Documentation for rxnparam.h and rxnparam.c

Steve Andrews

Header file: rxnparam.h

```
#ifndef __rxnparam_h
#define __rxnparam_h

/***  LOOK-UP FUNCTIONS FOR REACTION RATES AND BINDING AND UNBINDING RADII
   ***/

double numrxnrate(double step,double a,double b);
double actrxnrate(double step,double a);
double bindingradius(double rate,double dt,double difc,double b,int rel);
double unbindingradius(double pgem,double dt,double difc,double a);

/************    FUNCTIONS FOR INVESTIGATING AN ABSORBING SPHERE
   ***********/

double rdfabsorb(double *r,double *rdf,int n);
void rdfdiffuse(double *r,double *rdfa,double *rdfd,int n,double step);
void rdfreverserxn(double *r,double *rdf,int n,double step,double b,double
   flux);
double rdfsteadystate(double *r,double *rdfa,double *rdfd,int n,double
   step,double b,double eps);
void rdfmaketable();

#endif
```

History

12/16/12 Added ".0" to values in rdfmaketable for C++ conformity.

How to use this library

This library file implements the algorithms that are described in Andrews and Bray, 2004, and is integral to the Smoldyn program. It is in the public domain, it is written in plain ANSII C, and its only dependencies are to the C Standard Library, in the hopes that others will be able to benefit from it as well. The first four functions that are listed above are the only ones that a user should ever need to use. They can be used to convert between macroscopic reaction parameters and the microscopic simulator parameters. The first set include the reaction rate constant, the reverse reaction rate constant, the activation-limited rate constant, and the probability of geminate recombination, while the latter set includes the binding and unbinding radii. The second set of functions in this library were used to generate lookup-tables that the first set uses. The second set should

never need to be run by typical library users, but are supplied so that users can see how the look-up tables were computed or can use them for further algorithm development.

For an irreversible bimolecular reaction, find the binding radius with

```
bindrad=bindingradius(rate,dt,dsum,-1,0);
```

`rate` is the macroscopic rate constant, `dt` is the simulator time step, and `dsum` is the sum of the two reactant diffusion coefficients. To verify that this returned the right number, test it with the inverse function

```
rate=numrxnrate(step,bindrad,0);
```

`step` is the mutual rms step length of the reactants, $(2(D_A+D_B)\Delta t)^{1/2}$, and `rate` should equal the rate that was entered in the previous example.

For the reversible A + B $\leftrightarrow$ C reaction, one needs to find both the binding radius for the A and B reactants of the forward reaction and the unbinding radius for the A and B products of the reverse reaction. To calculate these, the library requires additional information about the association reaction, which could be the unbinding radius, the ratio of the binding and unbinding radii, or the probability of geminate recombination. The last one is the only one that has macroscopic meaning. For these respective three reverse reaction parameters (called `rparam`), the function calls are

fixed unbinding radius

```
bindrad=bindingradius(rate,dt,dsum,rparam,0);
unbindrad=rparam;
```

fixed ratio of unbinding radius to binding radius

```
bindrad=bindingradius(rate,dt,dsum,rparam,1);
unbindrad=bindrad*rparam;
```

fixed probability of geminate recombination

```
bindrad=bindingradius(rate*(1-rparam),dt,dsum,-1,0);
unbindrad=unbindingradius(rparam,dt,dsum,bindrad);
```

For all three options, check that the correct rate is calculated with the inverse function,

```
rate=numrxnrate(step,bindrad,unbindrad);
```

A final option for reversible reactions that Smoldyn allows is for a maximum probability of geminate recombination. This is the default option for Smoldyn, for which the `rparam` value is rather arbitrarily set to 0.2. The code fragment for addressing it is

maximum probability of geminate recombination

```
bindrad=bindingradius(rate,dt,dsum,0,0);
unbindrad=unbindingradius(rparam,dt,dsum,bindrad);
if(unbindrad>0) bindrad=bindingradius(rate*(1.0-rparam),dt,dsum,-1,0);
```

All of these example are used in Smoldyn. They can be found in the smolreact.c source code file, in the functions `rxnsetrate`, `rxnsetproduct`, and `rxncalcrate`.

Math for interpolation functions

Interpolation is done in a few functions here with simple cubic interpolation. This might be formally identical to the cubic spline algorithm that is described in *Numerical Recipes in C*, although it is a completely different implementation and so might lead to different results. My implementation is probably not as fast for long lists of input values, but has more transparent code, is fully contained within a single function, and is probably nearly as fast as their method for only a few input values.

Consider a input vectors X_i and Y_i, which define the known values and the input value x for which the unknown y is wanted. First, the position of x in the X_i vector is found, with the four X_i values that surround x are copied over into x_0 to x_3 such that x is between x_1 and x_2. Then, Lagrange's formula (see Numerical Recipies in C), is used to define the four polynomial coefficients as:

$$z_0 = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = -\frac{(x-x_1)(x-x_2)(x-x_3)}{6\Delta x^3}$$

$$z_1 = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = \frac{(x-x_0)(x-x_2)(x-x_3)}{2\Delta x^3}$$

$$z_2 = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = -\frac{(x-x_0)(x-x_1)(x-x_3)}{2\Delta x^3}$$

$$z_3 = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = \frac{(x-x_0)(x-x_1)(x-x_2)}{6\Delta x^3}$$

The first form allows unequally spaced x values, whereas the latter form assumes constant spacing on x of Δx. For interpolation on just one axis, the result is

$$y = z_0y_0 + z_1y_1 + z_2y_2 + z_3y_3$$

It doesn't matter if the input x value is too close to an edge of the tabulated data for it to be centered among 4 values, because exactly the same method is done if it is just one value in from the edge, or if extrapolation is required to go beyond the edge. For interpolation on two axes, interpolation is done four times on one axis to find 4 new y values; then it is done once on the other axis with these four y values. This is fairly obvious if a grid is drawn. While I haven't proven it, I suspect that the exact same result is gotten regardless of which axis is interpolated on first.

Look-up functions for reaction rates and binding and unbinding radii

The functions in this section are used by Smoldyn, and may be useful for other programs. They return binding and unbinding radii for bimolecular reactions, assuming a numerical version of the Smoluchowski model of chemical reactions, in which fixed size time steps are used.

`float numrxnrate(float step,float a,float b);`

`numrxnrate` calculates the bimolecular reaction rate for the simulation algorithm, based on the rms step length in `step`, the binding radius in `a`, and the unbinding radius in `b`. It uses cubic polynomial interpolation on previously computed data, with interpolation both on the reduced step length (`step`/`a`) and on the reduced unbinding radius (`b`/`a`). Enter a negative value of `b` for irreversible reactions. If the input parameters result in reduced values that are off the edge of the tabulated data, analytical results are used where possible. If there are no analytical results, the data are extrapolated. Variable components: `s` is step, `i` is irreversible, `r` is reversible with `b`>`a`, `b` is reversible with `b`<`a`, `y` is a rate. The returned value is the reaction rate times the time step Δt. In other words, this rate constant is per time step, not per time unit; it has units of length cubed.

`float actrxnrate(float step,float a);`

`actrxnrate` calculates the effective activation-limited reaction rate for the simulation, which is the reaction rate if the radial correlation function is 1 for all $r > a$. The returned value needs to be divided by Δt. The equation is

$$k_a = \frac{4\pi}{3}\left[\operatorname{erfc}\frac{\sqrt{2}}{s} + s\sqrt{\frac{2}{\pi}}\right] + \frac{2\sqrt{2\pi}}{3}s\left(s^2-1\right)\left[\exp\left(-\frac{2}{s^2}\right)-1\right]$$

It was calculated analytically and verified numerically.

`float bindingradius(float rate,float dt,float difc,float b,int rel);`

`bindingradius` returns the binding radius that corresponds to some given information. `rate` is the actual rate constant (not reduced), `dt` is the time step, and `difc` is the mutual diffusion constant (sum of reactant diffusion constants). If `b` is -1, the reaction is assumed to be irreversible; if `b` ≥ 0 and `rel` = 0, then the `b` value is used as the unbinding radius; and if `b` ≥ 0 and `rel` = 1, then the `b` value is used as the ratio of the unbinding to binding radius, `b`/`a`. This algorithm executes a simple search from `numrxnrate`, based on the fact that reaction rates monotonically increase with increasing `a`, for all the `b` value possibilities. The return value is usually the binding radius. However, a value of -1 signifies illegal input parameters.

Modified 2/22/08 to allow for `dt` = 0.

`float unbindingradius(float pgem,float dt,float difc,float a);`

`unbindingradius` returns the unbinding radius that corresponds to the geminate reaction probability in `pgem`, the time step in `dt`, the mutual diffusion constant in `difc`, and the binding radius in `a`. Illegal inputs result in a return value of -2. If the geminate binding probability can be made as high as that requested, the corresponding unbinding radius is returned. Otherwise, the negative of the maximum achievable `pgem` value is returned.

Modified 2/25/08 to allow for `dt` = 0.

Functions for investigating an absorbing sphere

The functions in this section were used to calculate the look-up tables that are used in the preceding functions. They are not used by Smoldyn. It is not anticipated that they will be required for most other programs either, although they may be useful for improving the look-up tables or for additional algorithm development.

```
double rdfabsorb(double *r,double *rdf,int n);
```

`rdfabsorb` integrates the radial diffusion function (rdf) for $0 \le r \le 1$, sets those values to 0, and returns the integral. `r` is a vector of radii; `r[0]` may equal zero but that is not required; if not, then it is assumed that the rdf has zero slope at the origin.

Integration uses a spherical version of the trapezoid rule: at positions r_0 and r_1, the function f has values f_0 and f_1, leading to the linear interpolation

$$f = \frac{(r-r_0)f_1 + (r_1-r)f_0}{r_1-r_0}$$

$$A = \int_0^{r_1} 4\pi r^2 f(r)\,dr$$

$$= \frac{4\pi}{r_1-r_0}\left[\frac{(f_1-f_0)(r_1^4-r_0^4)}{4} + \frac{(r_1f_0-r_0f_1)(r_1^3-r_0^3)}{3}\right]$$

$$= \pi(f_1-f_0)(r_1+r_0)(r_1^2+r_0^2) + \frac{4\pi}{3}(r_1f_0-r_0f_1)(r_1^2+r_1r_0+r_0^2)$$

The left end of the integral assumes zero slope for the rdf. The right end does not terminate exactly at 1, but includes the upper left triangle of the final trapezoid. That way, if there are two absorptions in a row, the second one will return an integral of 0, and area is properly conserved. The problem is that it does not terminate exactly at 1. Furthermore, the correct relative location of 1 between two `r[j]` points depends on the function. The best solution is to use an unevenly spaced `r[j]` vector, with a very narrow separation about 1 and no `r[j]` equal to 1.

```
void rdfdiffuse(double *r,double *rdfa,double *rdfd,int n,double step);
```

`rdfdiffuse` integrates the radial distribution function with the Green's function for radially symmetric diffusion to implement diffusion over a fixed time step. `r` is a vector of radii, `rdfa` is the input rdf, `rdfd` is the output rdf, `n` is the number of points, and `step` is the rms step length, equal to $(2Dt)^{1/2}$. `r[0]` may equal 0 but it is not required. It is assumed that `rdfa` has zero slope at $r = 0$. The boundary condition on the large r side is that the function tends to 1 with a functional form $1+a_2/r$, for large r. This is accomplished by fitting the 10% largest r portion of the rdf with the function $1+a_2/r$. After the integral over the tabulated data is complete, the rdf is integrated on to infinity using the previous fit information and an analytical result for that integral. The numerical portion of the integral is carried out exactly like the one in `rdfabsorb` but with a different integrand, which is

$$c(r) = \int_0^\infty 4\pi r'^2 \,\mathrm{rdfa}(r')\,\mathrm{grn}(r,r')\,dr'$$

$grn(r,r')$ is the Green's function, equal to

$$\mathrm{grn}(r,r') = \frac{1}{4\pi r r'}\left[G_s(r-r') - G_s(r+r')\right]$$

and $G_s(x)$ is a normalized Gaussian with mean 0 and standard deviation s. It is

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{x^2}{2\sigma^2}\right)$$

```
void rdfreverserxn(double *r,double *rdf,int n,double step,double b,double
     flux);
```

Analysis of the reverse reaction involves adding a delta function to the rdf and then convolving with the Green's function. However, this leads to numerical errors, although it is trivial analytically. The function `rdfreverserxn` is the analytic solution. It adds the diffusion Green's function to the rdf, based on a delta function at b, and after one diffusion step. `r` is a list of radii, `rdf` is the rdf, `step` is the rms step length, `b` is the delta function point (which does not have to be equal or unequal to a `r[j]` value), and `flux` is the area of the delta function.

```
double rdfsteadystate(double *r,double *rdfa,double *rdfd,int n,double
     step,double b,double eps);
```

`rdfsteadystate` calculates the radial distribution function (rdf) for alternating absorption and diffusion steps, for either irreversible or reversible reactions. `r` is a vector of radii, `rdfa` is input as a trial rdf and output as the result after absorption, `rdfd` is ignored on input but is output as the rdf after diffusion, `n` is the number of elements in the vectors, `step` is the rms step length, and `b` is either <0 if the reaction is irreversible or is the unbinding radius if the reaction is reversible. It executes until the fractional difference between successive steps is less than `eps`, but at least 30 times and no more than `maxit` times. It can also exit if it iterates more than `maxit`

times before converging or if the flux exceeds `maxflux`; if either of these happens, the function returns -1.

`void rdfmaketable();`

`rdfmaketable` is used to create data tables of reaction rates, including those used above in `numrxnrate`. All input is requested from the user using the standard input and all output is sent to standard output. Runtime is about 1 minute with mode `i`, 200 pts, `eps`=1e-4 (on a Mac G4 laptop).