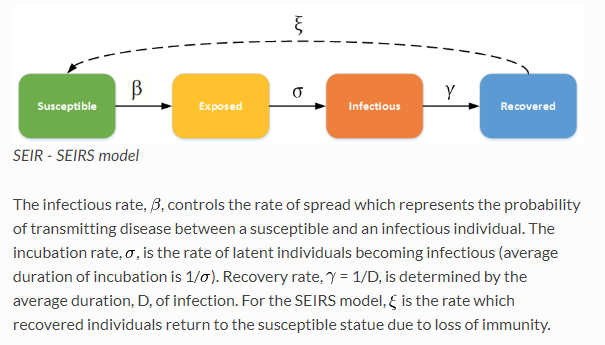
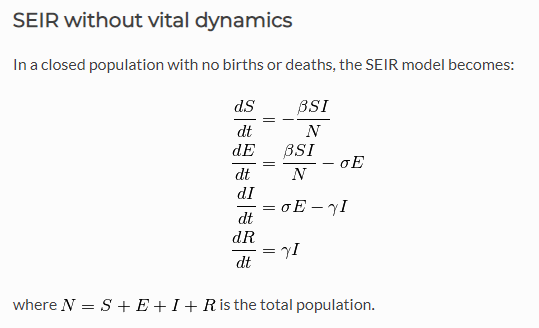
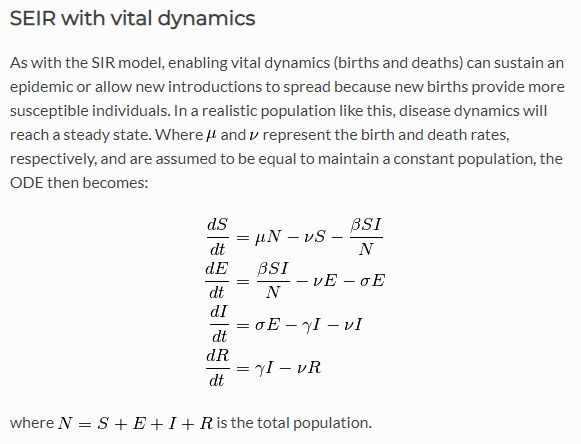
SINDy Lab book

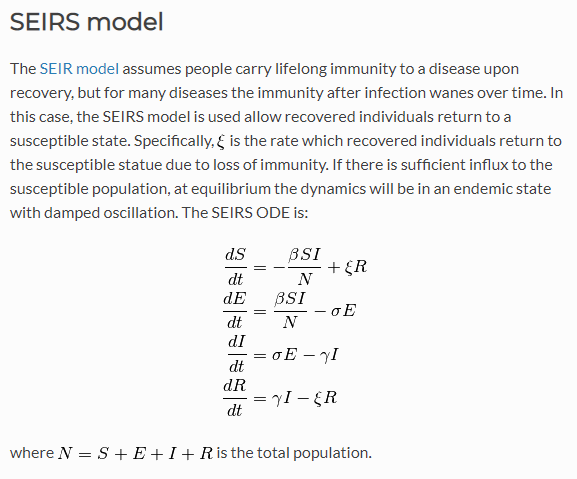
# Overview of SEIR and SEIRS models

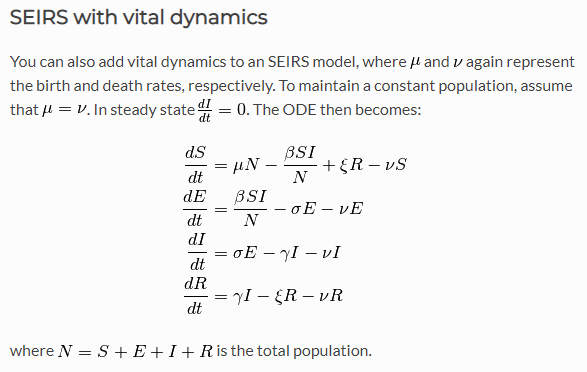
<https://institutefordiseasemodeling.github.io/Documentation/general/model-seir.html>











# Initial SEI SINDy exploration

August 27-29, 2018

Build one!

Build a dynamic model; get output; and then use SINDy to find the model.

Alright!

* Use Niall's SINDy+AIC code as a launch pad (from Niall et al. 2017)

<https://github.com/niallmm/SINDy_AIC/blob/master/EX_SEIR.m>

* Run it out of the box and see if it works.

Drop in my own SIR model.

* Start simple. Run it.
* Mess with it!

## SEI models and population dynamics

From Mangan et al. 2017

**Note: Output table labels**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Xdot | Ydot | Zdot |
| **1** |  |  |  |
| **x** |  |  |  |
| **y** |  |  |  |
| **z** |  |  |  |
| **xx** |  |  |  |
| **xy** |  |  |  |
| **xz** |  |  |  |
| **yy** |  |  |  |
| **yz** |  |  |  |
| **zz** |  |  |  |
| **…** |  |  |  |

**SEI(R), constant population**

EX\_SEIR.m excerpt:

%% generate Data

n = 3; % number of parameters

% all others are zero

% Transfer Parameters

B\_SE = 0.3;

B\_EI = 0.4;

B\_IR = 0.04;

Ntot = 1e4; % total population

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

N = 250; % number of time steps

% disease tranfer model

for ii =2:N

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

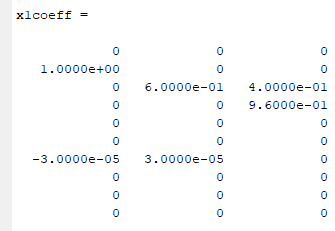
E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

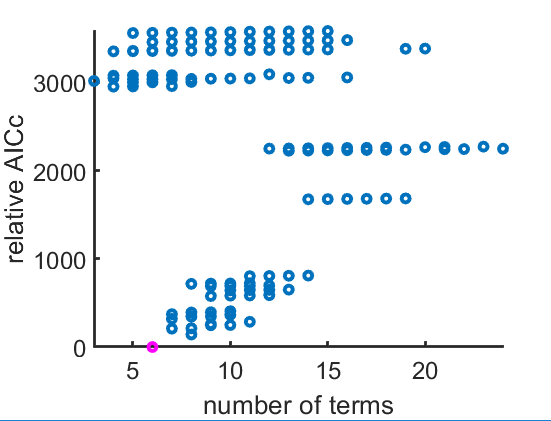
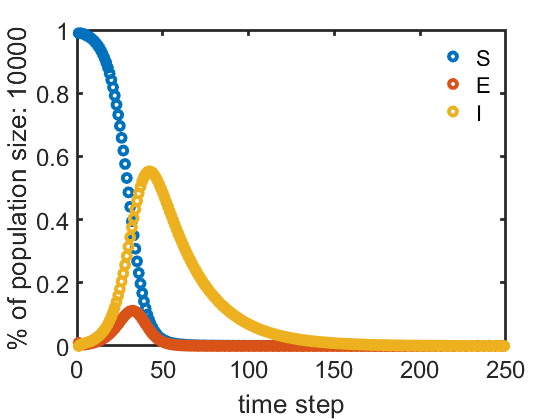
I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

% adding in the R data causes SINDy to fail.

end

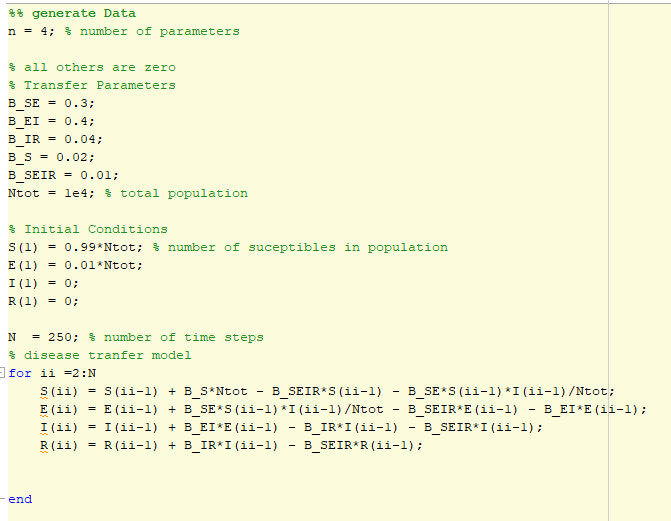
Model successfully identified:

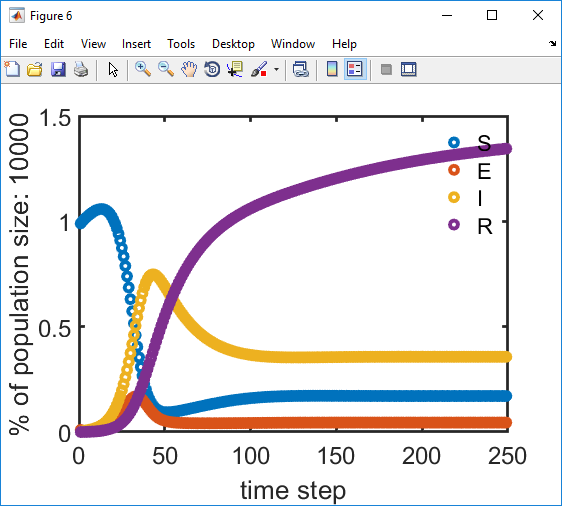
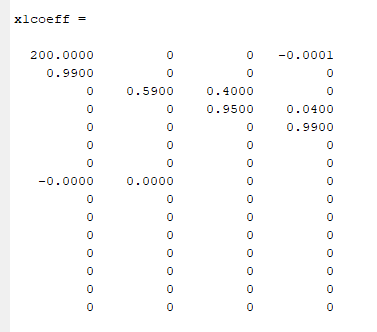


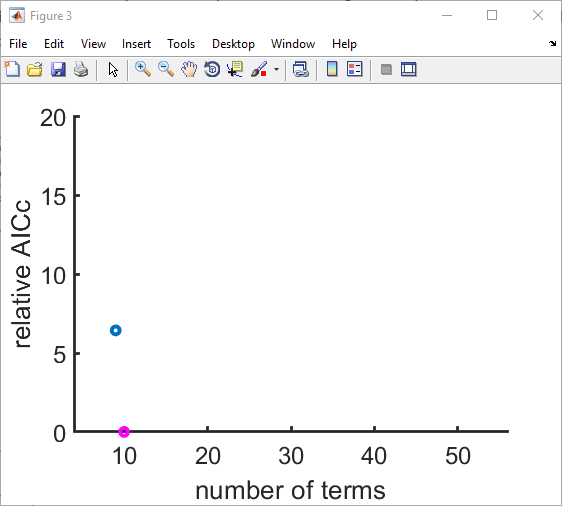
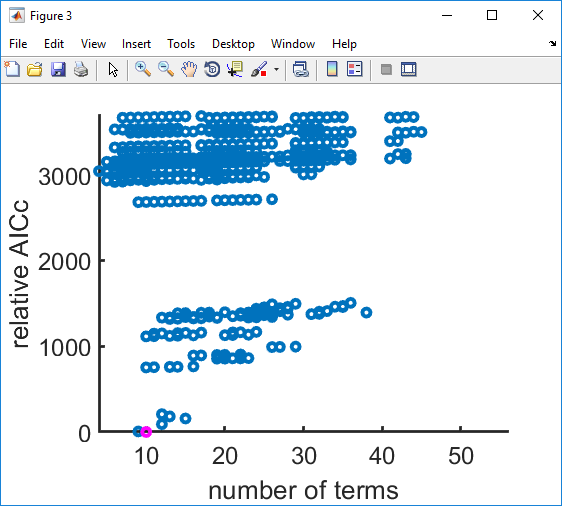


**SEIR with vital dynamics: Birth rate != Death rate; pop. parameter in model constant**

**Mistake! didn't update Ntot for each time step (goes into S and E calculations)**







**Again: birth rate != death rate; this time updating pop. at each time step**

**Mistake! Model requires maintaining constant population, equal birth and death rates**

%% generate Data

n = 4; % Number of equations

% all others are zero

% Transfer parameters

B\_SE = 0.3; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate

% Vital parameters

B\_S = 0.017; % Birth rate

B\_SEIR = 0.014; % Death rate

Ntot = 1e4; % Total (initial) population

N = 250; % number of time steps

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

R(1) = 0;

plotTitle = 'SEIRvital';

% disease tranfer model

for ii =2:N

%{

% SEIR model, static pop.

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

% adding in the R data causes SINDy to fail.

%}

% SEIR model, vital dynamics

Ntot = S(ii-1) + E(ii-1) + I(ii-1) + R(ii-1); % Update pop.

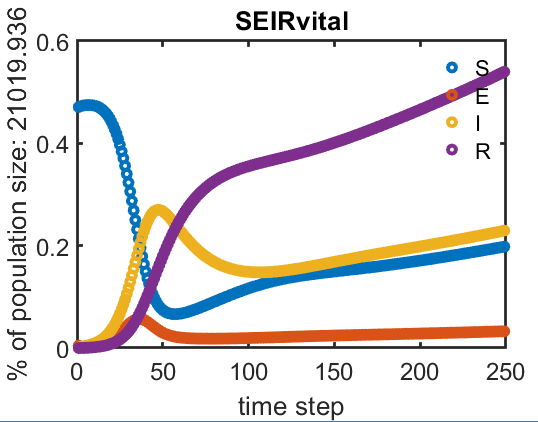
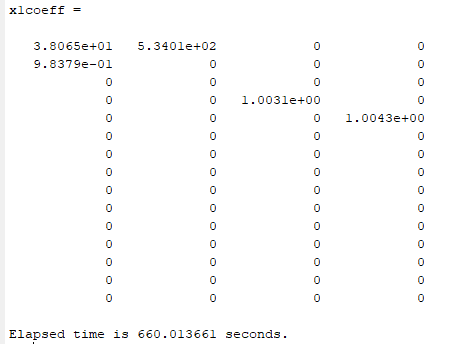
S(ii) = S(ii-1) + B\_S\*Ntot - B\_SEIR\*S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

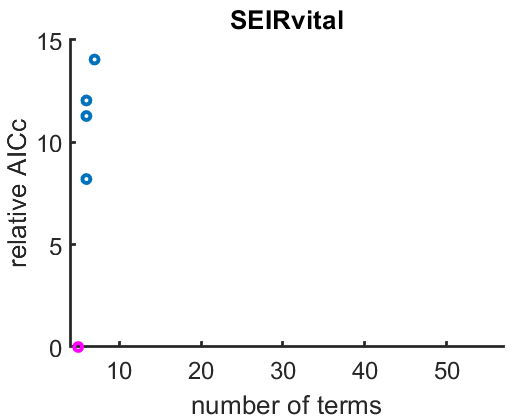
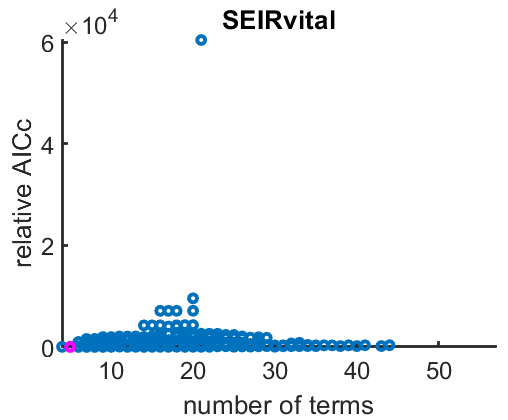
E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1) - B\_SEIR\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1) - B\_SEIR\*I(ii-1);

R(ii) = R(ii-1) + B\_IR\*I(ii-1) - B\_SEIR\*R(ii-1);

end





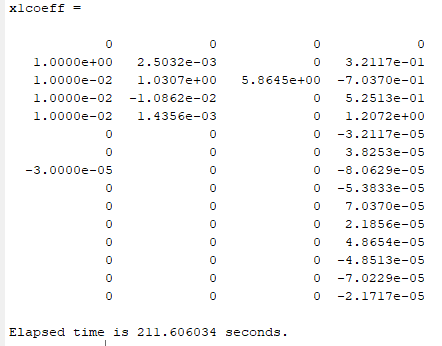
**Again:**

**This time vital dynamics with equal birth and death rates.**

Take out updating Ntot, constant pop.

"Warning: Rank deficient"

Model not identified:



R redundant again because pop is constant…

**Again, this time take R out:**

**SEIR with vital dynamics, Birthrate == Deathrate, Constant pop., No R**

%% generate Data

n = 3; % Number of equations

% all others are zero

% Transfer parameters

B\_SE = 0.3; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate

% Vital parameters

B\_S = 0.02; % Birth rate

B\_SEIR = 0.02; % Death rate

Ntot = 1e4; % Total (initial) population

N = 250; % number of time steps

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

%R(1) = 0;

plotTitle = 'SEIRvital';

% disease tranfer model

for ii =2:N

%{

% SEIR model, static pop.

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

% adding in the R data causes SINDy to fail.

%}

% SEIR model, vital dynamics

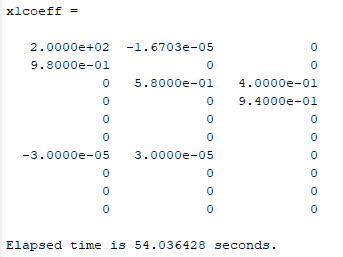
S(ii) = S(ii-1) + B\_S\*Ntot - B\_SEIR\*S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

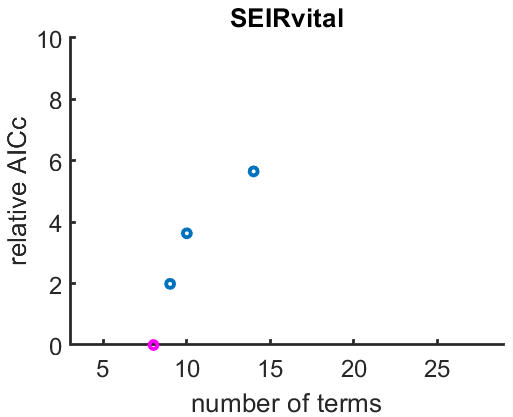
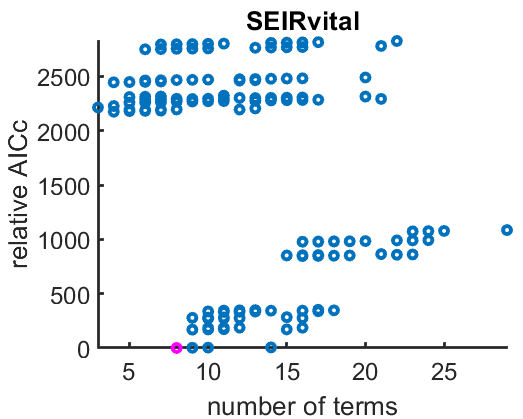
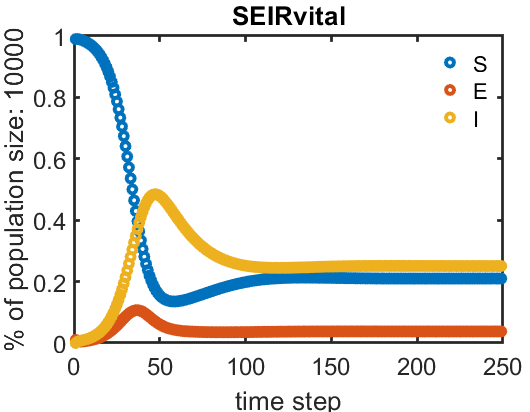
E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1) - B\_SEIR\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1) - B\_SEIR\*I(ii-1);

% R(ii) = R(ii-1) + B\_IR\*I(ii-1) - B\_SEIR\*R(ii-1);

end





**Ta dah!**

## Toy modeling

### Dependent variables

% Toy modeling

S(1) = 1;

E(1) = 1;

I(1) = 1;

%

b1 = 0.4;

b2 = 0.3;

b3 = 0.2;

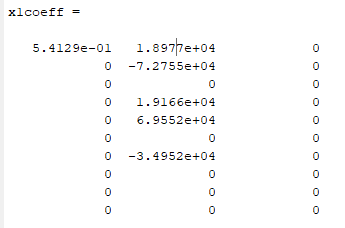
for ii =2:N

S(ii) = S(ii-1) - b1\*S(ii-1)\*I(ii-1);

E(ii) = E(ii-1) - b1\*S(ii-1) - b2\*I(ii-1)\*E(ii-1);

I(ii) = b3\*I(ii-1);

Model not successfully identified.



**Again:**

% Toy modeling

S(1) = 1;

E(1) = 1;

I(1) = 1;

%

b1 = 0.4;

b2 = 0.3;

b3 = 0.2;

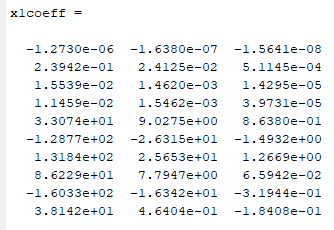
for ii =2:N

S(ii) = b1\*S(ii-1);

E(ii) = b2\*E(ii-1);

I(ii) = b3\*I(ii-1);

Model not successfully identified:



**Again:**

%% Toy modeling

S(1) = 1;

E(1) = 1;

I(1) = 1;

%

b1 = 1.2;

b2 = 0.9;

b3 = 0.8;

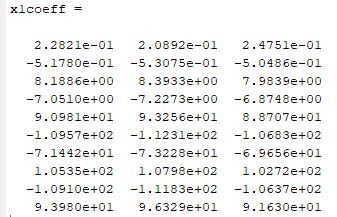
for ii =2:N

S(ii) = S(ii-1) + 0.04;

E(ii) = E(ii-1) + 0.041;

I(ii) = I(ii-1) + 0.039;

Model not successfully identified:



### Lambda

Is this because the lambda range doesn't cover "sparse enough"?

min(numcoeff) = 18

Before:

lambdavals.numlambda = 20;

lambdavals.lambdastart = -10;

lambdavals.lambdaend = 1;

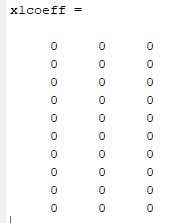
Changed lambda range:

lambdavals.numlambda = 20;

lambdavals.lambdastart = -1;

lambdavals.lambdaend = 10;

Model still not successfully identified, but I get different results!

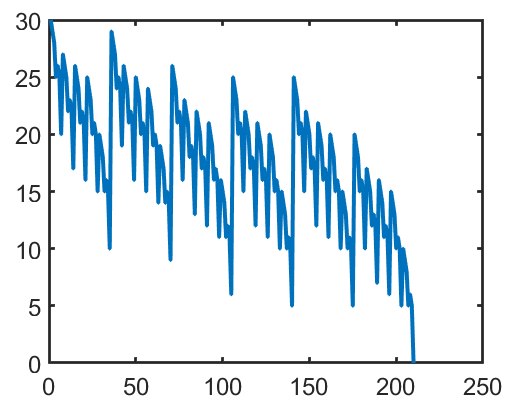


Note!

Could it be that the AIC is picking out the wrong model?

numcoeff plot:

Lambda = (-1 : 20 : 10)



Last value is 0, second to last value is 5.

Need more resolution? Higher lambdas?

**Try:**

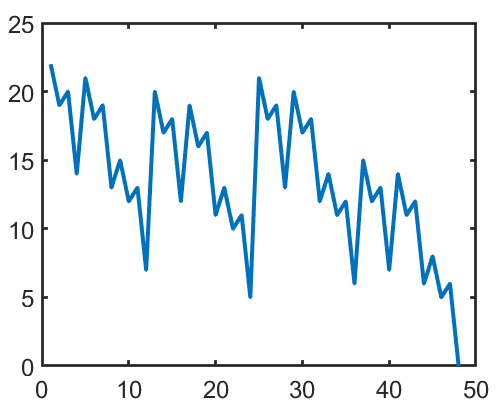
lambdavals.numlambda = 20;

lambdavals.lambdastart = 0;

lambdavals.lambdaend = 10;

Numcoeff plot: (48 models)

Lambda = (0 : 20 : 10)



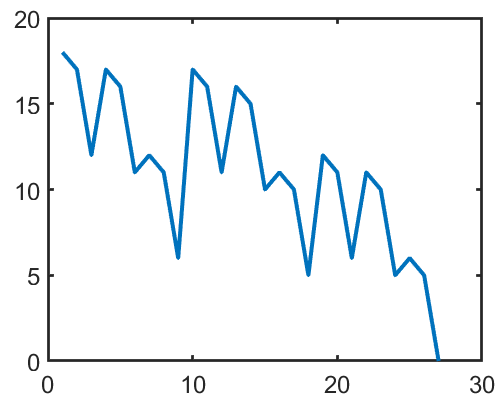
**Try:**

lambdavals.numlambda = 20;

lambdavals.lambdastart = 1;

lambdavals.lambdaend = 10;

Lambda = (1 : 20 : 10)



Turning up lambda start to 2, keeping numlambda and lambdaend the same breaks it: one model is returned with zero parameters.

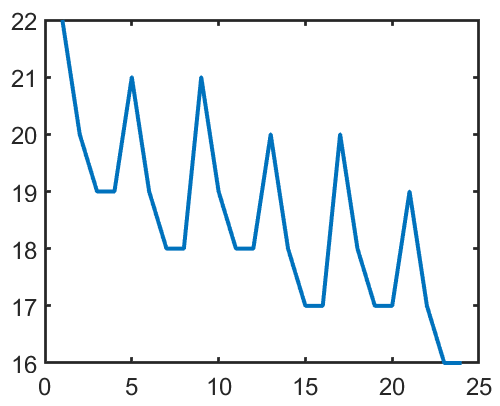
Turning down lambda end to 4, keeping numlambda and lambda end produces the same plot as above.

**Try:**

lambdavals.numlambda = 20;

lambdavals.lambdastart = 0;

lambdavals.lambdaend = 1;

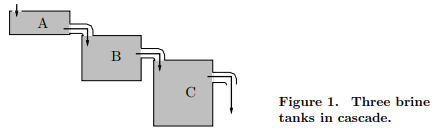


I don't think the problem is lambda.

### Rank: Simple brine tank example

**Nope, the problem is the specified equations:**

**Try these instead:**

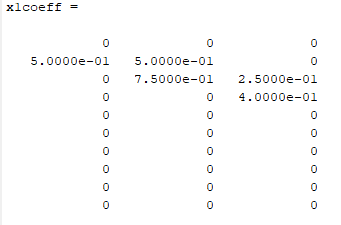


% Toy Model

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.6\*I(ii-1);



That works.

Okay, now let's break it. This is the issue of linear independence (rank).

% Toy Model: Test rank

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.6\*E(ii-1);

Yes. This throws the Rank insufficient warning and returns a model that is not correct.

Warning: rank insufficient comes from sparsifyDynamics.m

Line 20: Xi = Theta\dxdt

In these cases, SINDy returns an answer, but it is wrong.

That means that Input data to SINDy must be tested for rank and dependent variables removed.

# Timescale translations

## Check valid B\_SE range (infectious rate)

RESULTS: B\_SE between 0.3 and 0.5 is okay.

Why do values outside of that range break it? How do I know? How can I make it robust to that? Normalization/standardization already happens in the code?

Model and population:

Ntot = 1e4; % Total (initial) population

% Disease transfer model

for ii =2:N

% SEIR model, static pop.

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

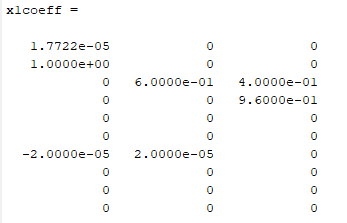
% adding in the R data causes SINDy to fail.

Test: B\_SE = 0.2   
% Transfer parameters

B\_SE = 0.2; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate



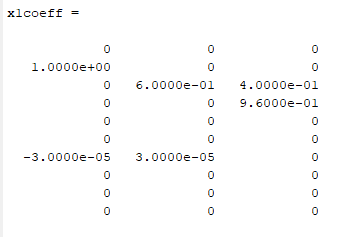
Test: B\_SE = 0.3 (default)

% Transfer parameters

B\_SE = 0.3; % Infectious rate % DEFAULT

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate



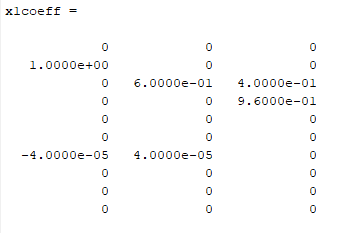
Test: B\_SE = 0.4

% Transfer parameters

B\_SE = 0.4; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate



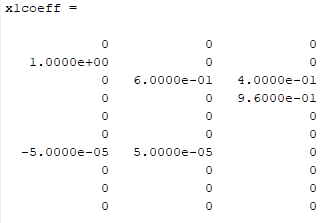
Test: B\_SE = 0.5

% Transfer parameters

B\_SE = 0.5; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate



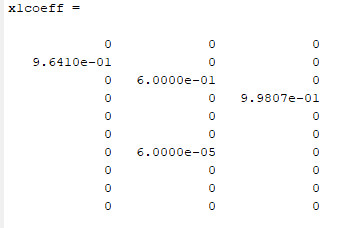
Test: B\_SE = 0.6

% Transfer parameters

B\_SE = 0.6; % Infectious rate

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate



## Making synthetic data

### Toy data experimentation

t = 0:500;

p = 2 + 0.01\*t + cos(0:.1:(.1\*500)); % Oscillator + line

d = 4.\*(p(2:end)-p(1:end-1)); % Piece-wise slope

s = 2+0.6.\*(d.\*p(1:end-1)); % Slope .\* p

figure

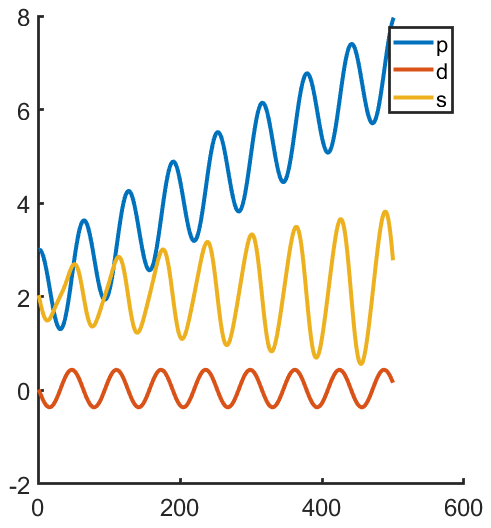
hold on

plot(p)

plot(d)

plot(s)

legend('p','d','s')



### PRECIPTIATION: Setup/Functions/Implications

Okay, let's make Infectious rate dependent on past precipitation.

i.e., infectious rate depends the history of precipitation at a higher frequency. Infectious rate at each low frequency timestep depends on high frequency history of precipitation. We are looking for the *function* to aggregate the *higher* frequency data for use in a model specification at the lower frequency.

Health model timestep (infectious rate updated at each time step)



Precipitation timestep

Possible ways to aggregate precipitation history:

* Sum
  + Functional form of sum = x1 + x2 +…+ xi
  + where each xi is a separate feature in the aggregation model and xi is precipitation at the ith timestep
* Mean
  + Functional form of mean = x1+x2 +…+ xi
  + As above, where each xi is precipitation at the ith timestep and separate feature in the aggregation model
* Min: [How/do I write functional forms for min and max?]
* Max

Notes!

* The length of the history of precipitation relevant to infectious rate at timestep *i* could be longer than the timestep of the health model.
* i.e. infectious rate update at day 0, 6, 12; precipitation days 5-12 relevant to infectious rate at day 12.
* You are finding the timescale of the impact of precipitation on infectious rate. Some timestep features in your aggregation model with have non-zero coefficients; some will have a zero coefficient.
* Further! Because you are not just finding a binary 1-0 feature selection, but a value of the coefficients, you are actually finding the memory kernel of the effects of precipitation on infectious rate.

Implementation:

This is now a non-autonomous model, i.e. precipitation is an exogenous factor.

So we need to make that data and hand it to the model. Let's use a random walk for precipitation.

Epidemic model specification now changes. Replace the infectious rate parameter (formerly a constant) with the aggregation functional form.

CAUTION:

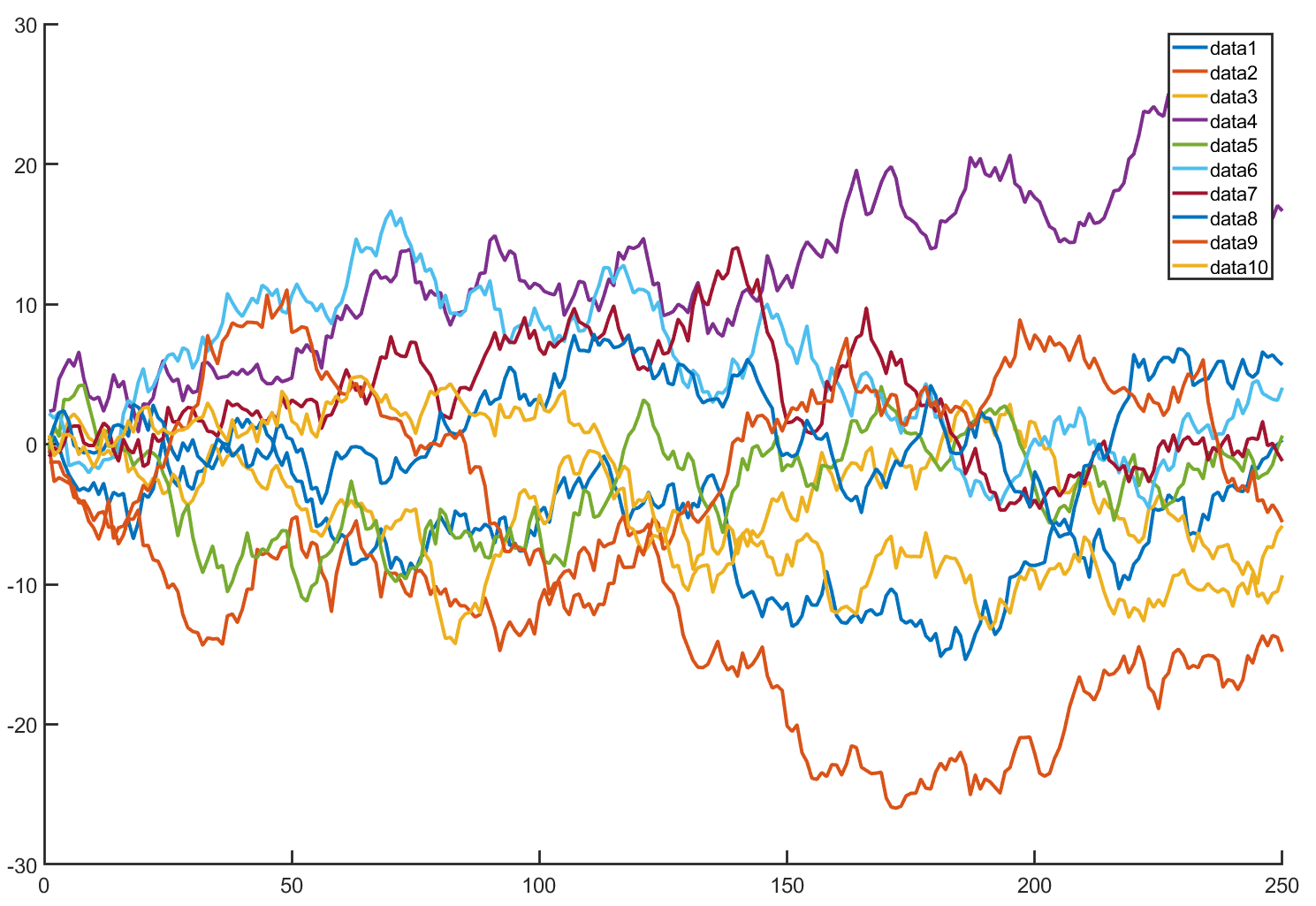
* Need to check what range of infectious rate is "safe" for the model so I can make sure I'm playing within that range. (see section above)
* Need to make a time variable on which to hang Epi model and precip time steps instead of indices… (?)

#### Pick a pretty random walk for precipitation

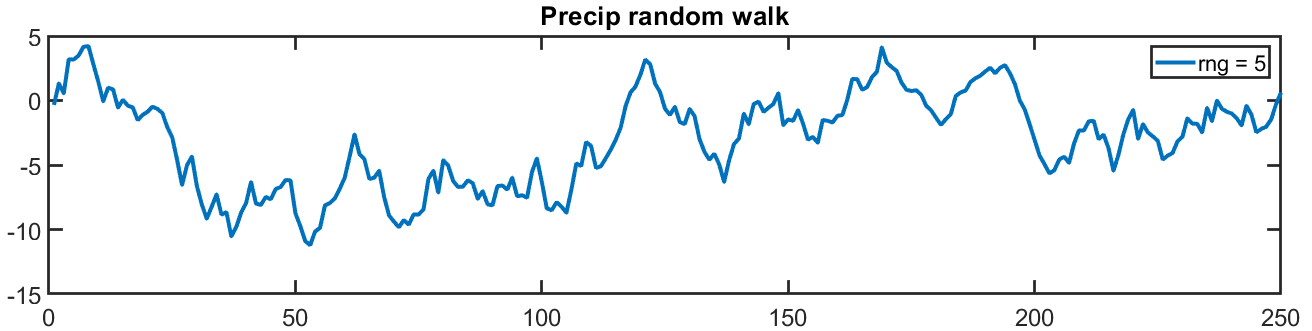
rng(seed#)

plot(cumsum(randn(1,250)))

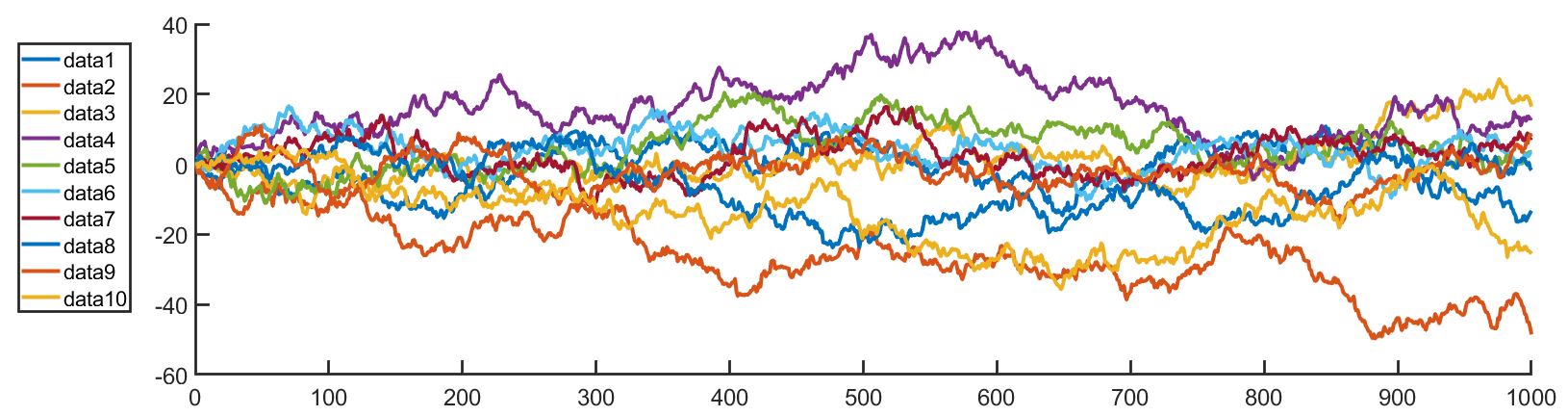
Plot below is seeds 1:10, time series number corresponds to seed

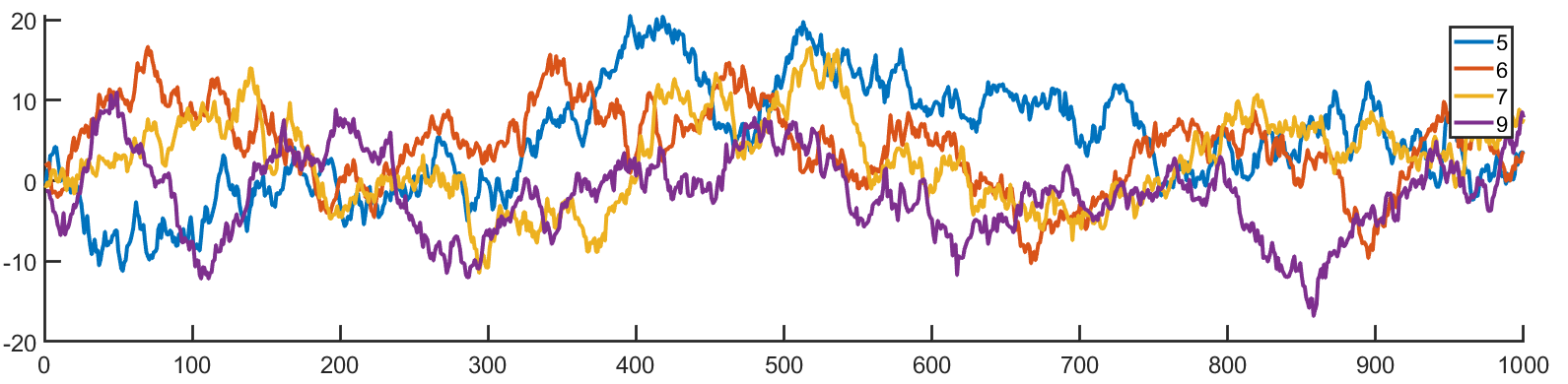


Seed = 5 looks good.



Except precip is going to be at a higher frequency (4x the frequency of the health model) so we need 1000 time steps instead of 250.





So with 1000 time steps, let's use rng(6) instead

#### Set up P for use with B\_SE

**In code:**

% Random walk precipitation

rng(6) % Set seed

N=250; % From SEI code

p = cumsum(randn(1,N\*4));

figure; plot(p)

title('Precip random walk')

legend('rng = 6')

% Center and standardize to +/- 1

range = max(p) - min(p);

pCenter = p-max(p)+0.5\*range; plot(pCenter)

pStand = pCenter./max(pCenter); plot(pStand)

p = pStand; % Reassign p to pStand

title('P (standardized to +/- 1)')

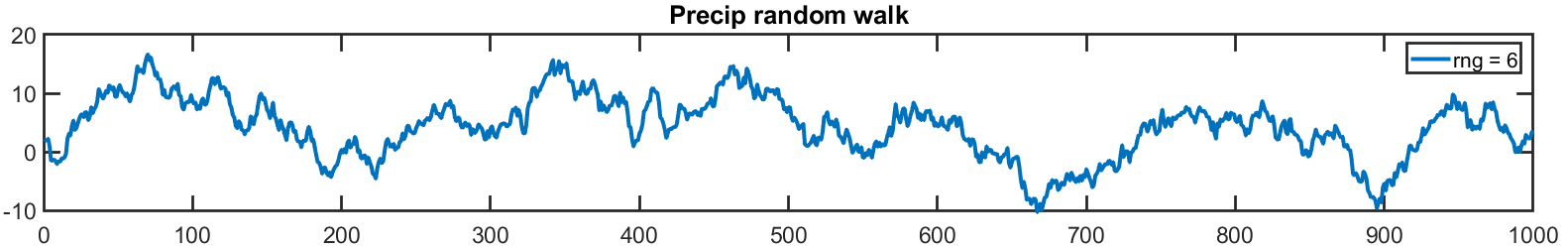
% Calculate derivative time series (p = pStand)

dp = [0,(p(2:end)-p(1:end-1))]; % Piece-wise slope

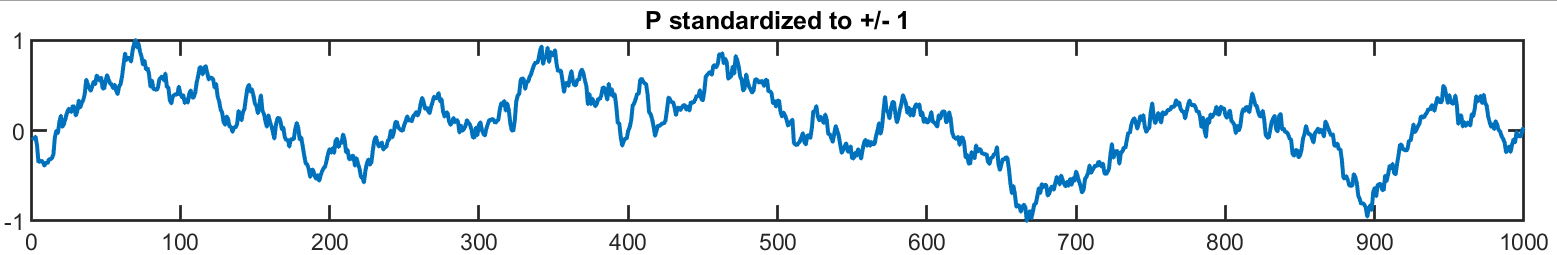
figure; plot(dp)

title('d(pStand)')

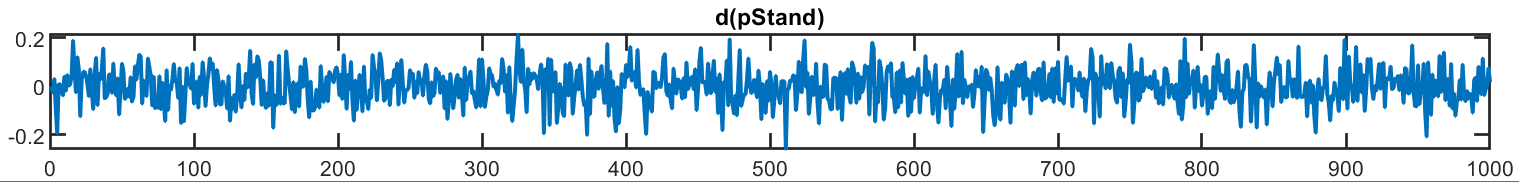
Make the random walk time series



Center and standardize (no variance adjustment)



Make and plot derivative time series (derivative of the centered and standardized P) – for now we aren't using this



## B\_SE = fn(sum(P))

Bnew = B\_SE + pSumRanged;

B\_SE = 0.4

valid B range = 0.3 – 0.5

range(pSumRanged) = +/- 0.1

### Make the sum(P); N=250 time series; Standardize to valid range

pSum steps through N=1000 by interval = 4

At each of those 250 timesteps, sum the current and previous 3 precipitation measurements (4 measurements total)

Now you have an N=250 timeseries that has rolled up the N=1000 precipitation timeseries by each sum of 4 previous measurements.

Standardize this pSum dataset to +/- 0.1 so we stay within the valid range of B\_SE when we add pSum to B\_SE=0.2

% Make 4 timestep sum rolled P

pStack = zeros(4,N\*4); % Initialize

pStack(:,4:end) = [p(4:end);

p(3:end-1);

p(2:end-2);

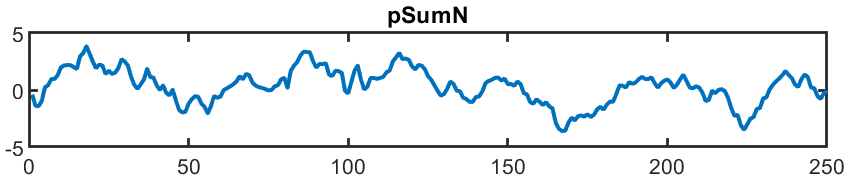
p(1:end-3)];

% Pick out only the indices corresponding to health model timesteps

pickInd = find(rem((1:1000),4)==0);

pSumN = sum(pStack(:,pickInd),1);

figure; plot(pSumN); title('pSumN')



**Standardize:**

% Standardize pSumN to the range taken by B\_SE = +/- 0.1

pSumRanged = 0.1.\*(pSumN/max(pSumN)); % plot(pSumRanged)

% Use the pSumRanged as input for our model

P = pSumRanged;

### Change Equations; Don't add P eqn

We are changing B\_SE to (B\_SE + P), where P = pSumRanged

(B/N)\*SI 🡪 ((B+P)/N))\*SI = (B/N)\*SI + (P/N)\*SI

= (B/N)\*SI + (1/N)\*SIP, where P is an exogenous timeseries variable

% Transfer parameters

B\_SE = 0.4; % Infectious rate: between 0.3 and 0.5 doesn't break

B\_EI = 0.4; % Incubation rate

B\_IR = 0.04; % Recovery rate

…

% SEIR model, static pop., B\_SE = fn(P)

S(ii) = S(ii-1) - (B\_SE\*S(ii-1)\*I(ii-1)/Ntot + S(ii-1)\*I(ii-1)\*P(ii-1)/Ntot);

E(ii) = E(ii-1) + (B\_SE\*S(ii-1)\*I(ii-1)/Ntot + S(ii-1)\*I(ii-1)\*P(ii-1)/Ntot)...

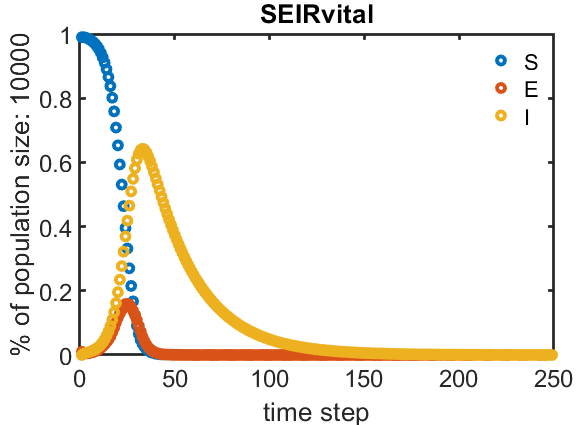
- B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

Just making those changes to the system of equations, this is the total system timeseries I get.

(The code gives me synthetic timeseries output.)

For B\_SE = fn(sum(P)):



Compare with the system timeseries from the original equations:

% SEIR model, static pop.

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

For different values of B\_SE:

|  |  |  |
| --- | --- | --- |
| B\_SE = 0.3 | B\_SE = 0.4 | B\_SE = 0.5 |
|  |  |  |

The range of B\_SE produces a continuum of system timeseries behavior.

**The B\_SE = fn(sun(P)) looks similar, but not exactly the same as the system for B\_SE = 0.4.**

(I ran B\_SE = 0.4 again and the dots line up exactly the same way every time, which is slightly different from the way they line up in B\_SE=fn(sum(P)).)

**We would expect that, because our pSumRanged timeseries is distributed around 0.**

(We could verify and also play with that pSumRanged distribution shape.)

### Model identification

Three equations set up, four included in SINDy step (?)

How did I set this up? Go back to code.

% SEIR model, static pop., B\_SE = fn(sum(P)),

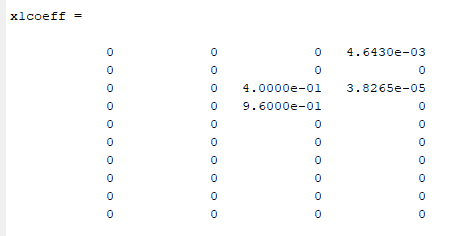
% where sum is taken from last four timesteps

S(ii) = S(ii-1) - (B\_SE\*S(ii-1)\*I(ii-1)/Ntot + S(ii-1)\*I(ii-1)\*P(ii-1)/Ntot);

E(ii) = E(ii-1) + (B\_SE\*S(ii-1)\*I(ii-1)/Ntot + S(ii-1)\*I(ii-1)\*P(ii-1)/Ntot)...

- B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);



Code runs, but model not identified.

### Time lags

% SEIR model, static pop., B\_SE = fn(P\_tlags)

S(ii) = S(ii-1) - (B\_SE+...

pt1(ii-1)+pt2(ii-1)+pt3(ii-1)+pt4(ii-1)+...

pt5(ii-1)+pt6(ii-1)+pt7(ii-1)+pt8(ii-1))\*...

S(ii-1)\*I(ii-1)/Ntot;

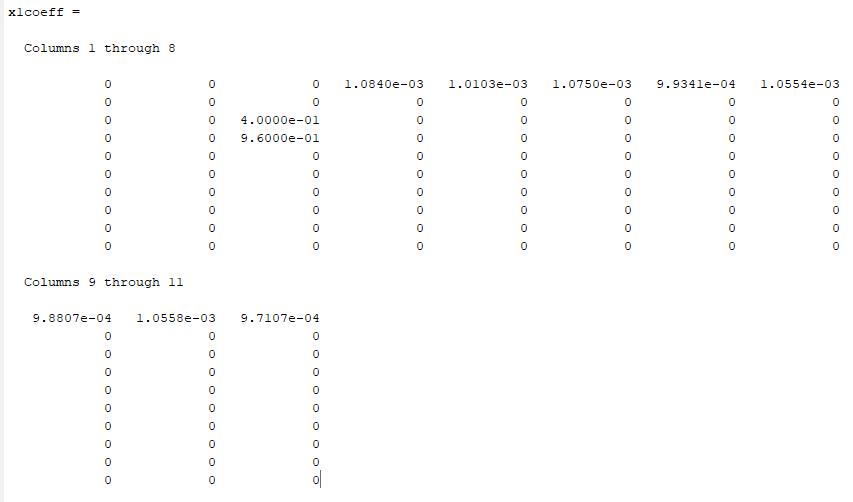
E(ii) = E(ii-1) + (B\_SE+...

pt1(ii-1)+pt2(ii-1)+pt3(ii-1)+pt4(ii-1)+...

pt5(ii-1)+pt6(ii-1)+pt7(ii-1)+pt8(ii-1))\*...

S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);



That took too long. Use fewer time lagged variables.

% SEIR model, static pop., B\_SE = fn(P\_tlags)

S(ii) = S(ii-1) - (B\_SE+...

0.01\*pt1(ii-1)+pt2(ii-1)+0.05\*pt3(ii-1)+0.08\*pt4(ii-1))\*...

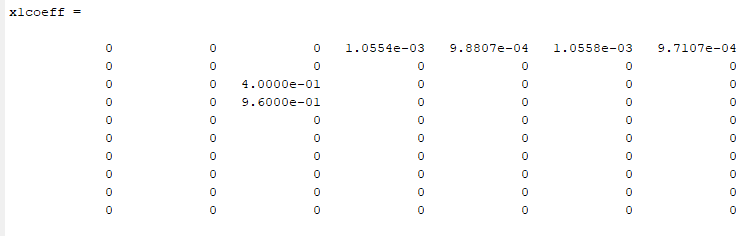
S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + (B\_SE+...

0.4\*pt1(ii-1)+0.015\*pt2(ii-1)+pt3(ii-1)+0.07\*pt4(ii-1))\*...

S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);



Model not identified

## Back to basics

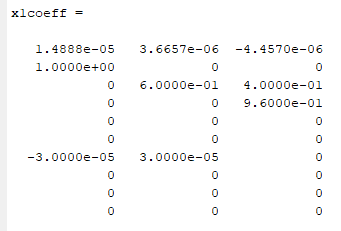
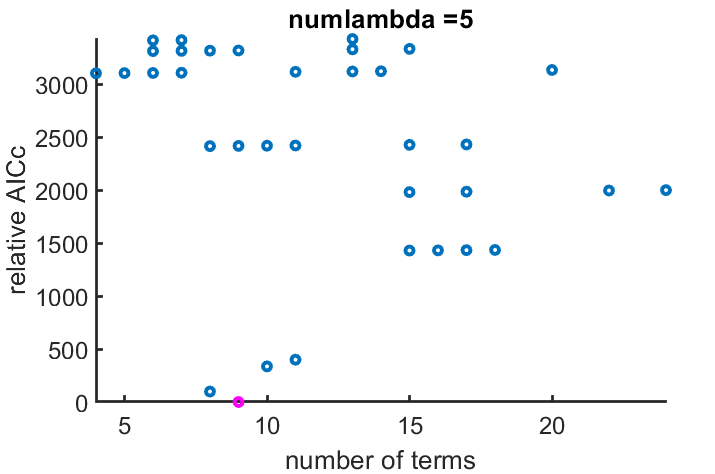
### Lambda library resolution affects model specification

Result: In Niall's AIC code, default numLambdas is 20. Turns out I do actually need at least this number to get a good model specification. In this case it's the *resolution* of the lambda library, not the range that impacts the ability to recover the model specification.

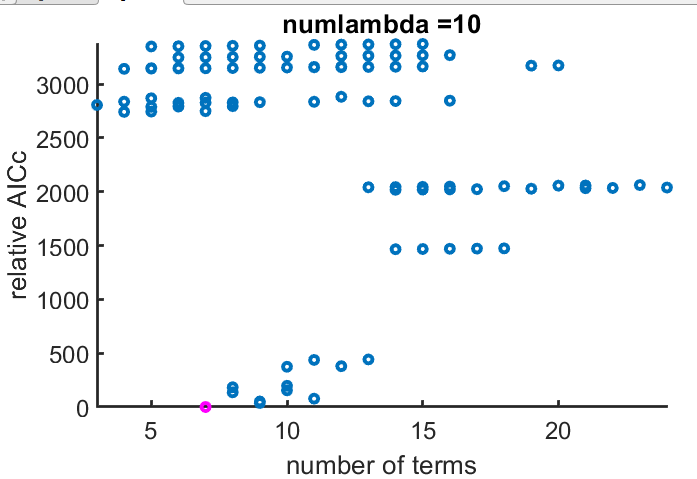
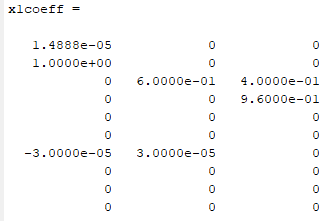
Proposal: Determine your resolution is fine enough by looking for model specification convergence with increasing resolution of lambda.

#### Testing:

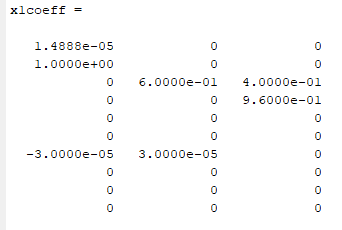
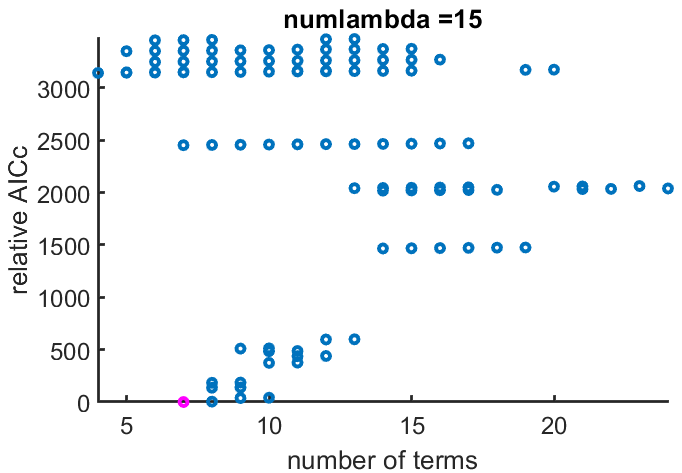
If I change numlambda to 5, I don't identify the right model:



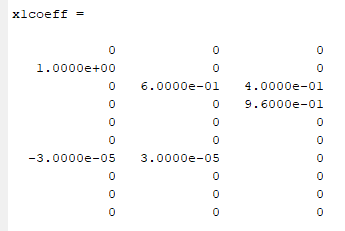
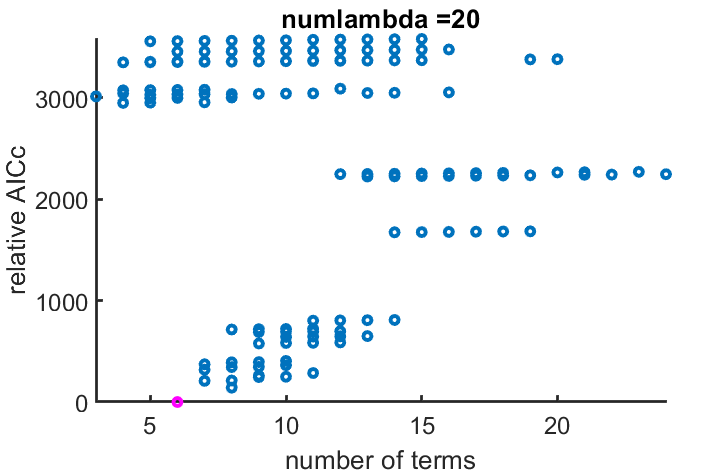
Nor numlambda = 10 (but it's better):

Numlabda = 15:



Numlambda = 20 (default) is successful:



### Mangan's dx isn't actually a derivative?

% create x and dx matrices with all variables:

x = [S(1:end-1)' E(1:end-1)' I(1:end-1)'];

dx = [S(2:end)' E(2:end)' I(2:end)'];

And this doesn't work:

% create x and dx matrices with all variables:

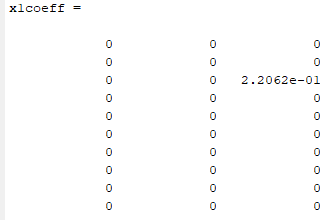
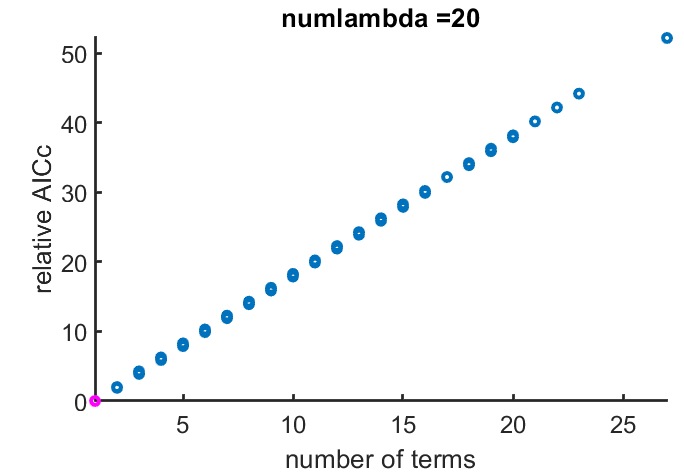
x = [S(1:end-1)' E(1:end-1)' I(1:end-1)'];

%dx = [S(2:end)' E(2:end)' I(2:end)'];

dx = [(S(2:end)-S(1:end-1))',...

(E(2:end)-E(1:end-1))',...

(I(2:end)-I(1:end-1))'];



### Adding in P

Starting again from Niall's EX\_SEIR.m doing incremental modifications.

#### Add a P equation. Simple, doesn't interact with anything.

**Equations:**

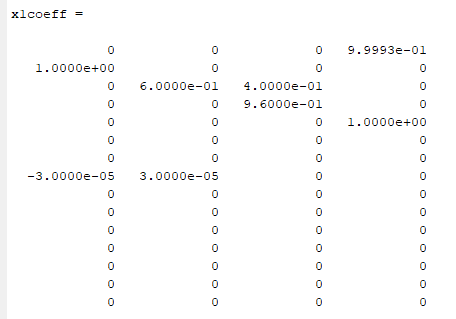
S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

P(ii) = P(ii-1) + 1;

**Results:**



**Good enough.**

#### Now make it an exogenous time-dependent variable:

**Model:**

% Transfer Parameters

B\_SE = 0.3;

B\_EI = 0.4;

B\_IR = 0.04;

Ntot = 1e4; % total population

N = 250; % number of time steps

p = cumsum(randn(1,N));

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

P(1) = p(1);

% disease tranfer model

for ii =2:N

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

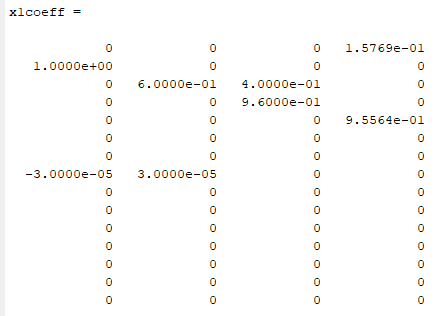
E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

P(ii) = p(ii);

end

**Results:**



i.e. correct model specification ( plus SINDy does its best to parameterize P)

!!! This takes a long time.

#### Add P interaction with B\_SE:

Note: anytime you change the eqns, check the time series plot to make sure you didn't break anything!

%% generate Data

n = 4; % number of equations

% all others are zero

% Transfer Parameters

B\_SE = 0.4;

B\_EI = 0.4;

B\_IR = 0.04;

Ntot = 1e4; % total population

N = 250; % number of time steps

rng(6) % Set seed for consistent results

p = cumsum(randn(1,N));

range = max(p) - min(p);

pCenter = p - max(p) + 0.5\*range;

pStand = 0.1 .\* (pCenter./max(pCenter)); % standardized to +/- 0.1

p = pStand;

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

P(1) = p(1);

% disease transfer model

for ii =2:N

S(ii) = S(ii-1) - (P(ii-1)+B\_SE)\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + (P(ii-1)+B\_SE)\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

P(ii) = p(ii);

% Don't forget to update the cross-validation!

end

% create x and dx matrices with all variables:

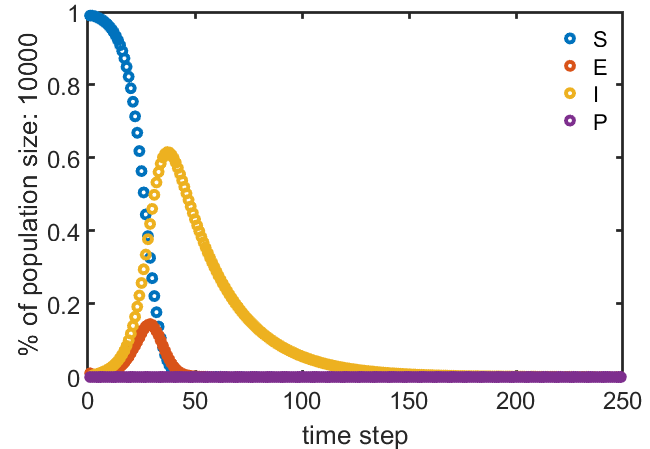
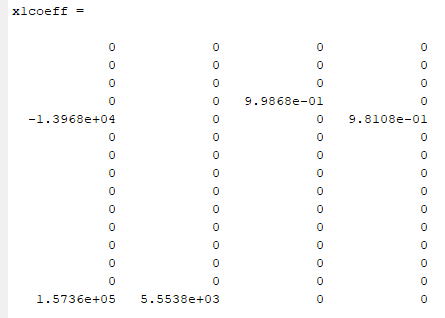
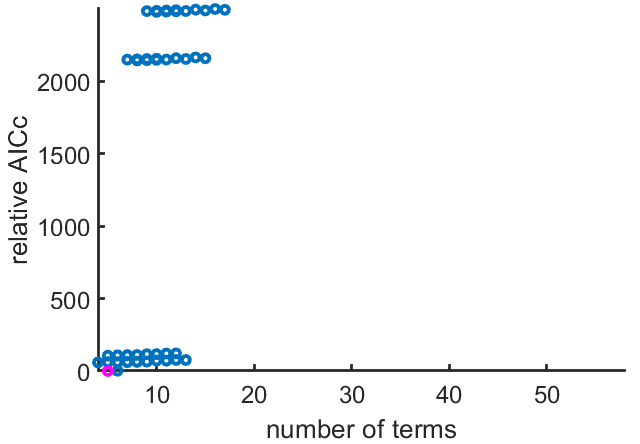
x = [S(1:end-1)' E(1:end-1)' I(1:end-1)' P(1:end-1)'];

dx = [S(2:end)' E(2:end)' I(2:end)' P(2:end)'];

% add noise to state variables

rng(10);

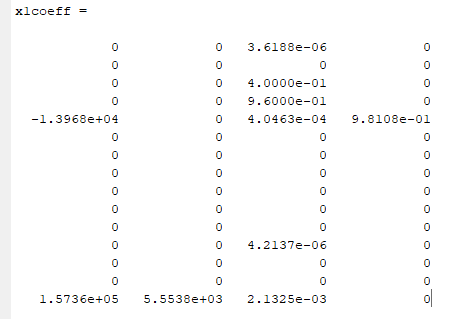
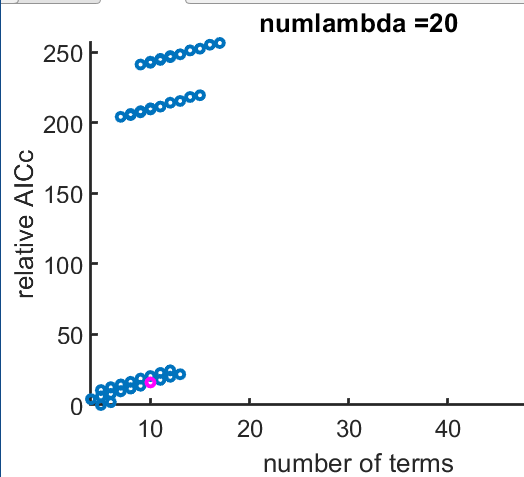
x = x+eps\*randn(size(x));

Not great. Also, it takes too long.

Reduce number of cross-validations to make it a little faster:

numvalidation = 10; % number of crossvalidation experiments

That's super weird. The pink one should be the lowest relative AIC…

But it is faster.

Running it again gives me the same results.

#### Return to non-interacting P

%% generate Data

n = 4; % number of equations

% all others are zero

% Transfer Parameters

B\_SE = 0.4;

B\_EI = 0.4;

B\_IR = 0.04;

Ntot = 1e4; % total population

N = 250; % number of time steps

rng(6) % Set seed for consistent results

p = cumsum(randn(1,N));

range = max(p) - min(p);

pCenter = p - max(p) + 0.5\*range;

pStand = 0.1 .\* (pCenter./max(pCenter)); % standardized to +/- 0.1

p = pStand;

% Initial Conditions

S(1) = 0.99\*Ntot; % number of suceptibles in population

E(1) = 0.01\*Ntot;

I(1) = 0;

P(1) = p(1);

% disease transfer model

for ii =2:N

S(ii) = S(ii-1) - B\_SE\*S(ii-1)\*I(ii-1)/Ntot;

E(ii) = E(ii-1) + B\_SE\*S(ii-1)\*I(ii-1)/Ntot - B\_EI\*E(ii-1);

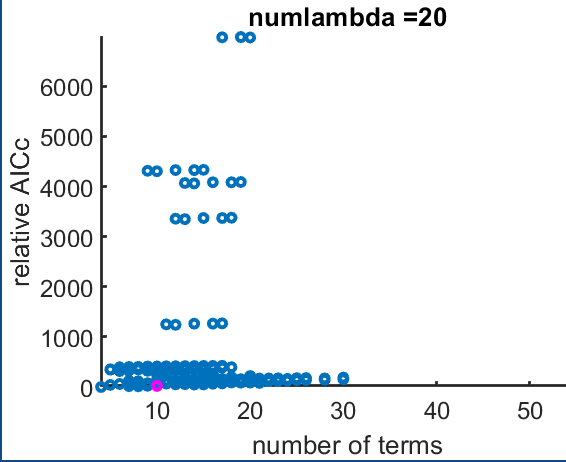
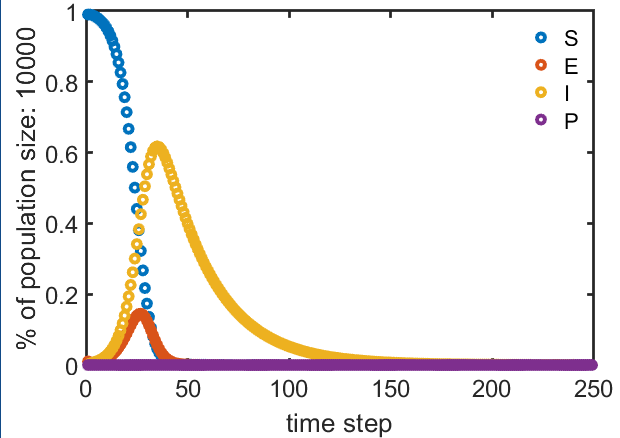
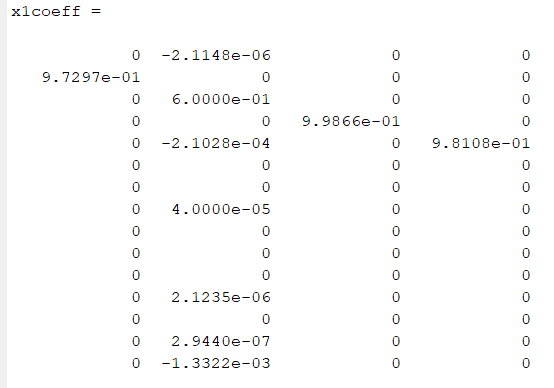
I(ii) = I(ii-1) + B\_EI\*E(ii-1) - B\_IR\*I(ii-1);

P(ii) = p(ii);

% Don't forget to update the cross-validation!

end

No good:

Um… The model specification is the same as the first non-interactive round.

Why would it be different? The only difference I can discern is the centering/standardization of P… and I don't see how that would make a difference….

But let's take it out and see.

In synthetic data creation section:

rng(6) % Set seed for consistent results

p = cumsum(randn(1,N));

% range = max(p) - min(p);

% pCenter = p - max(p) + 0.5\*range;

% pStand = 0.1 .\* (pCenter./max(pCenter)); % standardized to +/- 0.1

% p = pStand;

In cross-validation section:

%% calculate validation data for new intial conditions.

x0cross = 10.^(-1 + (4+1)\*rand(n,numvalidation));

p = cumsum(randn(N,numvalidation),2);

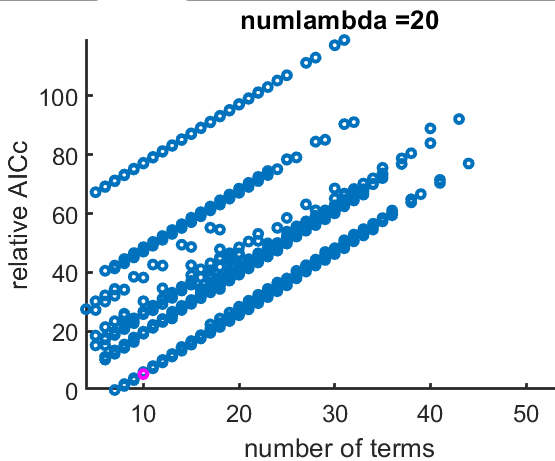
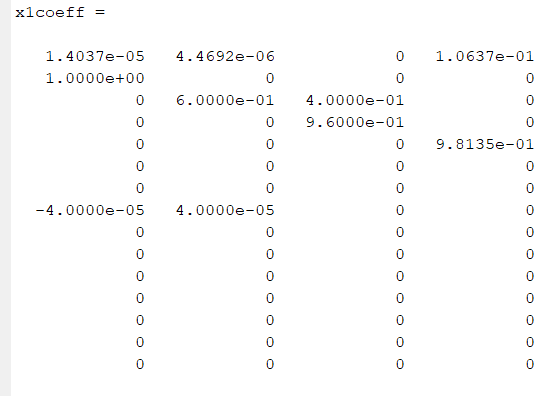
% range = max(p) - min(p);

% pCenter = p - max(p) + 0.5\*range;

% pStand = 0.1 .\* (pCenter./max(pCenter)); % standardized to +/- 0.1

% p = pStand;

Oh, and I also changed the number of cross-validations, but keep the new value (10) for now.

Better. Extraneous constant terms on the S and E specifications, but otherwise it's right.

(???)

Why does the standardization of P negatively affect the model identification success?

#### Best model selection vs. number of terms

And the magenta model is not the min(relativeAIC) model…

That doesn't seem right…

Spoiler alert: It's not.

Recall:

%% EX\_SEIR\_MVG.m

AIC\_rel =cell2mat({IC.aic})-min(cell2mat({IC.aic}));

%% AnalyzeOutput.m

minind = find(min(cell2mat({IC.aic\_c})) == cell2mat({IC.aic\_c}), 1, 'first')

…

plot(numcoeff, AIC\_rel, 'o')

…

plot(numcoeff(minind),AIC\_rel(minind),'om') % Highlight best model

SO: AIC\_c is used to pick the model, but relative AIC is used to plot.

If I add the "best" model picked by relative AIC:

find(AIC\_rel==min(AIC\_rel))

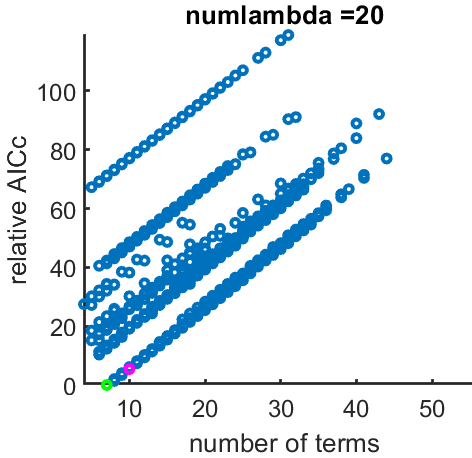
ans =

3422

hold on

plot(numcoeff(3422),AIC\_rel(3422),'og')

I do indeed get the model selection I expect from the plot:



y-axis label on the plot should be: "relative AIC"

AND: these two different models are not the same!

%% AnalyzeOutput.m

minind = find(min(cell2mat({IC.aic\_c})) == cell2mat({IC.aic\_c}), 1, 'first')

mincoeff = numcoeff(minind)

x1coeff = Xicomb{minind}

% TESTING

minind2 = find(min(AIC\_rel) == AIC\_rel, 1, 'first')

mincoeff2 = numcoeff(minind2)

x1coeff2 = Xicomb{minind2}

Output:

minind =

2589

mincoeff =

10

x1coeff =

1.4037e-05 4.4692e-06 0 1.0637e-01

1.0000e+00 0 0 0

0 6.0000e-01 4.0000e-01 0

0 0 9.6000e-01 0

0 0 0 9.8135e-01

0 0 0 0

0 0 0 0

-4.0000e-05 4.0000e-05 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

minind2 =

3422

mincoeff2 =

7

x1coeff2 =

0 0 0 1.0637e-01

9.7297e-01 0 0 0

0 6.0000e-01 4.0000e-01 0

0 0 9.6000e-01 0

0 0 0 9.8135e-01

0 0 0 0

0 0 0 0

0 4.0000e-05 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

In this case, the best model identification by the AIC metric is the same as by the BIC metric.

Note:

IC =

1×3528 struct array with fields:

aic

bic

aic\_c

In summary:

AIC\_rel is made from IC.aic

The model selection is made from IC.aic\_c

The plot is made with IC.aic

**And the "best" model is different between them!**

Now a closer look:

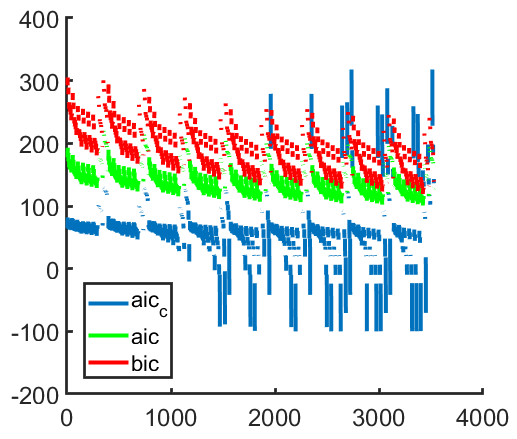
figure; hold on;

plot(cell2mat({IC.aic\_c}))

plot(cell2mat({IC.aic}),'g')

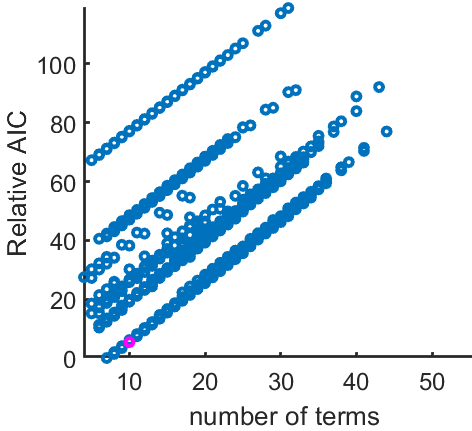
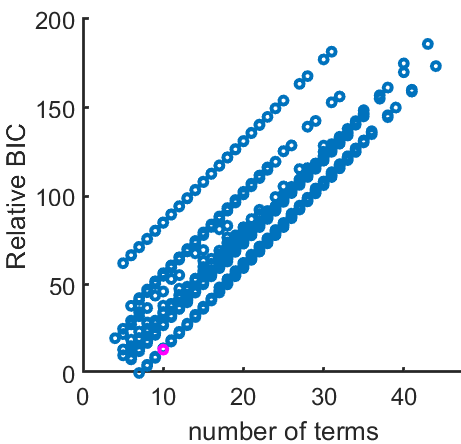
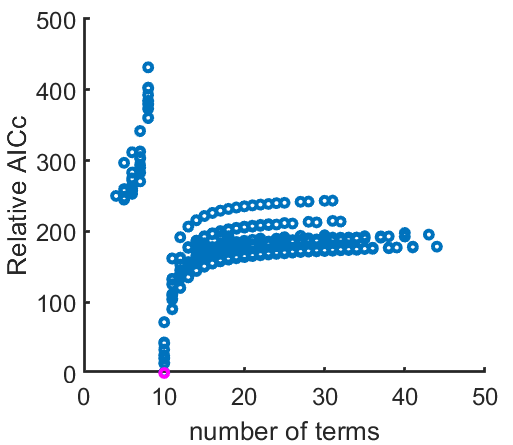
plot(cell2mat({IC.bic}),'r')

legend('aic\_c','aic','bic')



Updated IC vs. # of Terms:

Best model by AIC\_c criteria in magenta.

# Back to toy model: Brine tank cascade

## Start with three equations

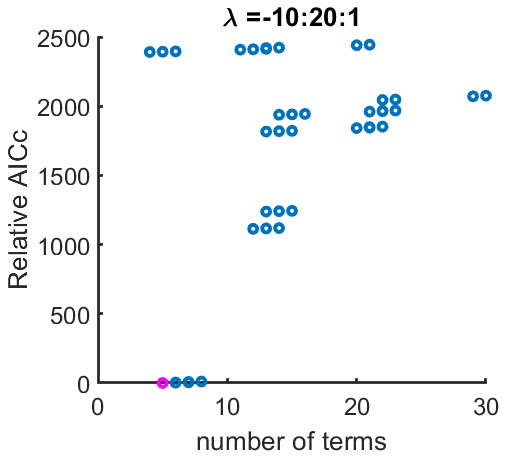
% Toy Model: Brine tank cascade

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.6\*I(ii-1);

Results:



x1coeff =

0 0 0

0.5000 0.5000 0

0 0.7500 0.2500

0 0 0.4000

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

>> lambdavec(minind,:)

ans =

8 10 10

## Lambdas for identified model are invariant

If the lambda values for the identified model do not change with multiple runs, I can potentially nix the lambda search routine.

Run the model identification again, change nothing.

I do still get the same lambda values:

>> lambdavec(minind,:)

ans =

8 10 10

# Adding a fourth variable (code development)

## Endogenous, Non-interacting

Start with adding a fouth endogenous variable that does not interact with the other state equations.

% Toy Model: Brine tank cascade

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1);

P(ii) = P(ii-1) + 1/N;

Change theta feature library to 1 degree for tractability.

Good:

x1coeff =

0 0 0 0.0039

0.5000 0.5000 0 0

0 0.7500 0.2500 0

0 0 0.3000 0

0 0 0 1.0001

ans =

9 11 10 5

>> 1/N

ans =

0.0040

Change P range for simplicity:

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.6\*I(ii-1);

P(ii) = P(ii-1) + N;

Still works:

minind =

223

x1coeff =

0 0 0 249.9999

0.5000 0.5000 0 0

0 0.7500 0.2500 0

0 0 0.3000 0

0 0 0 1.0000

ans =

9 10 10 5

## Endogenous, Interacting

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1) + 0.1\*P(ii-1);

P(ii) = P(ii-1) + 0.4;

Looks good:

x1coeff =

0 0 0 0.3999

0.5000 0.5000 0 0

0 0.7500 0.2500 0

0 0 0.3000 0

0 0 0.1000 1.0000

ans =

9 10 10 5

# Exogenous variables and time lags

## Scalability for exogenous variables (code development)

Change code such that an exogenous variable is included in Theta creation, but not in the model identification. Do this in a way that is easily changed.

It seems to work:

x1coeff =

0 0 0

0.5000 0.5000 0

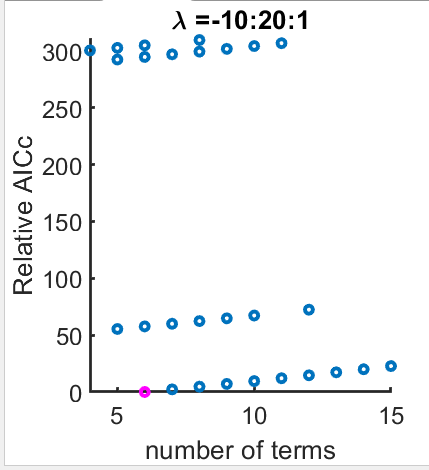
0 0.7500 0.2500

0 0 0.3000

0 0 0.1000

ans =

9 10 10



## Interacting, with time lags

% Toy Model: Brine tank cascade

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1) + ...

0.1\*P(ii-1) + 0.05\*P1(ii-1) + 0.025\*P2(ii-1);

Inexact model identification:

x1coeff =

0 0 0

0.5000 0.5000 0

0 0.7500 0.2500

0 0 0.3000

0 0 0.0752

0 0 0.0998

0 0 0

ans =

14 16 16

Choose a different model specification:

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1) + ...

0.5\*P(ii-1) + 1.0\*P1(ii-1) + 4.0\*P2(ii-1);

Again, not quite:

x1coeff =

0 0 -0.1332

0.5000 0.5000 0

0 0.7500 0.2500

0 0 0.3000

0 0 1.7806

0 0 1.7805

0 0 1.9390

ans =

14 16 5

Interesting: the endogenous variable identification is exact while the exogenous feature identification does not do so well.

It's possible that the code structure is not doing what it should be doing, but I don't find an error.

Back to the previous specification of lagged P:

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1) + ...

0.1\*P(ii-1) + 0.05\*P1(ii-1) + 0.025\*P2(ii-1);

x1coeff =

0 0 0

0.5000 0.5000 0

0 0.7500 0.2500

0 0 0.3000

0 0 0.0752

0 0 0.0998

0 0 0

ans =

14 16 16

And yes, that P2 on the lower right is actually zero:

>> format short e

>> x1coeff

x1coeff =

0 0 0

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 7.5216e-02

0 0 9.9784e-02

0 0 0

## Interacting, no timelags

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1)...

+ 0.1\*P(ii-1);

% + 0.05\*P1(ii-1) + 0.025\*P2(ii-1);

x1coeff =

0 0 0

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.0000e-01

ans =

9 10 10

That works as before.

So, interacting exogenous variable with no time lags works, but adding timelagged features breaks it.

Maybe there is something wrong with the way I make the timelagged time series?

The exogenous P variable is correctly identified when no timelags of P are included in the feature set, but is NOT correctly identified when timelagged P features are included.

# Add a second exogenous variable, no timelags

## Don't change the model specification

Just add another exogenous variable to the feature search space:

%% generate Data

N = 250; % number of time steps

nLags = 0; % number of lags

nExoVars = 2; % number of exogenous variables (without lags)

% Specify how many variables to use

nExo = nExoVars + nLags; % total number of exogenous variables included in poolData

nFunc = 3; % number of equations to keep for model ID

% Exogenous variable

P = linspace(0,10,N+nLags)';

exo = P;

% More than one exogenous variable

Q = linspace(5,15,N+nLags)';

exo = [exo Q];

Looks good:

x1coeff =

0 0 0

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.0000e-01

0 0 0

lambdas =

15 16 16

## Add 2nd exogenous variable to model specification

Nope.

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1)...

+ 0.1\*P(ii-1) + 0.2\*Q(ii-1);

Is identified as:

x1coeff =

0 0 3.0388e-01

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.6078e-01

0 0 1.3922e-01

lambdas =

15 16 5

Go back to the model structure with one exogenous variable (P) that worked…

### Changing the *coefficient* of P (from 0.1 to 0.2) breaks it (no second exogenous variable).

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1)...

+ 0.2\*P(ii-1);

Results in a model identification of:

x1coeff =

0 0 -4.4698e-01

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.1060e-01

0 0 8.9398e-02

lambdas =

15 16 4

So the problem is not necessarily the presence of the variable, but with the value of the coefficient.

### Making Q coefficient same as P doesn't work:

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1)...

+ 0.1\*P(ii-1) + 0.1\*Q(ii-1);

x1coeff =

0 0 5.3017e-02

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.1060e-01

0 0 8.9398e-02

lambdas =

15 16 4

### Making Q data itself identical to P doesn't work either.

% Exogenous variable

P = linspace(0,10,N+nLags)';

exo = P;

% More than one exogenous variable

Q = linspace(0,10,N+nLags)';

exo = [exo Q];

S(ii) = S(ii-1) - 0.5\*S(ii-1);

E(ii) = E(ii-1) + 0.5\*S(ii-1) - 0.25\*E(ii-1);

I(ii) = I(ii-1) + 0.25\*E(ii-1) - 0.7\*I(ii-1)...

+ 0.1\*P(ii-1) + 0.1\*Q(ii-1);

Nope.

x1coeff =

0 0 0

5.0000e-01 5.0000e-01 0

0 7.5000e-01 2.5000e-01

0 0 3.0000e-01

0 0 1.1120e-01

0 0 8.8804e-02

lambdas =

15 16 10

# Summary

* Adding a second exogenous to the feature search space doesn't break it if it isn't actually included in the model specification.
* Adding a second exogenous feature to the model specification does break it for the following use cases:
  + The second exogenous variable is a different time series and a different coefficient from the first exogenous variable.
  + The second exogenous variable has a different time series but identical coefficient from the first exogenous variable.
  + The second exogenous variable is an exact duplicate of the first exogenous variable (same time series and same coefficient in model specification).
* With a model specification with one exogenous variable that can be successfully identified, changing the coefficient of the exogenous variable breaks the model identification.
  + i.e. Model identification is sensitive to (exogenous variable) coefficient values of the model specification.
  + This is consistent with previous finding that model identification is sensitive to endogenous variable coefficient values.
  + This suggests the importance of scaling/standardizing/transforming each time series in the feature library beyond the normalization routine already in place.
* When the model identification breaks in these cases, it is only (or almost only) the coefficients of the exogenous variables that are not successfully identified while the endogenous variable coefficients are successfully identified. [Why?]
* Keep in mind also the linear independence requirement of SINDy.