilitating two reactions, described in
% the accompanying PDF
% https://github.com/tmlouw/Introduction-to-ODEs
clc
clear
clf

%% Define time region of interest
t = linspace(0, 1200); % s, Time over which to perform integration

%% Define parameters
p.cV = 0.045; % m2.5/s, Outlet flowrate coefficient
p.A =
```

$c_V = 0.045\ \mathrm{m}^{2.5}.\mathrm{s}^{-1}$

$A = 2\ \mathrm{m}^2$

$k_1 = 0.05\ \mathrm{s}^{-1}$

$k_{2f} = 2.5\ \mathrm{m}^6.\mathrm{mol}^{-2}.\mathrm{s}^{-1}$

$k_{2r} = 0.05\ \mathrm{s}^{-1}$

Implementing the model

```
%% Define parameters
p.cV = 0.045;   % m2.5/s, Outlet flowrate coefficient
p.A = 2;        % m2, Cross-sectional area of reactor
p.k1 = 5.0e-2;  % 1/s, Reaction 1 rate constant
p.k2f = 2.5e+0; % m6/mol2.s, Reaction 2 forward rate constant
p.k2r = 5.0e-2; % 1/s, Reaction 2 reverse rate constant

% Matrix of stoichiometric coefficients for each state variable
p.Nu.V  = [ 0 0];
p.Nu.nB = [+1 -2];
p.Nu.nA = [-1 -1];
p.Nu.nC = [ 0 +1];
```

Implementing the model

- Create file “MAIN_System_of_ODEs.m”
- Initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure
- **Provide exogeneous inputs as functions**

Implementing the model

q_1 remains constant, add “`0*t`” to ensure output is a vector with same length as “`t`”

```
%% Define exogeneous inputs
u.q1 = @(t) 0.02 + 0*t;                % m3/s, Inlet flowrate 1 (constant)
u.q2 = @(t) 0.01 + 0.05*(t > 400);    % m3/s, Inlet flowrate 2 (step-change)
u.cA1 = @(t) 1.50 + 0.50*(t > 800); % mol/m3, Inlet concentration A (step-change)


% Concentration cB2 will vary stochastically
temp(1) = 0;
for i = 2:length(t)
       temp(i) = 0.8*temp(i-1) + 0.05*randn;
end
temp = temp + 2.0;
u.cB2 = griddedInterpolant(t, temp);
```

q_2, c_{A1} undergoes a step-change

c_{B2} is stochastic, autocorrelated and normally distributed around 2.0

“`griddedInterpolant`” converts an array of times and values to an anonymous function with time as an input

Implementing the model

- Create file “MAIN_System_of_ODEs.m”
- Initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure
- Provide exogeneous inputs as functions
- **Define state structure and provide initial conditions**

```matlab
%% Define state structure and initial conditions
p.state_fields = {'V', 'nA', ‘nC', ‘nB'}; % Field names for each state
x0.V = 0.9;    % m3, initial tank volume
x0.nA = 0.15;  % mol/m3, initial concentration of A
x0.nB = 0.25;  % mol/m3, initial concentration of A
x0.nC = 0.30;  % mol/m3, initial concentration of A
x0_vec = xS2xV(x0, p.state_fields);
```

Define initial conditions as a structure.

```matlab
function xV = xS2xV(xS, fields)
% Map all elements in structure "xS" and indexed by "fields"
% to the corresponding element in the vector "xV"
n = length(fields);
for i = 1:n
       xV(i,:) = xS.(fields{i});
end
```

Create a function to convert the structure to a vector (for use with MATLAB built-in functions such as ode45).

```matlab
function xS = xV2xS(xV, fields)
% Maps all elements in vector "xV" to the structure "xS",
% using the elements in "fields" as field names.
n = length(fields);
for i = 1:n
       xS.(fields{i}) = xV(i,:);
end
```

Create a function to convert the vector to the corresponding structure, for use inside functions for improved readability

Implementing the model

- Create file “MAIN_System_of_ODEs.m”
- Initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure
- Provide exogeneous inputs as functions
- Define state structure and provide initial conditions
- **Create a function that calculates all <u>intermediate variables</u>, given time, state variables and exogeneous inputs**

```
function v = CalculateIntermediates(t, x, u, p)
% Calculate concentrations in CSTR
v.cA3 = x.nA ./ x.V; % mol/m3, concentration of A
v.cB3 = x.nB ./ x.V; % mol/m3, concentration of B
v.cC3 = x.nC ./ x.V; % mol/m3, concentration of C

% Calculate all flowrates into / out of CSTR
v.h = x.V/p.A; % m, liquid level in CSTR
v.q3 = p.cV*sqrt(v.h); % m3/s, flowrate out of CSTR
v.nA1 = u.q1(t).*u.cA1(t); % mol/s, molar flowrate of A into CSTR
v.nB2 = u.q2(t).*u.cB2(t); % mol/s, molar flowrate of B into CSTR
v.nA3 = v.q3.*v.cA3; % mol/s, molar flowrate of A out of CSTR
v.nB3 = v.q3.*v.cB3; % mol/s, molar flowrate of B out of CSTR
v.nC3 = v.q3.*v.cC3; % mol/s, molar flowrate of C out of CSTR

% Calculate reaction rates and source terms for each state variable
r(1,:) = p.k1*v.cA3; % mol/m3.s, reaction rate 1
r(2,:) = p.k2f*v.cA3.*v.cB3.^2 - p.k2r*v.cC3; % mol/m3.s, reaction rate 2

Nu = xS2xV(p.Nu, p.state_fields); % Convert structured coefficients to vector
S_vec = Nu*r; % Vector of source terms
v.S = xV2xS(S_vec, p.state_fields);
```

Notice how “xS2xV” can be used on any structure with the fields contained in “p.state_fields”.

Here, it is used to convert the stoichiometric coefficients into a matrix

Implementing the model

- Create file “MAIN_System_of_ODEs.m”
- Initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure
- Provide exogeneous inputs as functions
- Define state structure and provide initial conditions
- Create a function that calculates all <u>intermediate variables</u>, given time, state variables and exogeneous inputs
- **Create a function that calculates the <u>derivative of the state variables,</u> given time, state variables and exogeneous inputs**

Implementing the model

```
function dxdt = SystemODEs(t, x_vec, u, p)
% Calculate the time-derivative of all state variables

% Map state vector to structure and calculate intermediate variables
x = xV2xS(x_vec, p.state_fields);
v = CalculateIntermediates(t, x, u, p);

% Calculate state derivatives as structure
ddt.V  = u.q1(t) + u.q2(t) - v.q3 + v.S.V;
ddt.nA = v.nA1   - v.nA3          + v.S.nA;
ddt.nB = v.nB2   - v.nB3          + v.S.nB;
ddt.nC = 0       - v.nC3          + v.S.nC;

% Map state derivative structure to vector
dxdt = xS2xV(ddt, p.state_fields);
```

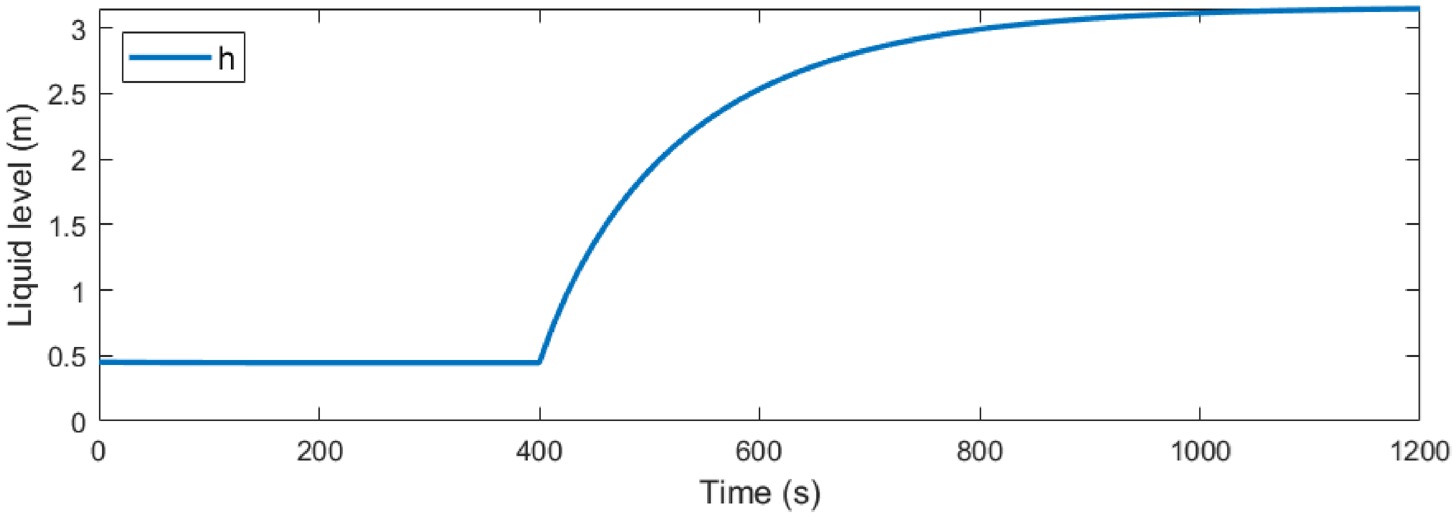
Implementing the model

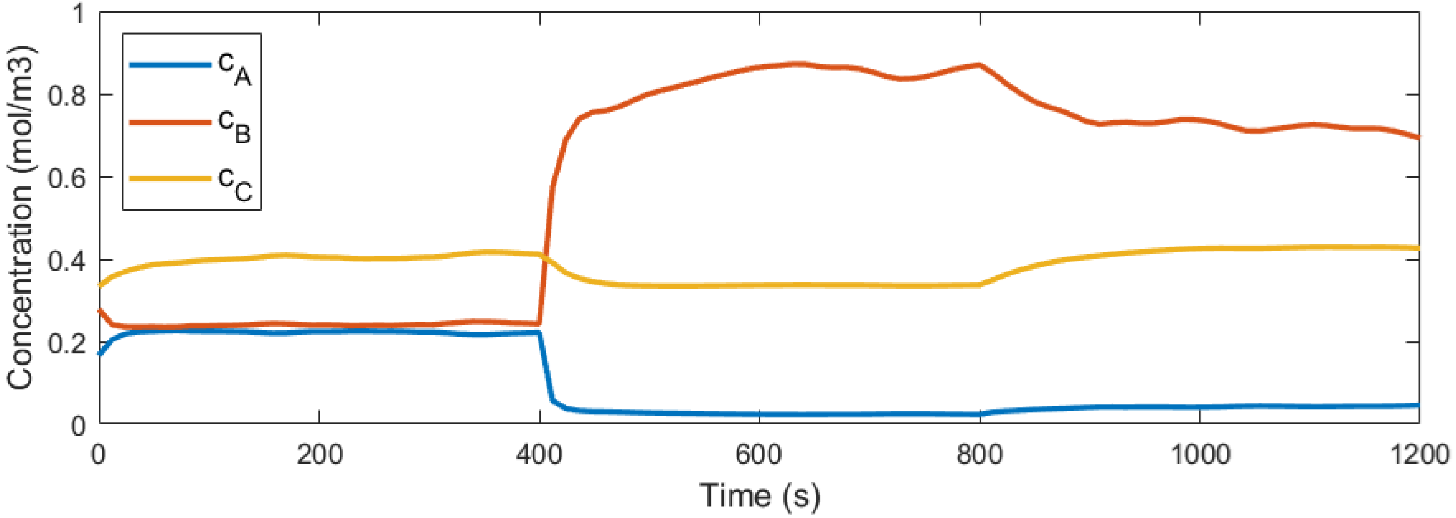
- Create file “MAIN_System_of_ODEs.m”
- Initialize
- Provide time vector (from 0 to 1200 s)
- Provide parameter values as structure
- Provide exogeneous inputs as functions
- Define state structure and provide initial conditions
- Create a function that calculates all <u>intermediate variables</u>, given time, state variables and exogeneous inputs
- Create a function that calculates the <u>derivative of the state variables,</u> given time, state variables and exogeneous inputs
- **Simulate system and plot results**

Implementing the model

```
<in MAIN_System_of_ODEs>

%% Simulate system of ODEs
[~, x_vec] = ode45(@(t, x) SystemODEs(t, x, u, p), t, x0_vec);
x = xV2xS(x_vec', p.state_fields);
v = CalculateIntermediates(t, x, u, p);

%% Plot simulation results
tiledlayout flow
ax_height = nexttile;
plot(t, v.h, 'LineWidth',2);
legend('h','Location','northwest');

ax_concentration = nexttile;
plot(t, v.cA3, t, v.cB3, t, v.cC3, 'LineWidth', 2)
legend('c_A', 'c_B', 'c_C','Location','northwest’)
```

Implementing the model





Adding measurements

- Running simulation = running plant
- **Simulate measurement instrumentation**

Adding measurements

```
<in MAIN_System_of_ODEs>

%% Define measurement noise, frequency and delay
% Create measurement structures:
%   fields: names of measurements
%   func: function describing how measurement is calculated
%   var: assume gaussian noise with variance "var"
%   T: measurement period T = 1/frequency
%   D: measurement delay y ~ y(t - D)
meas.fields = {'h', 'cC3'};

meas.h   = struct('func', @(t, x, u, v, p) x.V/p.A,   'var', 0.1,  'T', 5,  'D', 2);
meas.cC3 = struct('func', @(t, x, u, v, p) x.nC./x.V, 'var', 0.02, 'T', 60, 'D', 60);
```

Adding measurements

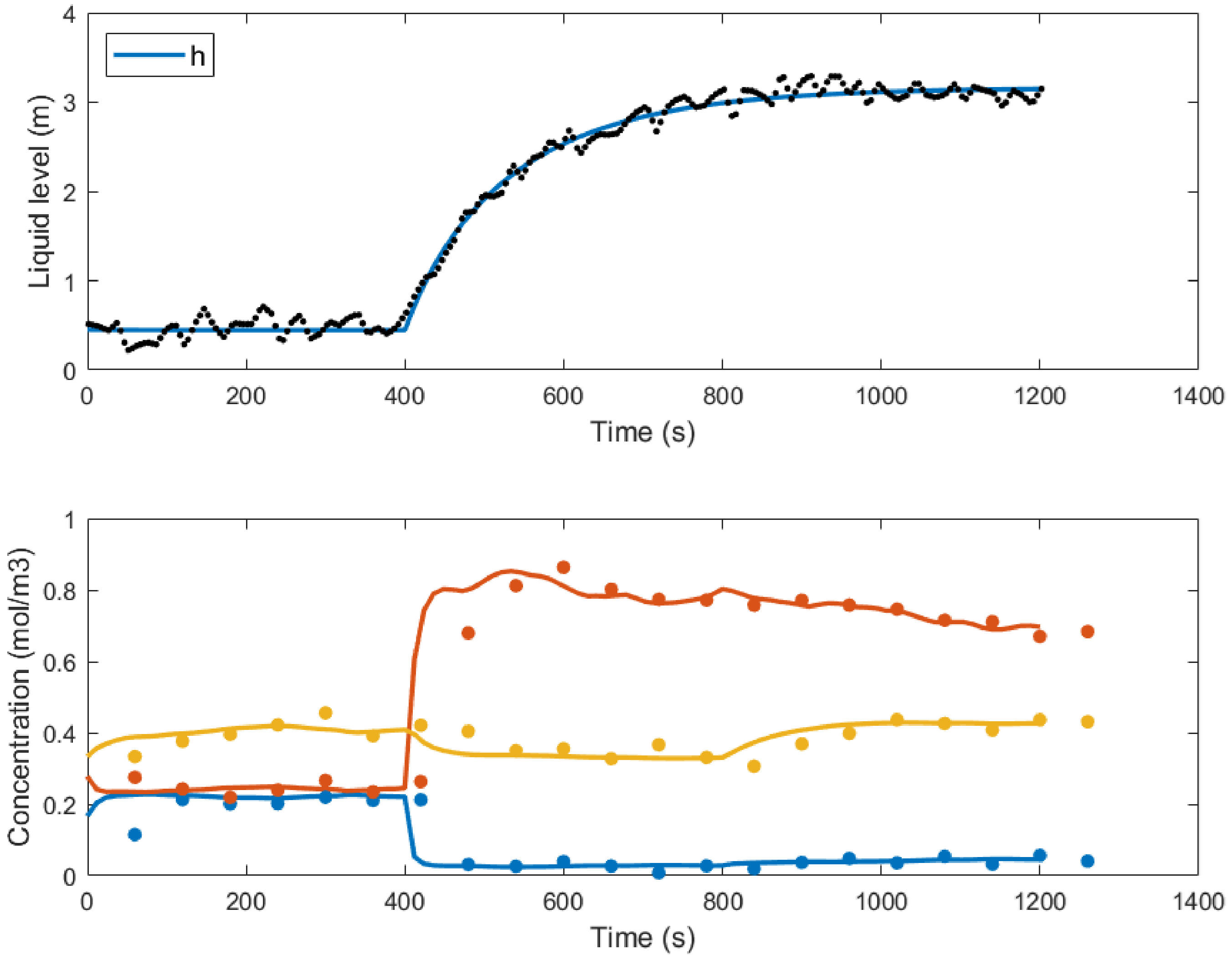
```
<in MAIN_System_of_ODEs>

%% Define measurement noise, frequency and delay
% Create measurement structures:
% fields: names of measurements
% var: assume gaussian noise with variance "var"
% T: measurement period T = 1/frequency
% D: measurement delay y ~ y(t - D)
meas.fields = {'h', 'cC3'};

mea
mea
```

```
function y = Measurements(t, x, u, v, p, meas)
% Calculate measurement values for each field in "meas"
for i = 1:length(meas.fields)
    current = meas.(meas.fields{i});
    values = current.func(t, x, u, v) + current.var*randn(size(t));
    times = 0 : current.T : t(end);
    interp_values = interp1(t, values, times);
    y.(meas.fields{i}) = timeseries(interp_values, times + current.D);
end
```

Adding measurements

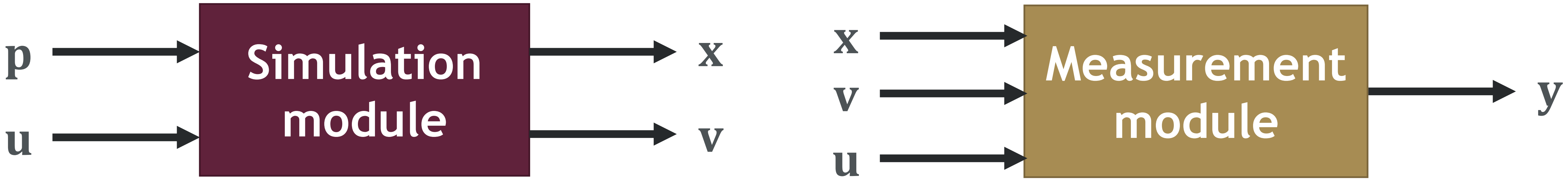
```
<in MAIN_System_of_ODEs>

% Record measurements
y = Measurements(t, x, u, v, p, meas);

%% Plot measurements
axes(ax_height)
hold on
plot(y.h,'k.','MarkerSize', 8)
legend('h', 'Location','northwest')

axes(ax_concentration)
hold on
plot(y.cC3,’k.','MarkerSize',20)
legend('c_C’, 'Location’, 'northwest')
```

Adding measurements



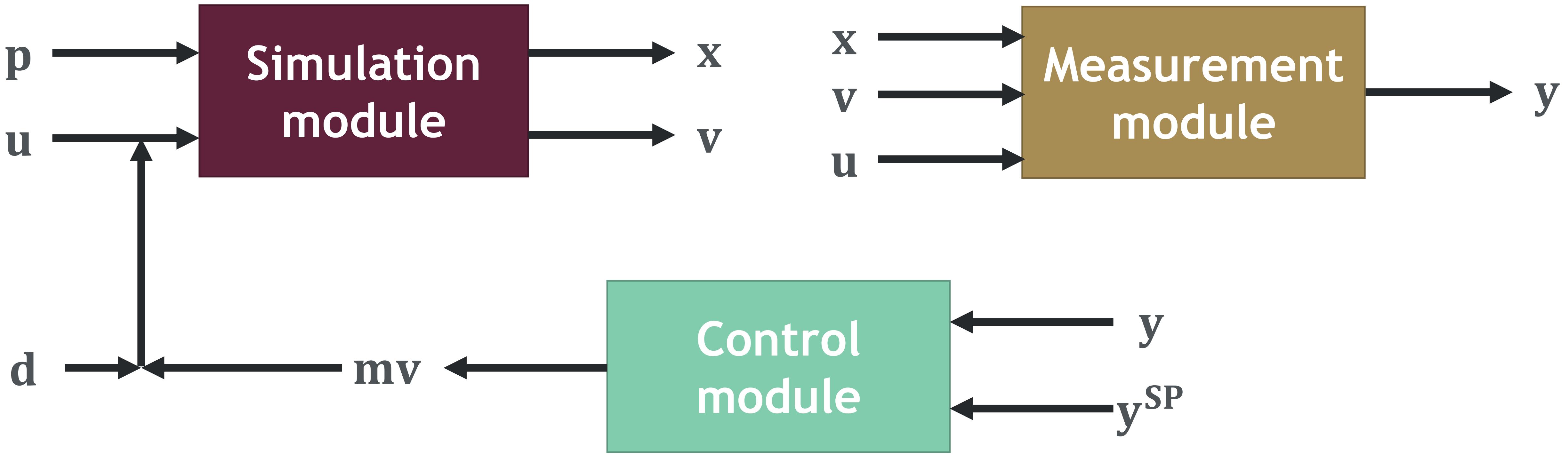
Adding measurements

- Running simulation = running plant
- **Simulate measurement instrumentation**



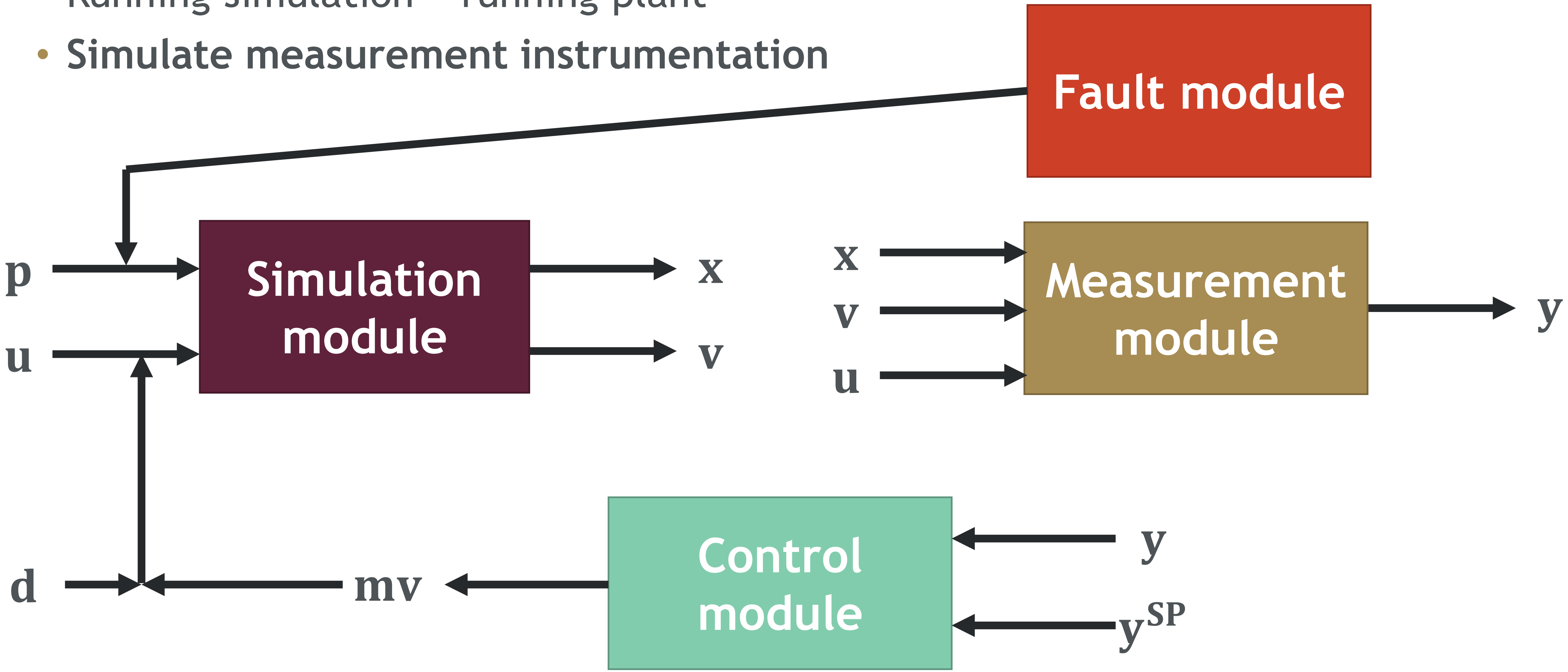
Adding measurements and control

- Running simulation = running plant
- **Simulate measurement instrumentation**



Adding measurements, control and faults…

- Running simulation = running plant
- **Simulate measurement instrumentation**



Repository

- https://github.com/tmlouw/Introduction-to-ODEs