Exact Methods for Computing Biological System Dynamics

Drew Endy (http://mit.edu/endy/)

Goals Covered Last Time

E. Review how the physics model leads to computational method

F. What is the complete computational method?

Goals for Today

G. What is the difference between a reaction rate and a reaction propensity?

H. How can we make this stuff compute faster?

55. What is the difference between a reaction rate and a reaction propensity?

Most folks think about and use continuous, approximate methods. Thus, most descriptions (models) of biological systems use continuous rate constants to describe reaction kinetics. Can we convert from continuous reaction rate constants to reaction propensities? Yes. The key is to look at the units of the constants.

Consider the following system: A <-> B+C

A -> B+C, with rate k_{off} = 1/time

B+C -> A, with rate k_{on} = 0.6E9/molar/time

k_{off} is a first-order reaction, whose rate is equivalent to the chance that an individual reaction event will take place, ***c***.

k_{on} is a second-order reaction. Here, the units don't match. We need to convert from 1/molar/second to 1/#/time. Molar has units of moles per volume. So, the units of k_{on} are 1/(moles/volume)/time. We can convert this to 1/#/time by multiplying the denominator by the (numbers of molecules / mole) and the volume of the system.

56. Here's an example. Consider:

B+C -> A, with rate k_{on} = 0.6E9/molar/time in a reaction the size of a bacteria (1E-15 L)

$\boldsymbol{c} = k_{on} / V/A_{\#}$ = 0.6E9 / [6E23#/mole x 1E-15L] = 1/#/time.

57. For higher order reactions...

$c = k / V^{(N-1)}/A_{\#}$, where N is the reaction order. [correct by N! for homogenous rxns]

58. Need for speed. Unfortunately, in practice both the Direct and First Reaction methods are incredibly slow / computational expensive. [e.g., back in 2001, you could compute 1E10 reaction events per day on an 800-MHz Pentium III processor via a Gibson-accelerated First Reaction Method (below); back then we estimated that there are 1E14-16 reaction events per bacterial cell doubling. So, 1E4-1E6 processor days to compute one cell doubling. Ugh!].

59. How can we make things go faster?

Gillespie's First Reaction Method costs o(2M) plus a sort per reaction event. The 2M costs come from computing M ***a***s and M RNDs (recall the M is the number of reactions defining the system).

Gillespie's Direct Method costs o(M)+2RNDs per reaction event. The M costs come from computing M ***a***s.

Do we need to do all this computation? Maybe there is a faster way! Michael Gibson, a graduate student in Shuki Bruck's group at Caltech recognized this problem in 2000 and figured out some cool hacks (published via *J. Phys. Chem. A* **2000,** *104,* 1876-1889).

Mike started with Gillespie's First Reaction method (developed last time):

i. Compute $\boldsymbol{a_i}$ for all reactions.

ii. Compute next reaction event times for all reactions via $\boldsymbol{\tau}$ = (1/***a***)*ln(1/***r***)

iii. Find the reaction that occurs first (hence the name of the method)

iv. Carry out the first reaction, update time to $\boldsymbol{\tau}$, and goto step i.

and replaced it with something called the "Next Reaction" method.

60. The big difference between the First and Next Reaction methods depends on the observation that when a single reaction event occurs, it might be the case that not everything in the system needs to be recomputed. From this "a ha," Mike developed an algorithm that implemented a faster method that's computationally equivalent to the the slower, original algorithms.

61. How do we know what needs updating? Reaction Graphs. Develop something called a "reaction graph" or a "dependancy graph." The reaction graph tells you what reaction's substrates change in number given the execution of any other reaction. [Consider an example system in class / by yourself].

62. What's the reaction graph good for? So, we can use the reaction graph to only update the time a reaction will occur for those reactions whose substrate abundances have changed. [Note that to do this we have to remember to store the ***a***s and ***t***s from before, whereas Gillespie's original methods didn't need to keep these in memory]. What's very cool is that we can update these times w/o re-sampling a PDF.

63. How do we update *t*s? First, recompute ***a*** for any reaction whose substrate numbers have changed. Then...

$$\mathbf{t}_{new} = (\boldsymbol{a}_{old} / \boldsymbol{a}_{new})(\mathbf{t}_{old} - \mathbf{t}) + \mathbf{t}$$

[Discuss above in class].

64. Then, for the reaction event which did occur... generate a new **t** via a new RNG as before.

65. Holy Parsimony! This method is only going to use new one random number per reaction event. And the overall cost is going to depend on the coupling in the reaction graph. Much better. Good enough to be useful in fact. [Note that there are some wasted computations -- reactions that never executed -- but these are noise in the grand scheme of things].

66. Summary of Next Reaction method's algorithm.

1a. Set initial numbers of molecules for all species, set time at zero, generate reaction dependancy graph.

1b. Compute $\boldsymbol{a_i}$ for all reactions

1c. Compute reaction event times for all reactions via $\boldsymbol{\tau} = (1/\boldsymbol{a})*\ln(1/\boldsymbol{r})$

1d. Store ***a*** and $\boldsymbol{\tau}$ values.

2. Find the reaction which will occur next. Carry out reaction and update time.

3a. Update ***a*** for any reaction whose substrate numbers have changed.

3b. Update times for any reaction whose ***a*** was updated.

3c. Compute new ***a*** and ***t*** for reaction that executed (a la 1c).

4. Go to step 2.

67. Comment. So, the Next Reaction algorithm uses computer memory and some smarts to dramatically accelerate a 25-year old algorithm. Sometimes it pays handsomely to consider how your computers are doing what you want them to be doing!