

Systems Biology

Simulation of Dynamic Network States



Lecture #3

Simulating Dynamic Mass Balances

Outline

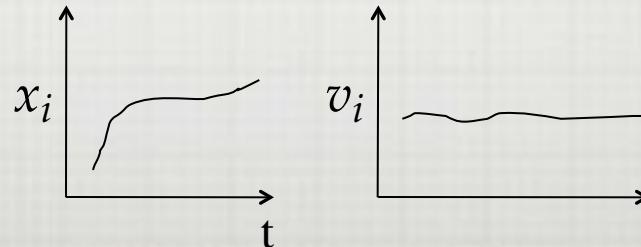
- Basic simulation procedure
- Graphical representation
- Post-processing the solution
- Multi-scale representation

BASIC PROCEDURE

Dynamic Simulation

1. Formulate the mass balances: $\frac{dx}{dt} = Sv(x; k)$
2. Specify a) the numerical values for k and b) ICs
 $k_i = \text{value}; \text{ I.C. } x_{i0} = x_i(t = 0)$
3. Obtain the numerical solution
Run a software package
4. Analyze the results

Graphically:



Typical Workbook: details to follow

STEP 1

Set up the equations

```

Step 1: Set up the equations


$$\begin{aligned} & \text{Let } x_1 = \text{magenta}, x_2 = \text{cyan}, x_3 = \text{yellow}, x_4 = \text{black} \\ & \text{1. } \text{magenta} + \text{cyan} + \text{yellow} + \text{black} = 100 \\ & \text{2. } \text{cyan} + \text{yellow} + \text{black} = 70 \\ & \text{3. } \text{magenta} + \text{yellow} + \text{black} = 80 \\ & \text{4. } \text{cyan} + \text{black} = 60 \\ \\ & \text{5. } \text{magenta} = 100 - \text{cyan} - \text{yellow} - \text{black} \\ & \text{6. } \text{cyan} = 100 - \text{magenta} - \text{yellow} - \text{black} \\ & \text{7. } \text{yellow} = 100 - \text{magenta} - \text{cyan} - \text{black} \\ & \text{8. } \text{black} = 100 - \text{magenta} - \text{cyan} - \text{yellow} \end{aligned}$$


model1 = sm.formula('model1')
model2 = sm.formula('model2')
model3 = sm.formula('model3')
model4 = sm.formula('model4')

for model, name in zip([model1, model2, model3, model4],
                      ('magenta', 'cyan', 'yellow', 'black')):
    print(f"\n{model}\n{sm.OLS(name, name).fit().summary()}")

```

```

Step 2: Define parameter values

v1id = 1; v2kf = 0.1; v3kf = 0.001
v4kf = 1e-006; v5 = 2;
for nce_rate in model.rates.Demand:
    print("Var %s %s (%s, %s, %s)" % (nce_rate, nce_rate))

print("Model - parameter (%s)" %)
print("Model - parameter (%s)" %)

abc_v1kf(v1id) = v2kf/Req_v1
abc_v2kf(v2id) = v3kf/Req_v2
abc_v3kf(v3id) = v4kf/Req_v3
Req_v1 = 1000000.0; abc_v1kf(v1id) = abc_v1kf(Req_v1)
Req_v2 = 1000000.0; abc_v2kf(v2id) = abc_v2kf(Req_v2)
Req_v3 = 1000000.0; abc_v3kf(v3id) = abc_v3kf(Req_v3)
inf

```

Specify parameters

STEP 2

STEP 3

Solve

Step 3: Set up conditions and simulate

```

    sim = Simulation(model)
    sim.F = 0; sim.M = 0
    conc_pcl = sim.simulate_model(model, time=(t0, tf))

```

Step 4: Graph the results

```

116 = DLT.DILUTE(0,INITIALIZED_LAYOUT*TE0, INITIALIZE*10, 10)
117 = fig.add_subplot(gs[0, 4], aspect=1, height_ratio=[1, 1, 1, 1, 1, 1])
118 =
119 = fig.add_subplots(gs[0, 0])
120 = fig.add_subplots(gs[1, 0])
121 = fig.add_subplots(gs[2, 0])
122 = fig.add_subplots(gs[3, 0])

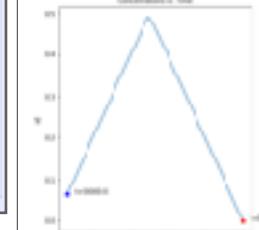
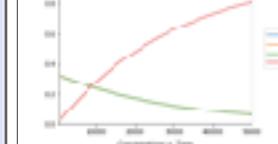
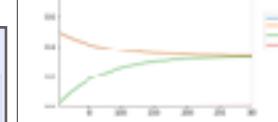
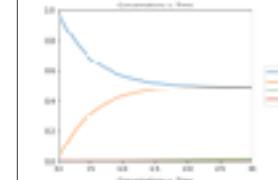
plot_simulation(ccrc_x0l, axccrl, legend="right outside",
                 xlim=(0, 3), ylim=(0, 1),
                 title="Concentrations vs. Time");
plot_simulation(ccrc_x0l, axccrl2, legend="right outside",
                 xlim=(0, 100), ylim=(0, 1),
                 title="Concentrations vs. Time");
plot_simulation(ccrc_x0l, axccrl3, legend="right outside",
                 xlim=(0, 10), ylim=(0, 1),
                 title="Concentrations vs. Time");
plot_phase_plane(ccrc_x0l, xlabel="x1", ylabel="x2",
                  xlabel="x1", ylabel="x2",
                  title="Phase Plane Plot");

```

Specify graphical output

STEP 4

Plot and export



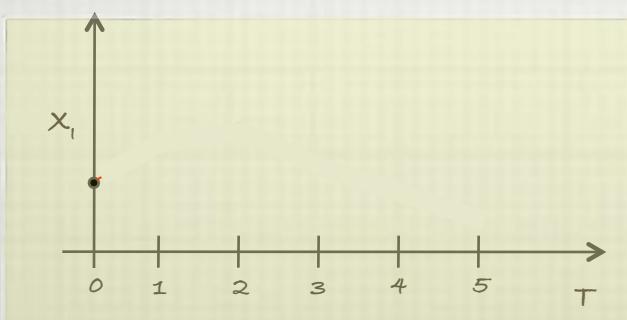
Export Theorem

100 - 2000

RESULTS

The Results of Simulation: a table of numbers

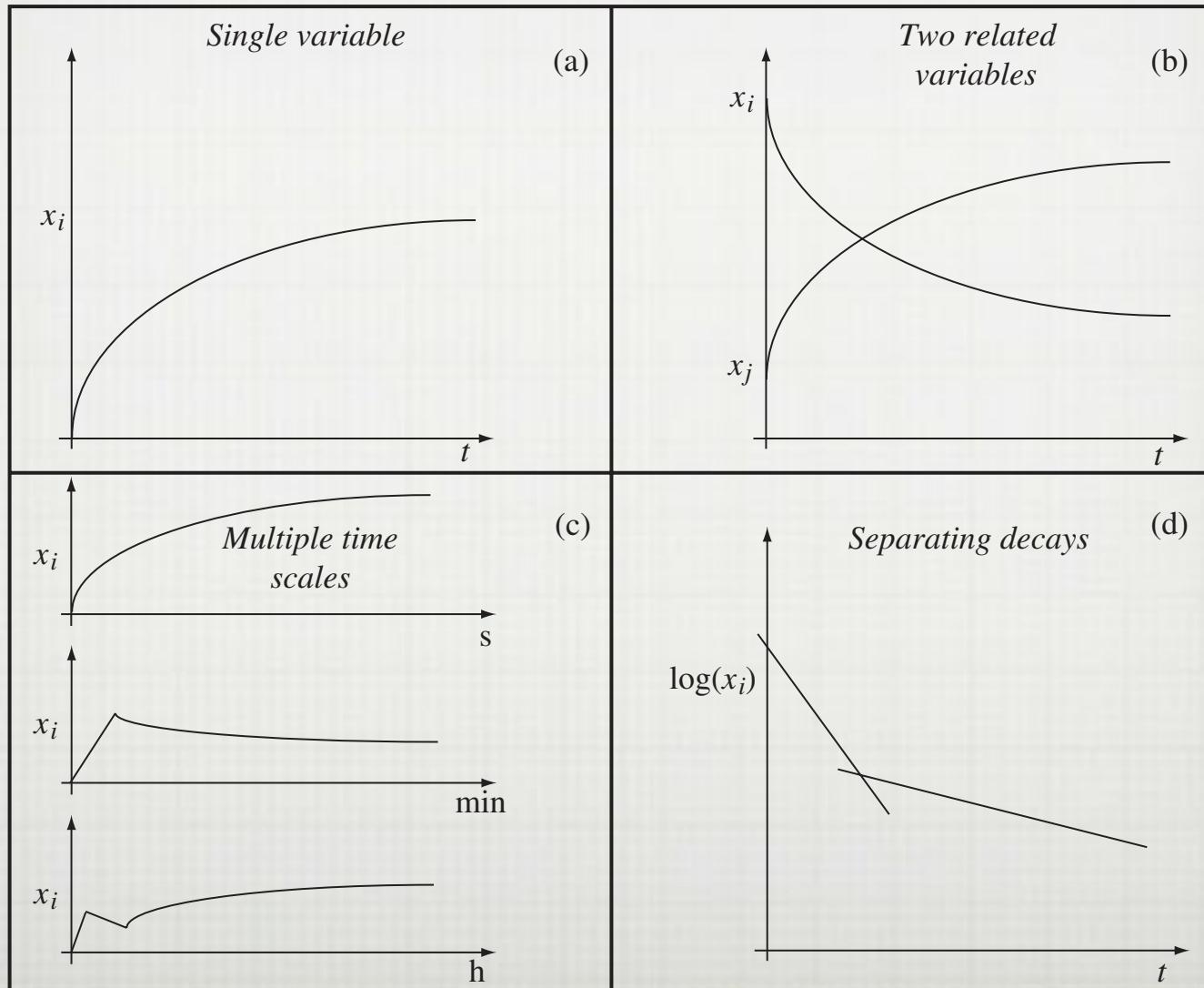
t	x_1	x_2	...	x_m
t_1	$x_1(t_1)$	$x_2(t_1)$		$x_m(t_1)$
t_2	$x_1(t_2)$	$x_2(t_2)$		$x_m(t_2)$
t_3	$x_1(t_3)$	$x_2(t_3)$		$x_m(t_3)$
\vdots	\vdots			\vdots
t_n	$x_1(t_n)$	$x_2(t_n)$		$x_m(t_n)$
t_{n+1}	$x_1(t_{n+1})$	$x_2(t_{n+1})$		$x_m(t_{n+1})$
t_{n+2}	$x_1(t_{n+2})$	$x_2(t_{n+2})$...	$x_m(t_{n+2})$



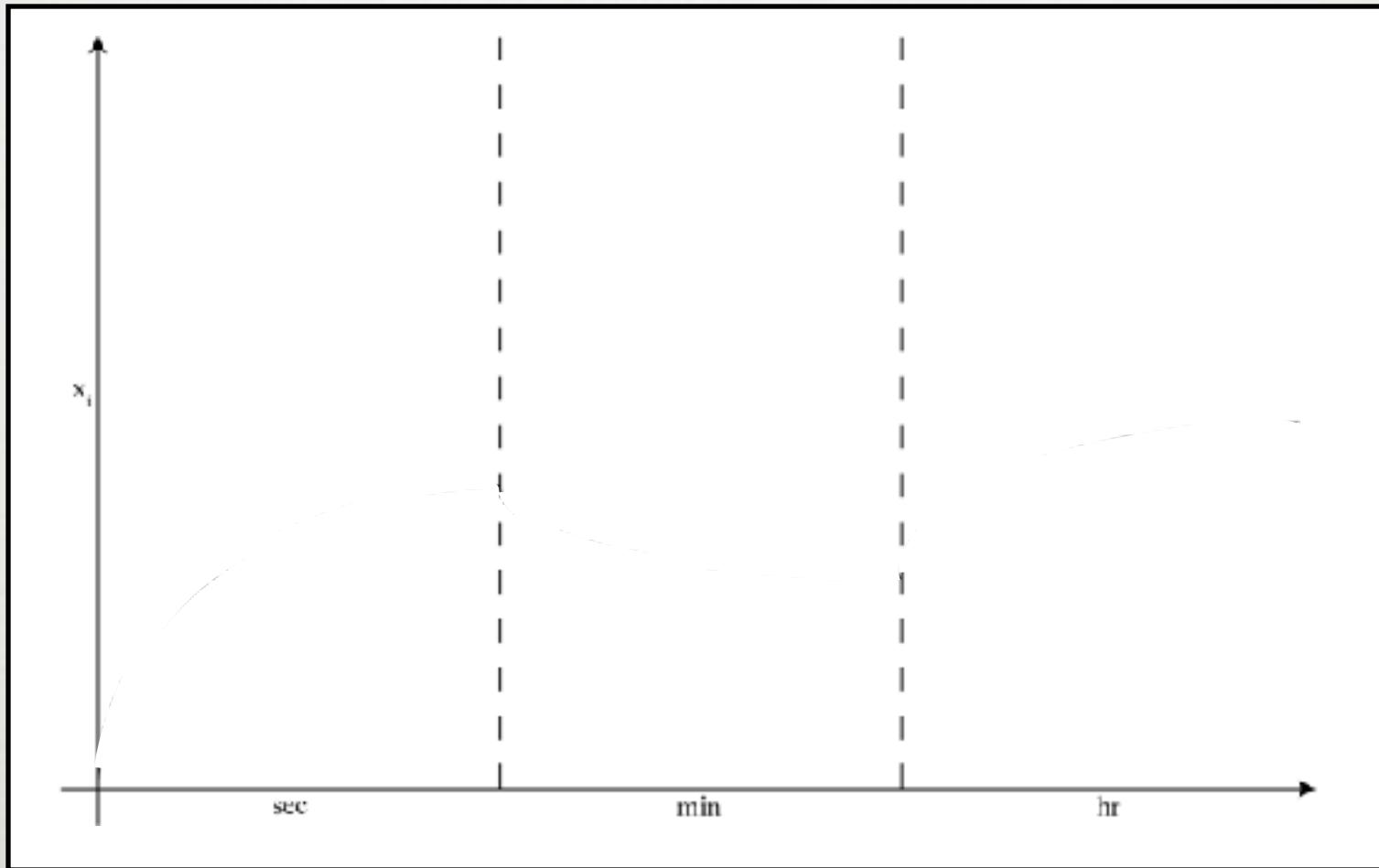
Can correlate variables, r^2
• off-set in time
• on a particular time scale

GRAPHICAL REPRESENTATION

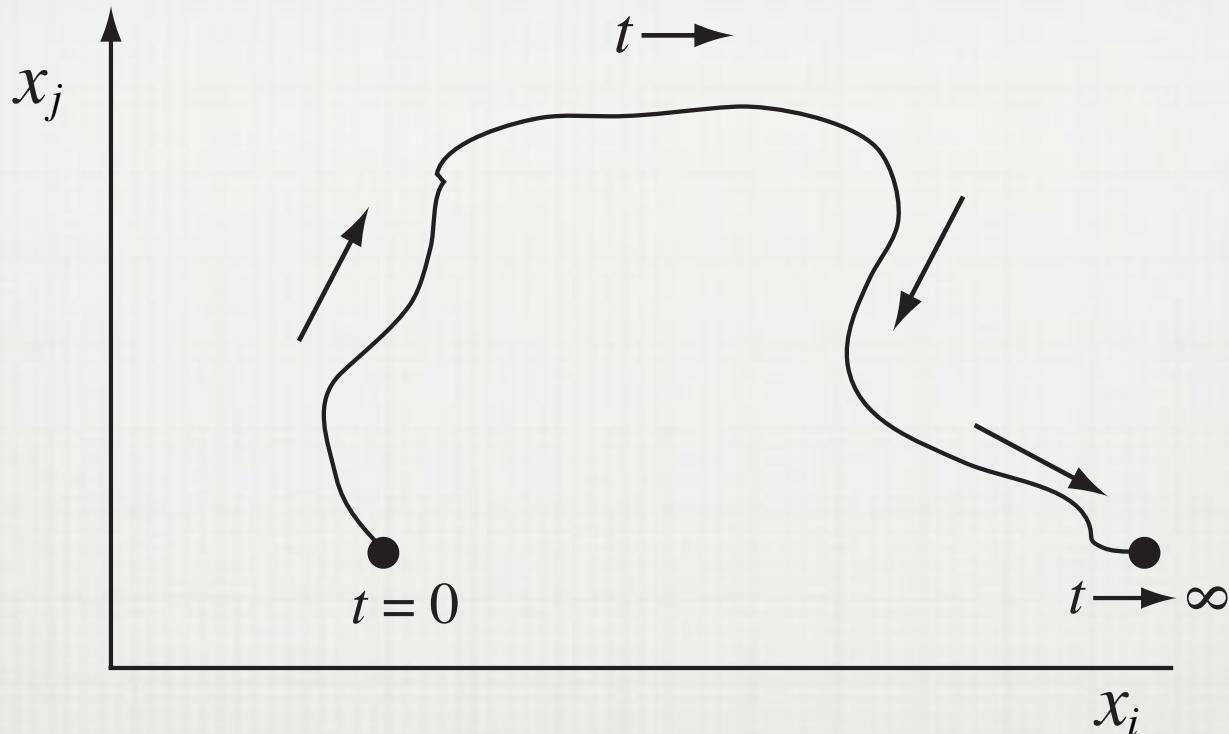
Example Graphical Representation of the Solution, $\mathbf{x}(t)$



Multi-time scale representation: segmenting the x-axis

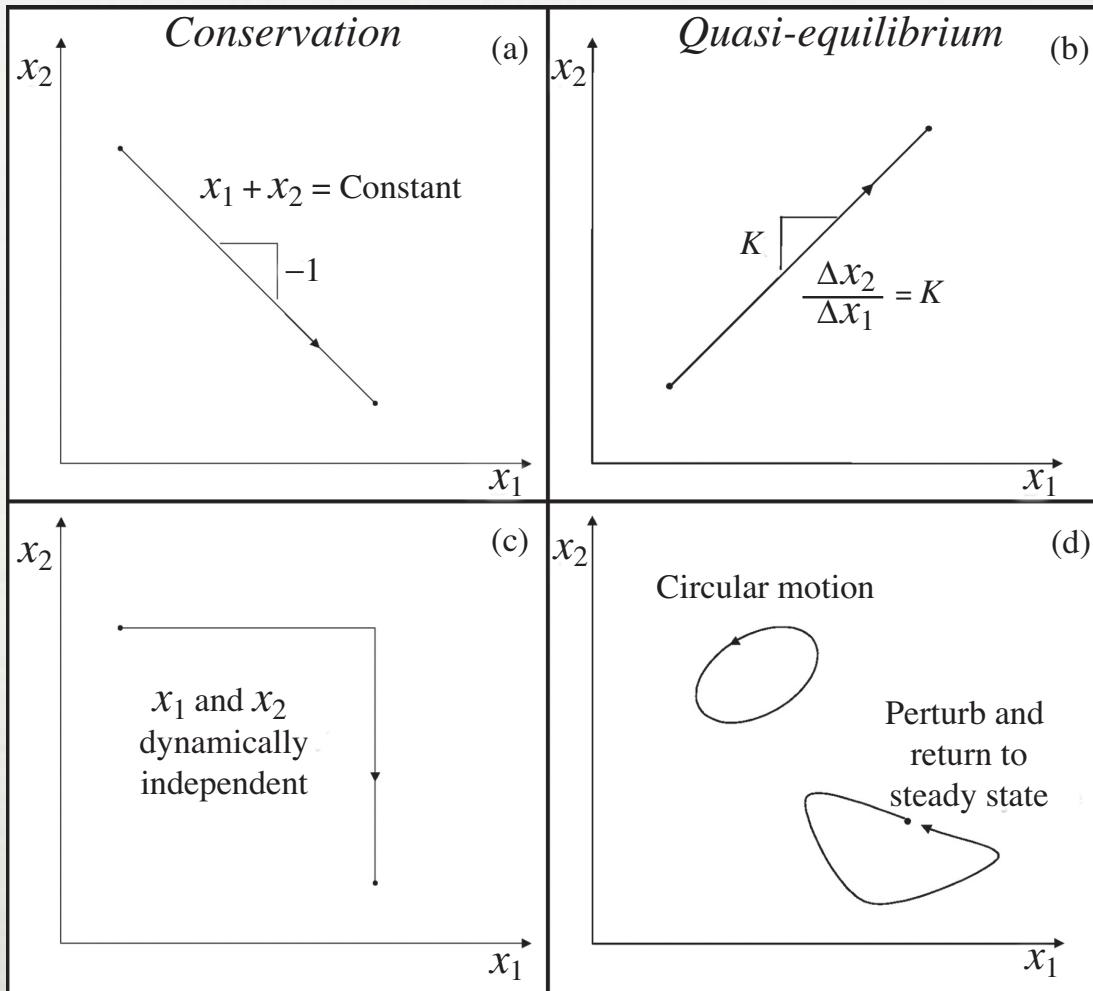


Dynamic Phase Portrait



Can also plot fluxes $v_i(t)$ and pools $p_i(t)$ in the same way;
this is a very useful representation of the results

Characteristic “Signatures” in the Phase Portrait

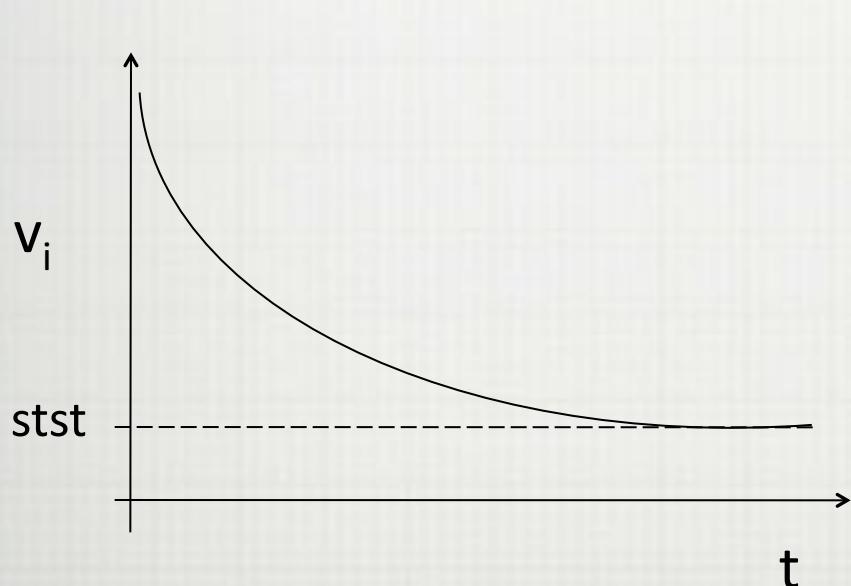


POST-PROCESSING THE SOLUTION

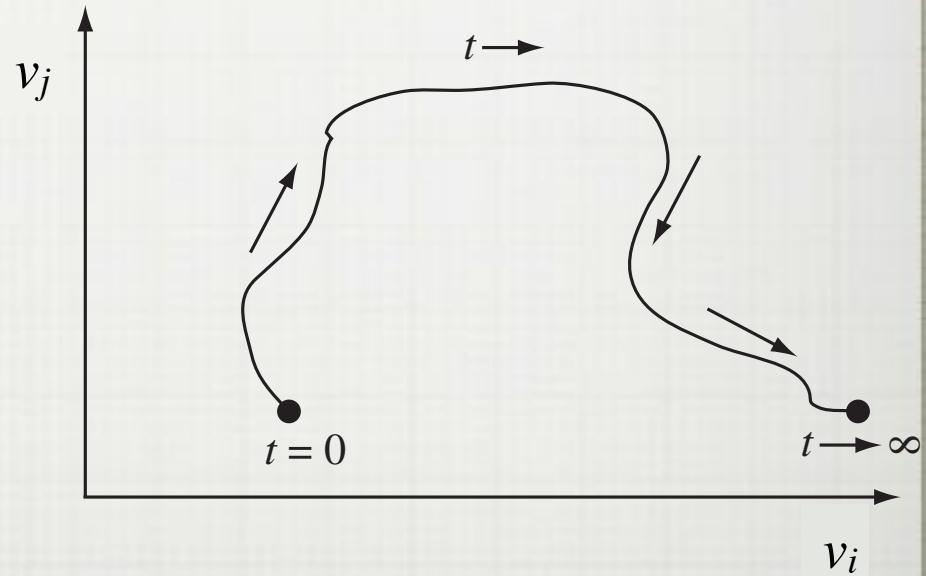
Post-processing the Solution

1. Computing the fluxes

$$v(t) = v(x(t))$$



dynamic response



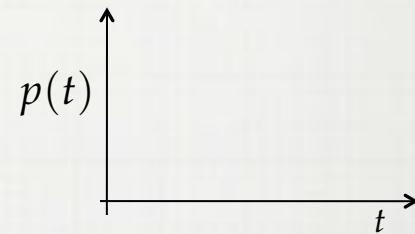
phase portrait

Post-processing the Solution

2. Forming pools; two examples

Compute “pools”: $p(t) = Px(t)$

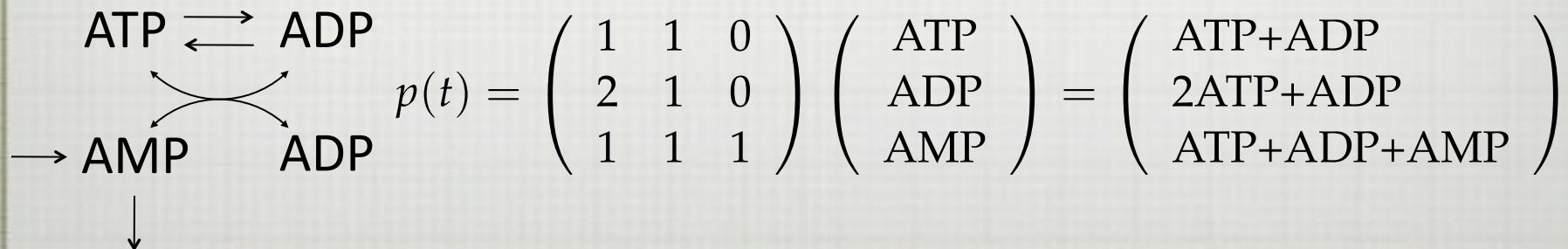
then plot



Example #1

$$x_1 \rightleftharpoons x_2 \quad K_{eq} = 1 \quad p(t) = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ x_1 - x_2 \end{pmatrix} = \begin{pmatrix} \text{total mass} \\ \text{dis-equil} \end{pmatrix}$$

Example #2



Post-processing the Solution

3. Computing auto correlations

Example 1

$$\sigma_{1,m} = \frac{1}{n} \sum_{k=1}^n (x_1(k) - \bar{x}_1)(x_m(k) - \bar{x}_m)$$

$n=3$
first segment

Example 2

$$\sigma_{1,m} = \frac{1}{n} \sum_{k=1}^n (x_1(k) - \bar{x}_1)(x_m(k) - \bar{x}_m)$$

$n=m$
entire range

Example 3

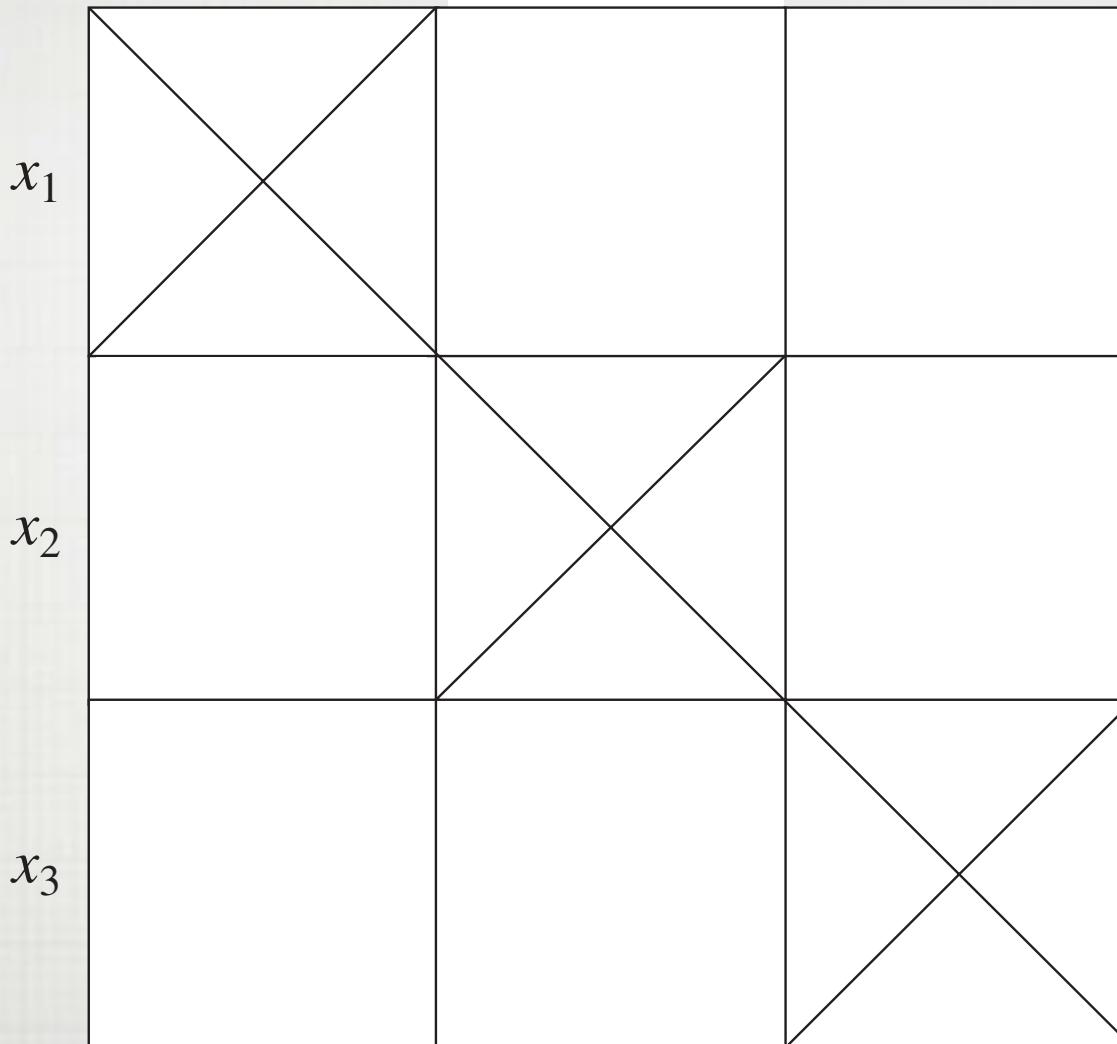
$$\sigma_{1,2} = \frac{1}{n} \sum_{k=\tau}^m (x_1(k) - \bar{x}_1)(x_2(k-\tau) - \bar{x}_2)$$

$n=m$
off-set in time, $\tau=2$

t	x_1	x_2	...	x_m
t_1	$x_1(t_1)$	$x_2(t_1)$		$x_m(t_1)$
t_2	$x_1(t_2)$	$x_2(t_2)$		$x_m(t_2)$
t_3	$x_1(t_3)$	$x_2(t_3)$		$x_m(t_3)$
\vdots	\vdots			\vdots
t_n	$x_1(t_n)$	$x_2(t_n)$		$x_m(t_n)$
t_{n+1}	$x_1(t_{n+1})$	$x_2(t_{n+1})$		$x_m(t_{n+1})$
t_{n+2}	$x_1(t_{n+2})$	$x_2(t_{n+2})$...	$x_m(t_{n+2})$

MULTI-SCALE REPRESENTATION

Tiling Phase Portraits



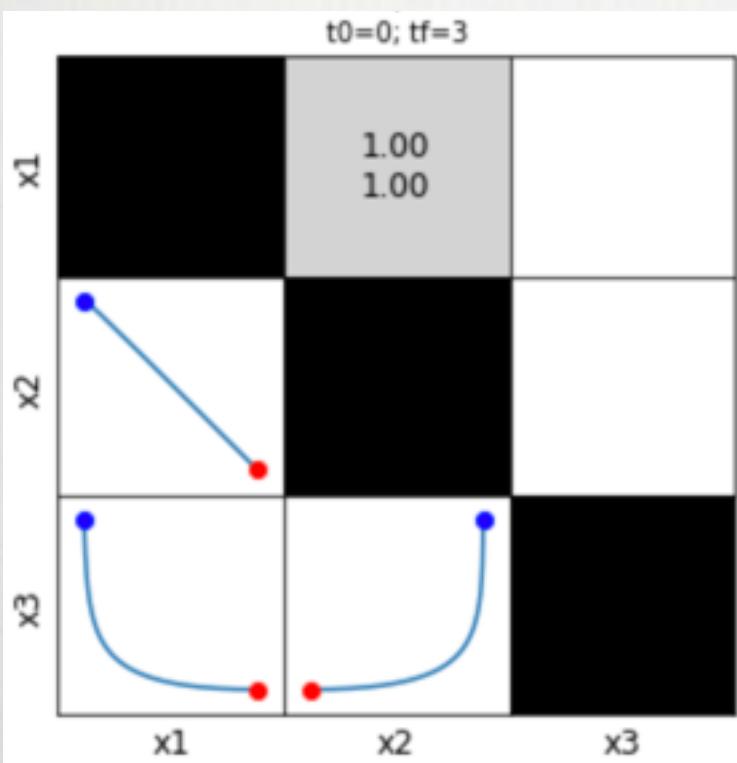
x_1 vs. x_2 same
as x_2 vs. x_1
→symmetric array

use corresponding
locations in array
to convey different
information
•graph
•statistics

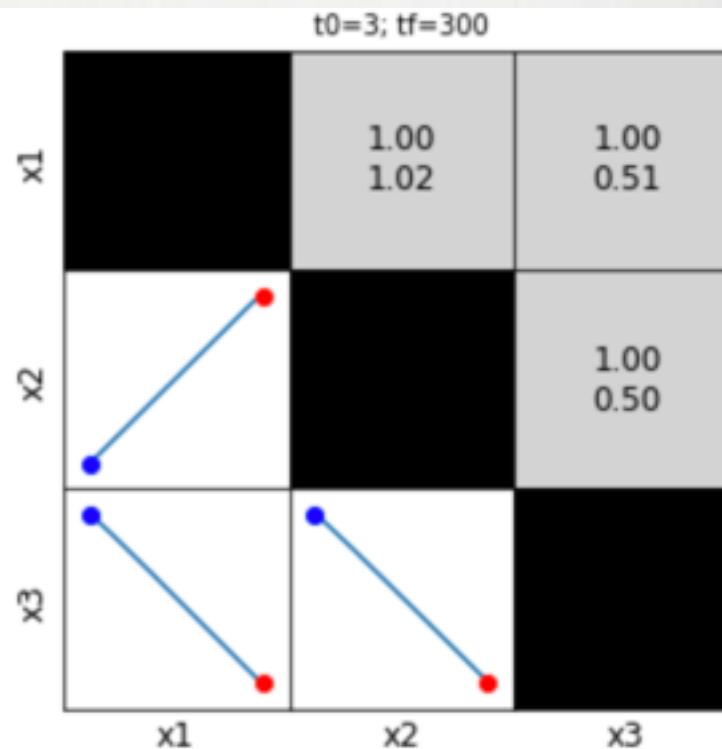
*A series of tiled phase portraits can be prepared,
one for each time scale of interest*

Representing Multiple Time-Scales: tiling phase portraits on separate time scales

Time: 0 -> 3 sec



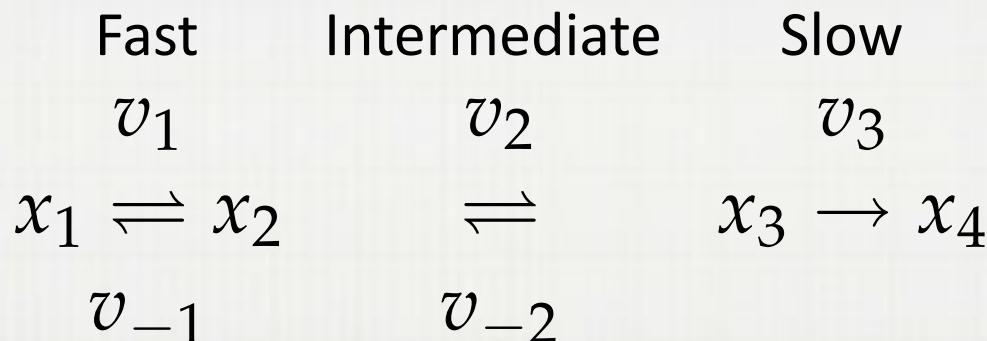
Time: 3 -> 300 sec



Red: t0

Blue: tf

Example System:



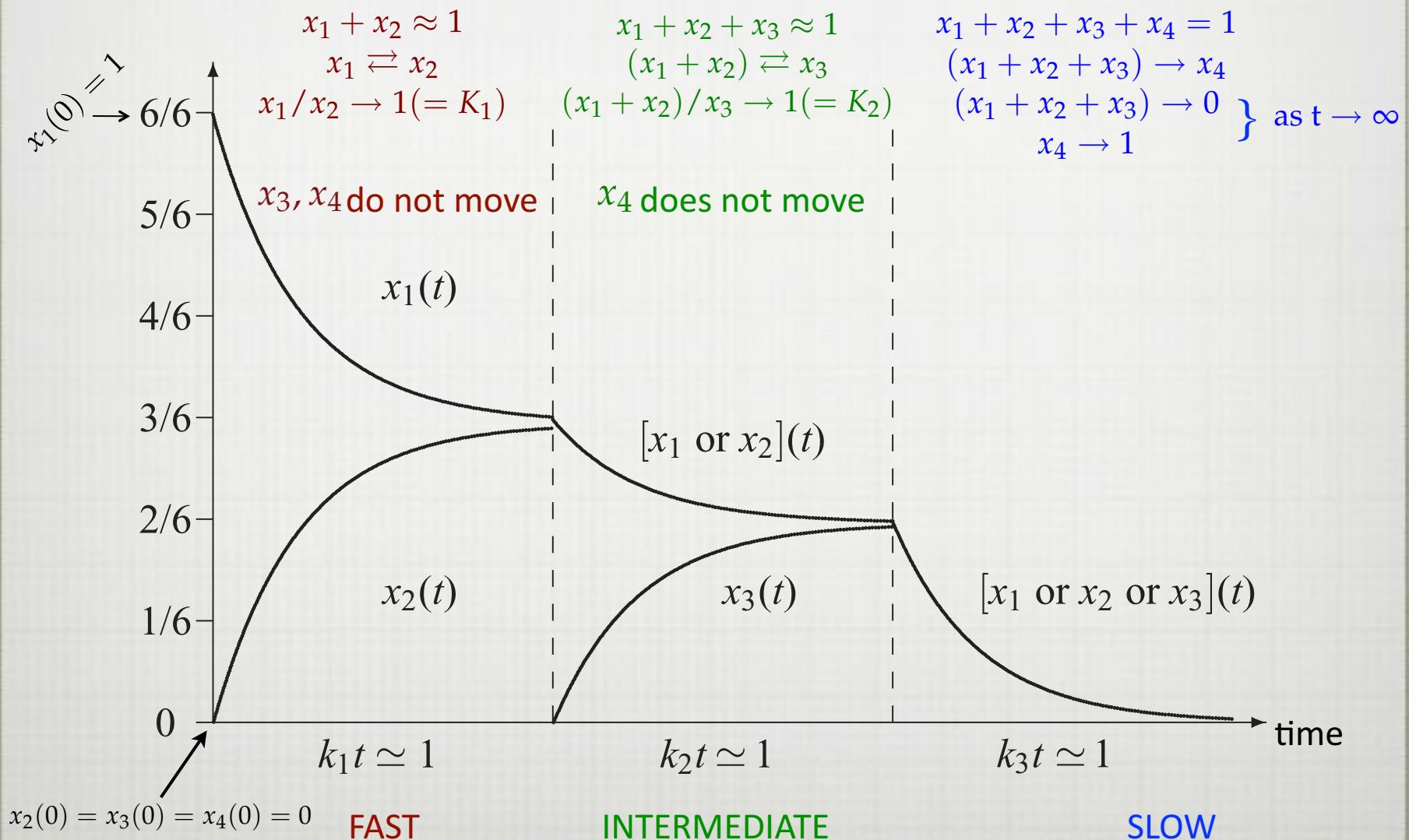
$$\frac{dx_1}{dt} = -v_1 + v_{-1} = -k_1 x_1 + k_{-1} x_2 \quad x_{10} = x_1(t=0)$$

$$\frac{dx_2}{dt} = v_1 - v_{-1} - v_2 + v_{-2} = etc. \quad x_{20} = x_2(t=0)$$

$$\frac{dx_3}{dt} = v_2 - v_{-2} - v_3 = etc. \quad x_{30} = x_3(t=0)$$

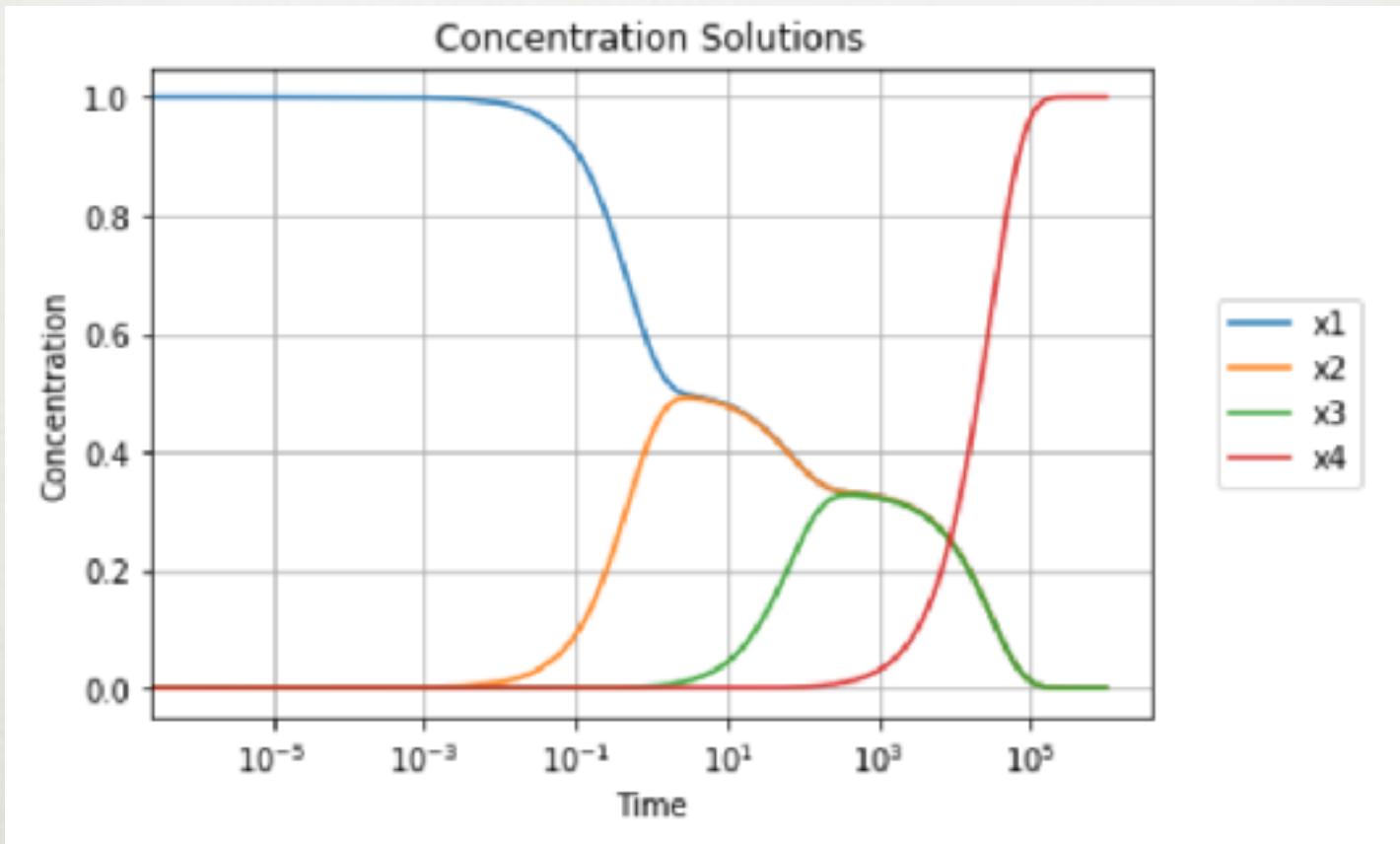
$$\frac{dx_4}{dt} = v_3 = k_3 x_3 \quad x_{40} = x_4(t=0)$$

Example of Time-Scale Decomposition



Example of Time-Scale Decomposition

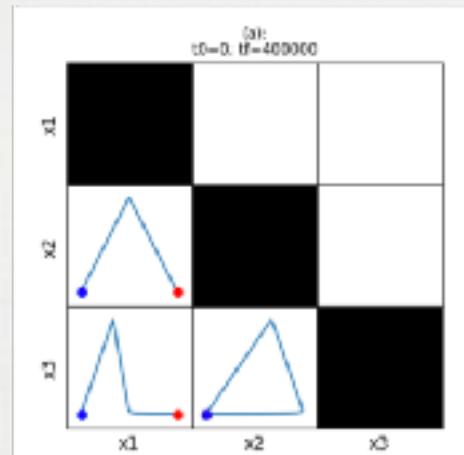
Actual Numerical response



Tiled Phase Portraits: overall and on each time scale

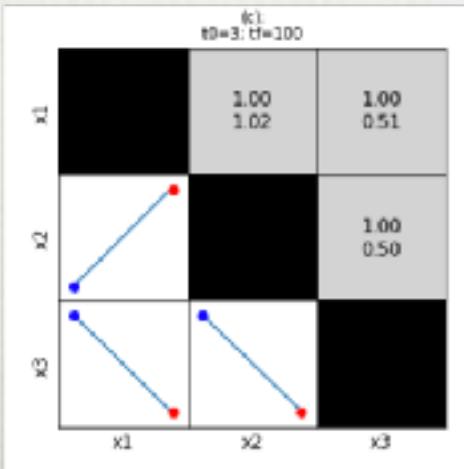
All Times

(a)



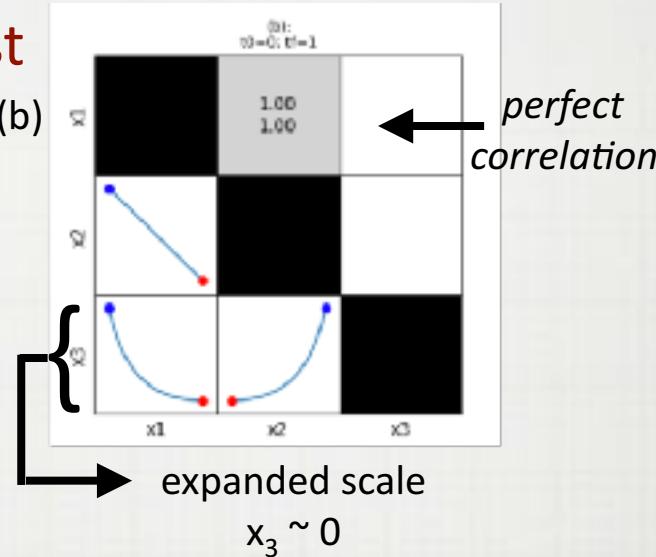
Intermediate

(c)



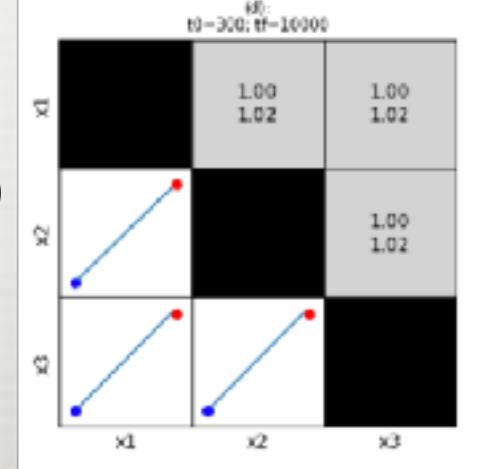
Fast

(b)



Slow

(d)



Post-processing the Solution

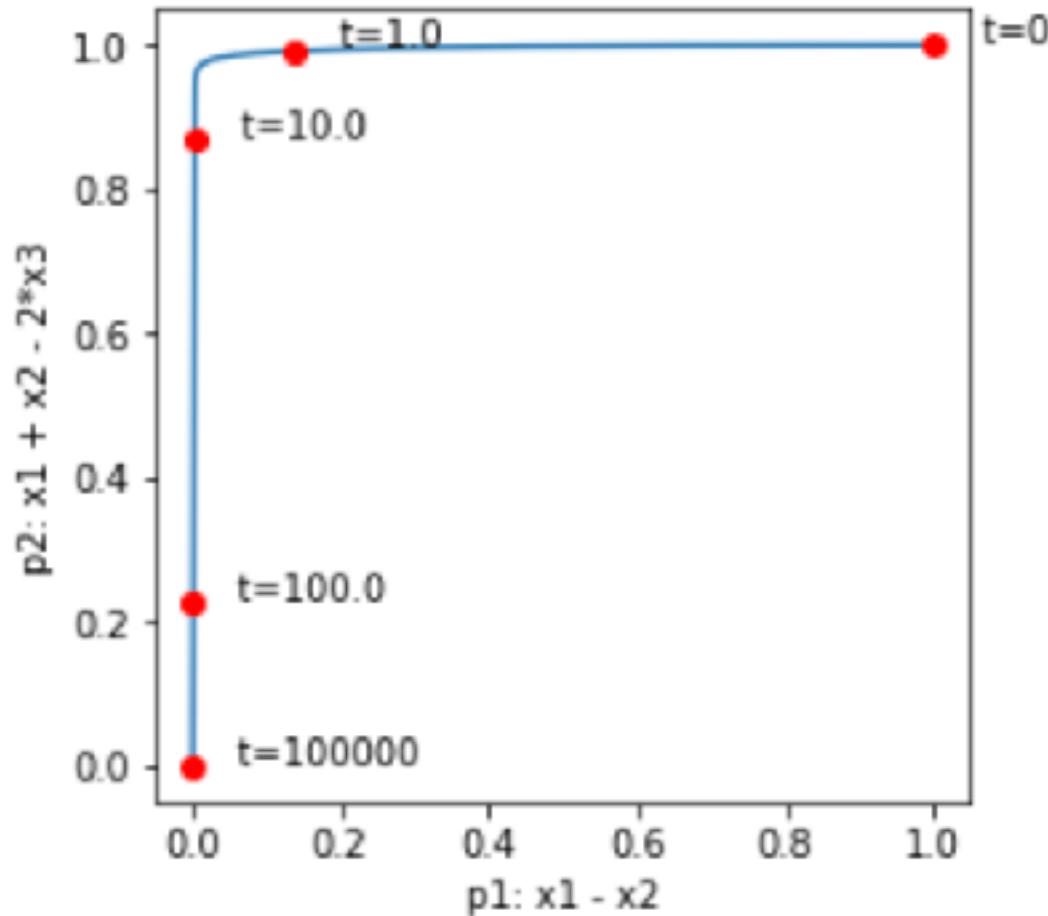
1. Forming pools

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 \\ 1/2 & 1/2 & -1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 - x_2 \\ (x_1 + x_2)/2 - x_3 \\ x_1 + x_2 + x_3 \end{pmatrix}$$

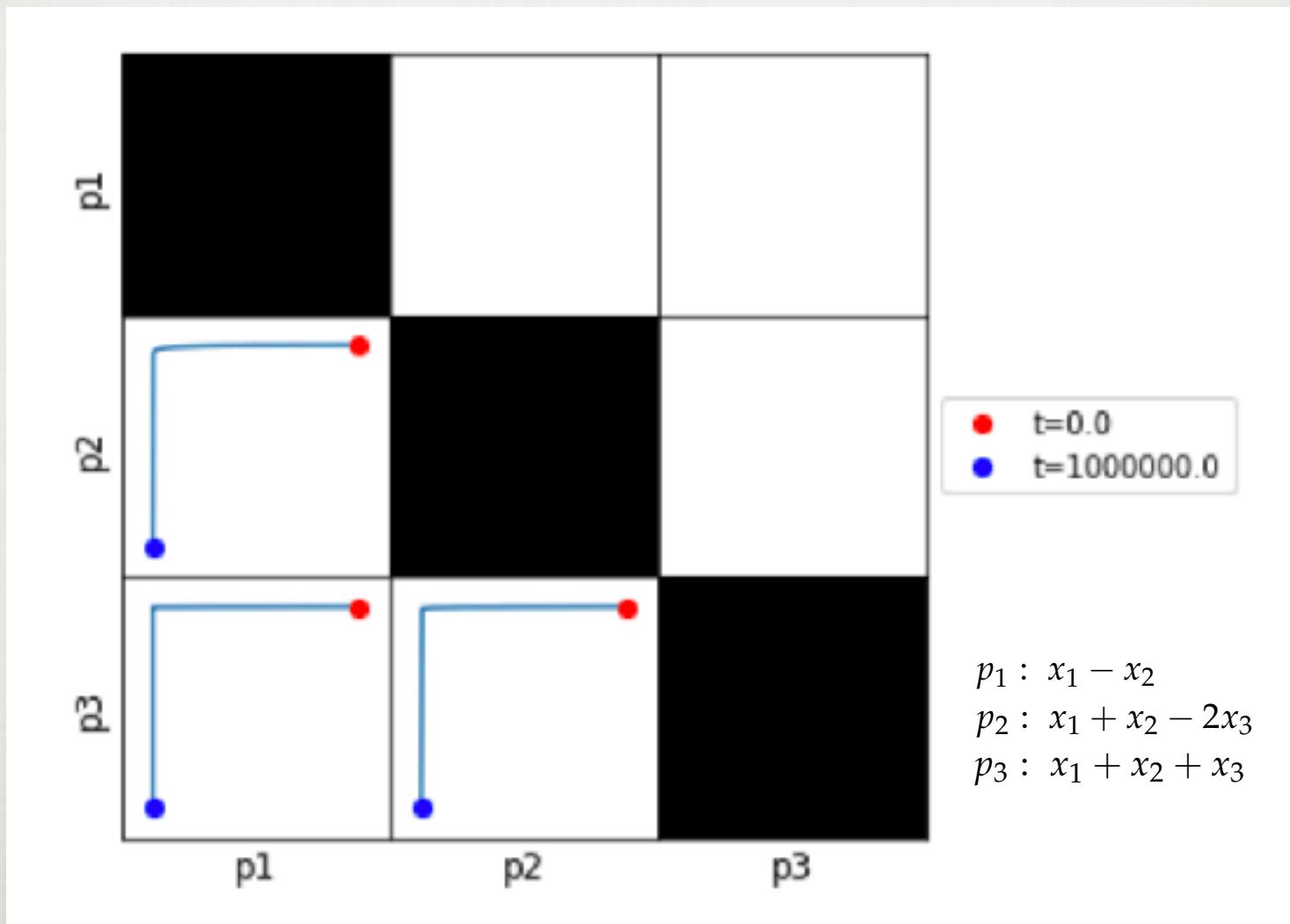
p_1 and p_2 are dis-equilibrium variables

p_3 is a conservation variable

Phase Portrait of Pools: L-shaped



Tiled Phase Portrait of Pools



Summary

- Network dynamics are described by dynamic mass balances $dx/dt = Sv(x; k)$ that are formulated after applying a series of simplifying assumptions
- To simulate the dynamic mass balances we have to specify the numerical values of the kinetic constants(k), the initial conditions(x), and any fixed boundary fluxes.
- The equations with the initial conditions can be integrated numerically using a variety of available software packages. We use MASSpy
- The solution is in a file that contains numerical values for the concentration variables at discrete time points. Often an interpolation function is provided. The solution is graphically displayed, i.e., as concentrations over time, or on a phase portrait.
- The solution can be post-processed following its initial analysis to bring out special dynamic features of the network.
- For the rest of the class, we will apply this simulation procedure to progressively more complicated situations, from the simple to the realistic.

The End