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);
double rdfabsorbprob(double *r,double *rdf,int n,double prob);
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.
2/6/17	Added `rdfabsorbprob` function.
3/20/18	Added probability input to `actrxnrate`.
3/22/18	Added `rdfmodelrdf` function.
9/17/18	Largely finalized `numrxnrateprob` and `bindingradiusprob` functions

Computation errors

I have checked the results from this library quite thoroughly and have found good agreement between simulations and theory in nearly all cases. However, it still returns incorrect results in a few cases. (1) If rms step lengths are less than about $0.05\sigma_b$, then the lookup functions are off of their data tables and so have to either extrapolate tabulated results or use analytical theory. This often works well, but the theory seems to be incorrect if $\sigma_u \sim \sigma_b$. (2) If $\sigma_u < \sigma_b$ and rms step lengths are very small, then the reaction rate is enormous and is again off of the tabulated region. Here, results from `numrxnrate` use cubic interpolations from the known data and those from `numrxnrateprob` use linear interpolations, which can lead to very different results due to the highly non-linear structures of the data. These errors can be explored using the SmolEmulate utility program. See the SmolEmulate documentation for further details.

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 ↔ 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

<u>fixed unbinding radius</u>

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

<u>fixed ratio of unbinding radius to binding radius</u>

```
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 cubic 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 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, then 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 Recipes 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

$$y = z_0 y_0 + z_1 y_1 + z_2 y_2 + z_3 y_3$$

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.

Utility functions

```
double modelrxnrate(double a,double b,double difc,double chi);
```

This returns the steady-state reaction rate constant for continuous time models. Enter the binding radius in `a`, the unbinding radius in `b`, the mutual diffusion coefficient in `difc`, and the fraction diffusion-limited in `chi`. For the Smoluchowski model, set `chi` to 1. For irreversible reactions, set `b` to -1.

The returned functions are

irreversible: $k = 4\pi D\sigma_b\chi$

reversible: $$k = \frac{4\pi D\sigma_b\sigma_u\chi}{\sigma_u - \sigma_b} = \frac{4\pi D\sigma_b\chi}{1-\phi_S}$$

Note that $\phi_S = \sigma_b/\sigma_u$ is the probability of geminate recombination in the Smoluchowski model. The function returns -1 if $\sigma_u \le \sigma_b$, which should be interpreted as a value of infinity.

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.

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

`numrxnrate` calculates the bimolecular reaction rate constant 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.

```
double numrxnrateprob(double step,double a,double b,double prob);
```

Calculates the simulated bimolecular reaction rate for a given rms step length, in `step`, the binding radius in `a`, unbinding radius in `b`, and reaction probability in `prob`. Enter a negative value for `b` for irreversible reactions. If `prob` equals 1, this calls `numrxnrate` for the answer instead. This also caps the returned result so that it is always less than or equal to that for `numrxnrate`; it would be larger in some cases, generally when the probability is close to 1, due to differences between linear and cubic interpolation.

This uses linear interpolation between tabulated values, with interpolation on the reduced step length, reduced unbinding radius, and reaction probability. As with `numrxnrate`, variables are `s` for step, `i` for irreversible reactions, `r` for reversible with `b>a`, and `b` for reversible with `b<a`. This uses `k` for rate. Tabulated data are: `kirr` has rows with probability and columns with step length (e.g. index it with `kirr[pindx*snum+sindx]`), `krev` has layers with `b` value, rows with probability, and columns with `step` (e.g. index as `krev[bindx*snum*pnum+pindx*snum+sindx]`), and `kb` has the same as `krev`. The tabulated data were computed by the `rdfmaketableprob` function, called by the SmolEmulate program. Note that the tabulated data are in order of increasing unbinding radii, probabilities, and steps, except for the `kb` table, which uses decreasing steps.

For linear interpolation, suppose we have a known **x** vector. Define **a** as a table coordinate in which all components are less than those of **x** and **b** as a table coordinate in which all components are larger than those of **x**. In 2-D for simplicity, the weights of the 4 adjacent table points are

$$w_{00} = \frac{(b_1 - x_1)(b_2 - x_2)}{(b_1 - a_1)(b_2 - a_2)} \qquad w_{01} = \frac{(b_1 - x_1)(x_2 - a_2)}{(b_1 - a_1)(b_2 - a_2)}$$

$$w_{10} = \frac{(x_1 - a_1)(b_2 - x_2)}{(b_1 - a_1)(b_2 - a_2)} \qquad w_{00} = \frac{(x_1 - a_1)(x_2 - a_2)}{(b_1 - a_1)(b_2 - a_2)}$$

This extends trivially for higher dimensions. Then combine these to give the interpolated y value as

$$y = w_{00}y_{00} + w_{01}y_{01} + w_{10}y_{10} + w_{11}y_{11}$$

Again, this extends trivially for higher dimensions.

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.

Function status as of 6/28/18: everything seems to basically work. However, the data quality could be improved, and also perhaps changed for better sampling points. Graphing the results shows artifacts when b=1 and b<1, especially for small step values.

```
double actrxnrate(double step,double a,double prob);
```

`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

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

It was calculated analytically and verified numerically. This equation, without the reaction probability, appears in Andrews and Bray, 2004. This function is largely obsolete because I now feel that the activation-limited reaction rate is better defined through the Noyes relation ($k^{-1} = k_S^{-1}+k_a^{-1}$).

```
double bindingradius(double rate,double dt,double difc,double 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.

If σ_u is known, enter `b` = σ_u and `rel` = 0.
If σ_u/σ_b is known, enter `b` = σ_u/σ_b and `rel` = 1.
If ϕ (geminate recombination) is known, `rate` = $k(1-\phi)$, `b` = 0, and `rel` = 0.

```
double bindingradiusprob(double rate,double dt,double difc,double b,int
    rel,double chi,double *probptr);
```

This is the same as `bindingradius`, but allows non-unit reaction probabilities (i.e. it's for the λ-ρ algorithm rather than the Andrews-Bray algorithm). `rate` is the actual rate constant (not reduced), `dt` is the time step, `difc` is the mutual diffusion coefficient, `b` is the unbinding radius (or send in `b` < 0 for irreversible reaction), `rel` tells whether the b value is relative to the binding radius or is absolute, and `chi` is the diffusion-limited fraction. Enter `chi` as a negative value to indicate that it's not known; if it's not known and no probability is entered in `probptr`, then this function calls `bindingradius` instead. For most uses, enter `probptr` as a pointer to a reaction probability value that will get set by this algorithm; however, if `chi` is entered negative, then enter the reaction probability with `probptr`. This returns the binding radius directly and the reaction probability pointed to by `probptr`.

If `dt` is set to 0, then this function uses analytical equations to compute the binding radius and it returns the λ-ρ algorithm λ value with `probptr`. To do so, this function computes the reduced λ-ρ algorithm reaction rate constant λ' using the function

$$\lambda' = \frac{3.18243\chi - 3.40864\chi^2 + 0.984385\chi^3}{(1-\chi)^{2.08761}}$$

from my Notes18C06 file, and then unreduces it.

The function runs a greedy directed search algorithm. It cycles through the search parameters (`a` and `prob`, here) up to 300 times each, or until the error is less than 1 part in 10^6. At each iteration, it tries moving in the same direction as last time. If that improved the result, it keeps the result and increases the step size for next time; if it led to a worse result, it keeps the prior result, reverses direction, and substantially reduces the step size. Tests on this procedure shows that it works remarkably well, with the error typically oscillating at first and then decreasing roughly log-linearly.

```
double unbindingradius(double pgem,double dt,double difc,double 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 rdfmodelrdf(double r,double sigmab,double sigmau,double lambdap,double
     gamma);
```

Computes the analytical radial distribution function for the Smoluchowski, Collins and Kimball or Doi models, for either reversible or irreversible reactions. Equations are from my Notes18C06 document. Enter the radius to be evaluated in `r` and the binding radius in `sigmab`. Enter the unbinding radius in `sigmau` (which must be larger than `sigmab`) for reversible reactions or as a negative value for reversible reactions. Enter the Collins and Kimball boundary parameter in `gamma` if that model is used or enter `gamma` as a negative value if that model is not used. Enter the reduced Doi model reaction rate in `lambdap` if that model is used or enter `lambdap` as a negative number if it is not used.

For the Collins and Kimball model, the boundary condition is

$$\left.\frac{\partial g(r)}{\partial r}\right|_{\sigma_b}=\frac{g(\sigma_b)}{\gamma}$$

Sometimes κ is used instead, where $\gamma = D/\kappa$ and D is the diffusion coefficient. For the Doi model, the reduced reaction rate is

$$\lambda' = \sigma_b\sqrt{\frac{\lambda}{D}}$$

where λ is the unreduced reaction rate.

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

`rdfabsorb` integrates the radial diffusion function (rdf) for $0 \le r \le$ `bindrad`, sets those values to 0 (if `prob` equals 1), and returns the integral. Absorption can also be probabilistic, with probability `prob`. `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. Conceptually, this returns

$$flux = P\int_0^{\sigma_b} 4\pi r^2 g(r)\,dr$$

where P is the probability, and $g(r)$ is the rdf.

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

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

$$\begin{aligned}A &= \int_{r_0}^{r_1} 4\pi r^2 f(r)\,dr\\ &= \frac{4\pi}{r_1-r_0}\left[\frac{(f_1-f_0)(r_1^4-r_0^4)}{4}+\frac{(r_1f_0-r_0f_1)(r_1^3-r_0^3)}{3}\right]\\ &= \pi(f_1-f_0)(r_1+r_0)(r_1^2+r_0^2)+\frac{4\pi}{3}(r_1f_0-r_0f_1)(r_1^2+r_1r_0+r_0^2)\end{aligned}$$

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.

If `prob` is not entered as 1, the area returned is `prob` times the total area between 0 and `bindrad`, and the rdf is modified to represent `prob` fraction of the area removed.

`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

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

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

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

Note that the Green's function is given in Carslaw and Jaeger, 14.7(1) (page 366). To convert from their notation to mine, replace κ with D and $(2Dt)^{1/2}$ with s. Also, in the limit that r goes to 0, the Green's function approaches

$$\mathrm{grn}(0,r')=\frac{1}{2\pi s^2}G_s(r')$$

Math for fitting the final 10% of the rdf to the function $y = a_1 + a_2/r$. (From my Cambridge notes p. G-5.) Following Numerical Recipes, define:

$$A = \begin{bmatrix} 1 & 1/x_1 \\ 1 & 1/x_2 \\ 1 & 1/x_3 \\ \vdots & \vdots \end{bmatrix} \qquad b = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} \qquad \alpha = A^T A \qquad \beta = A^T b$$

Best fit is for a1 and a2 values given with

$$A^T A \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = A^T b \quad \text{so} \quad \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$$

which solves to

$$a_1 = \frac{\beta_1 \alpha_{22} - \beta_2 \alpha_{12}}{\alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}} \qquad a_2 = \frac{\beta_2 \alpha_{11} - \beta_1 \alpha_{21}}{\alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}}$$

with

$$\alpha_{11} = n \qquad \alpha_{12} = \alpha_{21} = \sum_i \frac{1}{x_i} \qquad \alpha_{22} = \sum_i \frac{1}{x_i^2} \qquad \beta_1 = \sum_i y_i \qquad \beta_2 = \sum_i \frac{y_i}{x_i}$$

This function uses a simplified version of this derivation, for the fitting function $y = 1 + a_2/x$, or $y - 1 = a_2/x$. The above variables simplify to

$$A = \begin{bmatrix} 1/x_1 \\ 1/x_2 \\ \vdots \end{bmatrix} \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix} \qquad \alpha = \sum_i \frac{1}{x_i^2} \qquad \beta = \sum_i \frac{y_i - 1}{x_i} \qquad a_2 = \frac{\beta}{\alpha}$$

```
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 a,double b,double eps,double prob);
```

`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, `a` is the binding radius, 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`. Most 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.

`void rdfmaketableprob();`

Makes data tables of reaction rates as a function of reduced rms step lengths, unbinding radii, and reaction probabilities. Most input is requested from the user from the standard input and all output is sent to standard output. In most cases, the data are best exported to Excel, manipulated there, reformatted, and then pasted into the `numrxnrateprob` function.