demo

August 11, 2017

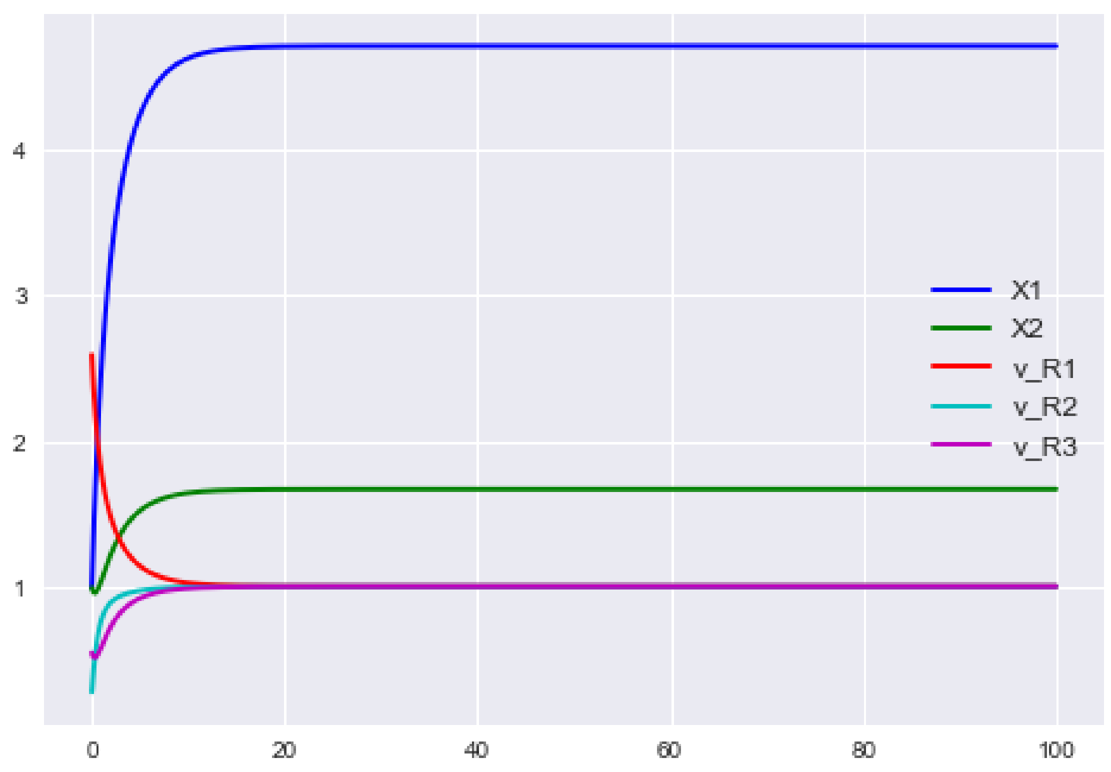
1 A demonstration of some basic computations using `infotopo` package

- Link: http://github.com/leihuang/infotopo
- Reference: https://github.com/leihuang/infotopo/blob/master/docs/PhDThesisLeiHuang.pdf (Section 2.7)
- About `SloppyCell`, around which much of the computation on reaction networks in `infotopo` is wrapped: http://sloppycell.sourceforge.net/user.pdf

```
In [1]: from __future__ import division

        import numpy as np

        from infotopo.models.rxnnet.examples.toynets import path3mmh as net
        from infotopo.models.rxnnet import experiments
        from infotopo import residual, fitting, sampling

        from util import butil
```

```
In [2]: # setting up the network
        net.set_var_ic('KE1', 5)
        net.set_var_ic('KE2', 4)
        p = net.p0.randomize(seed=0, sigma=1)
```

```
In [4]: # integrating the trajectory and plot it
        traj = net.get_traj((0,100), p=p)
        traj.plot(legends=traj.varids, legendloc='center right')
```



```
In [5]: # get steady-state concentrations
        s = net.get_s(p=p)
        print s
```

```
X1    4.704357
X2    1.673599
dtype: float64
```

```
In [6]: # Structural calculation: get stoichiometry matrix and steady-state flux vectors
        print net.N, '\n\n', net.K
```

```
     R1   R2   R3
X1  1.0 -1.0  0.0
X2  0.0  1.0 -1.0

    J1
R1   1
R2   1
R3   1
```

```
In [7]: # MCA calculation: get flux control matrix (rows should sum up to one by summation the
        nCJ = net.get_flux_ctrl_mat(p=p, normed=1)
        print nCJ, '\n\n', nCJ.sum(axis=1)
```

```
          log_v_R1  log_v_R2  log_v_R3
log_J_R1  0.276937  0.372677  0.350386
log_J_R2  0.276937  0.372677  0.350386
log_J_R3  0.276937  0.372677  0.350386

log_J_R1    1.0
log_J_R2    1.0
log_J_R3    1.0
dtype: float64
```

At the core of **information geometry** and **information topology** is the abstraction of mathematical models as functions (f) that map from parameters (p) to predictions (y):

$$f : p \mapsto y, \text{ or } y = f(p)$$

.

With an f specified, one can do a number of standard modeling tasks: * **Parameter estimation**: $p^* = f^{-1}(y)$ with uncertainty estimated as $\delta p = \left(Df^{-1}\right)\delta y$ * **Ensemble sampling**: eg, sampling posterior distribution $f(p|y) \propto f(p)f(y|p)$ * **Model comparison** (which includes **model selection** and **model reduction**): given f_1 and f_2 that both predict y, how does one compare them and which one is better?

```
In [8]: # get experiments objects that specify the model predictions (that is, y)
        expts_xc = experiments.get_experiments(net.xids, ['C1','C2'],
                                               us=butil.get_product(*[[1,2,3]]*2))
        expts_jc = experiments.get_experiments(net.Jids[0], ['C1','C2'],
                                               us=butil.get_product(*[[1,2,3]]*2))
        expts = expts_xc + expts_jc

        # combine model objects and experiments objects to get predict objects (that is, f)
        pred_xc = net.get_predict(expts_xc, tol_ss=1e-13)
        pred_jc = net.get_predict(expts_jc, tol_ss=1e-13)
        pred = net.get_predict(expts, tol_ss=1e-13)
```

Spectrum refers to the list of **singular values** of Df and the number of zeros tells how well-conditioned is f.

Here, the presence of two numerical zeros mean that f has two degrees of **structural nonidentifiability**.

```
In [9]: pred_xc.get_spectrum(p)

Out[9]: array([  1.20905442e+01,   4.93033641e+00,   2.52758635e-01,
                 1.39687397e-01,   1.01009238e-01,   3.91820667e-02,
                 8.71208005e-03,   2.85825399e-14,   3.90464196e-16])
```

```
In [10]: # generating simulation data
         dat = pred.get_dat(p=p)
         print dat
```

```
                                  Y  sigma
(C1=1, C2=1), X1, inf    2.946751      1
(C1=1, C2=1), X2, inf    1.810122      1
(C1=1, C2=1), J_R1, inf  0.578283      1
(C1=1, C2=2), X1, inf    3.415678      1
(C1=1, C2=2), X2, inf    2.923024      1
(C1=1, C2=2), J_R1, inf  0.419593      1
(C1=1, C2=3), X1, inf    3.719251      1
(C1=1, C2=3), X2, inf    3.971140      1
(C1=1, C2=3), J_R1, inf  0.326581      1
(C1=2, C2=1), X1, inf    5.312293      1
(C1=2, C2=1), X2, inf    2.366553      1
(C1=2, C2=1), J_R1, inf  0.845595      1
(C1=2, C2=2), X1, inf    6.115603      1
(C1=2, C2=2), X2, inf    3.616832      1
(C1=2, C2=2), J_R1, inf  0.655089      1
(C1=2, C2=3), X1, inf    6.658750      1
(C1=2, C2=3), X2, inf    4.772198      1
(C1=2, C2=3), J_R1, inf  0.539740      1
(C1=3, C2=1), X1, inf    7.452233      1
(C1=3, C2=1), X2, inf    2.820369      1
(C1=3, C2=1), J_R1, inf  1.016070      1
(C1=3, C2=2), X1, inf    8.545576      1
(C1=3, C2=2), X2, inf    4.179453      1
(C1=3, C2=2), J_R1, inf  0.811509      1
(C1=3, C2=3), X1, inf    9.301372      1
(C1=3, C2=3), X2, inf    5.422006      1
(C1=3, C2=3), J_R1, inf  0.685206      1
```

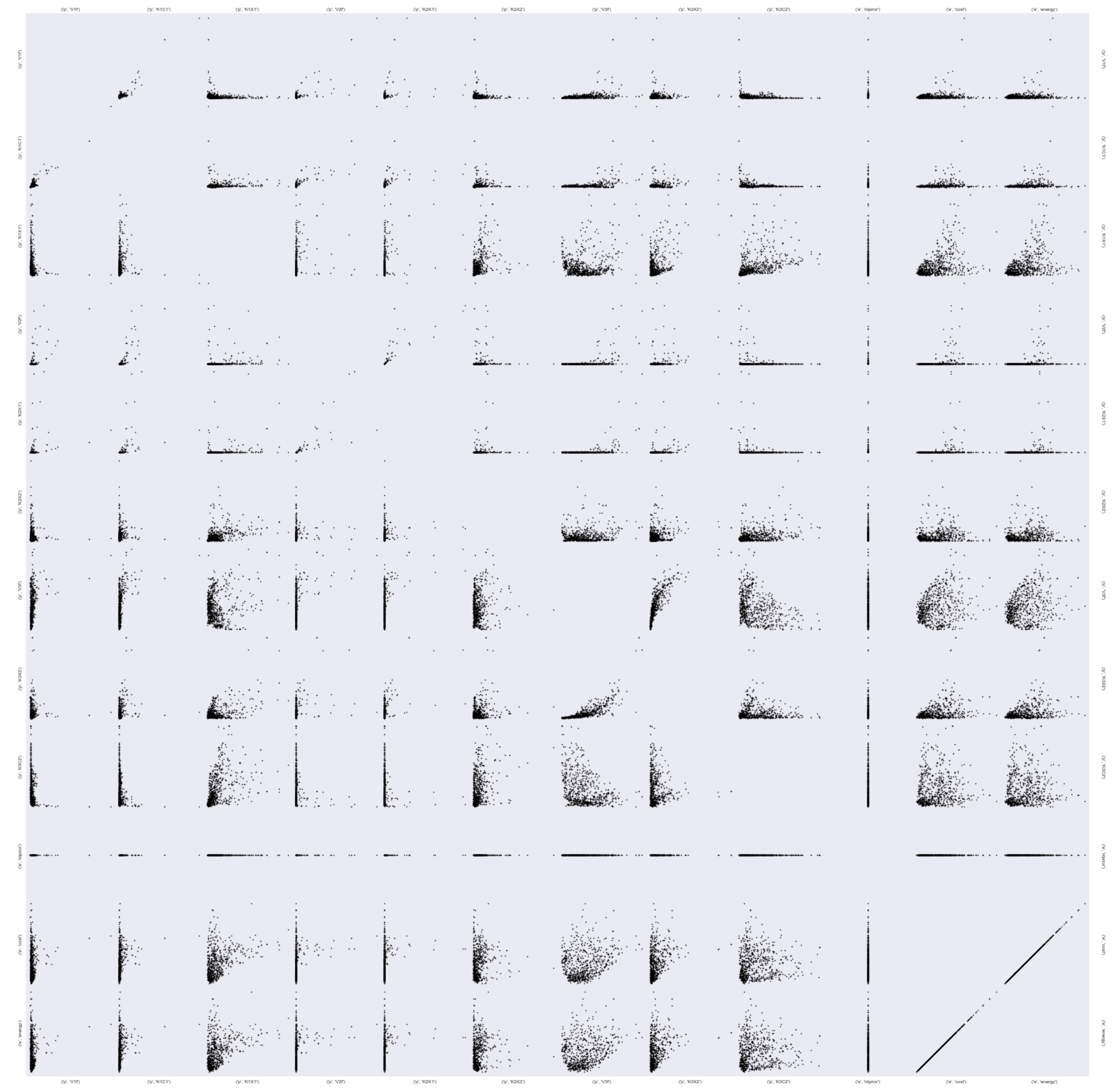
```
In [11]: # make a residual object that contains the predict object and data,
         # and do the fitting (using Levenberg-Marquardt algorithm)
         res = residual.Residual(pred=pred, dat=dat)
         fit = fitting.fit_lm_scipy(res, p0=p.randomize(sigma=0.2))
```

```
In [12]: # confirm that the best fit parameter recovers the original parameter used for genera
         fit.p.values - p.values
```

```
Out[12]: array([  1.86517468e-14,   4.44089210e-15,  -4.88498131e-15,
                  2.48689958e-14,   6.03961325e-14,   2.77555756e-15,
                  2.66453526e-15,   2.33146835e-15,   1.33226763e-15])
```

```
In [13]: # generating ensembles
         ens = sampling.sampling(res, 1000, stepscale=0.1)
```

```
In [15]: # scatter-plot the ensembles
         ens.scatter()
```


('p', 'V1f')
('p', 'K1C1')
('p', 'K1X1')
('p', 'V2f')
('p', 'K2X1')
('p', 'K2X2')
('p', 'V3f')
('p', 'K3X2')
('p', 'K3C2')
('e', 'nlprior')
('e', 'cost')
('e', 'energy')