Documentation for SurfaceParam.h and SurfaceParam.c

Steve Andrews

2008 - 2023

1 Header file: SurfaceParam.h

```c
/* File SurfaceParam.h, written by Steven Andrews, 2008.
This code is in the public domain. It is not copyrighted and may not be
copyrighted.
This is a header file for SurfaceParam.c. */

#ifndef __srfparam_h
#define __srfparam_h

/********************** FUNCTIONS FOR EXTERNAL USE **********************/

enum SurfParamAlgo {SPAirrTrans,SPAirrTransT,SPAirrTransQ,SPArevTrans,SPAirrAds,
    SPAirrAdsT,SPAirrAdsQ,SPAirrAdsEC,SPArevAds,SPArevAdsND,SPAirrDes,SPArevDes,
    SPAirrFlip,SPArevFlip,SPAirrDesC,SPArevAdsC};

double surfaceprob(double k1,double k2,double dt,double difc,double *p2ptr,enum
    SurfParamAlgo algo);
double desorbdist(double step,enum SurfParamAlgo algo);
double surfacerate(double p1,double p2,double dt,double difc,double *k2ptr,enum
    SurfParamAlgo algo);
int surfacetransmit(double *kap1ptr,double *kap2ptr,double *p1ptr,double *p2ptr,
    double difc1,double difc2,double dt);

/********************** PARAMETER CALCULATION FUNCTIONS ********************/

double lookupirrevadsorb(double value,int pfromk);
double lookuprevadsorbnd(double probon,double proboff);
double lookuprevads(double value1,double value2,int pfromk,double *ans2ptr);
double lookuprevtrans(double pf,double pb,double *kbptr);

/******* FUNCTIONS FOR INVESTIGATING A PARTIALLY ADSORBING SURFACE *******/

void xdfdiffuse(double *x,double *xdfa,double *xdfd,int n);
double xdfadsorb(double *x,double *xdf,int n,double probon);
void xdfdesorb(double *x,double *xdf,int n,double b,double flux);
void xdfdesorbdelta(double *x,double *xdf,int n,double b,double flux);
double xdfsteadystate(double *x,double *xdfa,double *xdfd,int n,double cs,double b
    ,double probon,double proboff,double eps);
void xdfmaketableirrev(void);
void xdfmaketable(void);

#endif
```

2 History

2008-2009 Developed theory, wrote software, and wrote initial documentation.

11/19/10 Fixed a minor bug in `surfaceprob` to allow a wider range of input values in the `SPArevTrans` section.

6/6/23 Converted documentation to LaTeX.

6/10/23 Fixed a minor bug in `surfaceprob` to address the possibility of probabilities exceeding 1.

3 Introduction

This file converts between molecule-surface interaction coefficients on the one hand, and the appropriate simulator probabilities on the other. It implements the algorithms that I described in the research paper "Accurate particle-based simulation of adsorption, desorption, and partial transmission" and published in Physical Biology in 2009.

As a default, this file depends upon my library files math2.c and random2.c. The latter file uses the SFMT random number generator, which is fast and very high quality. Alternatively, if two "#include" lines that are near the top of the SurfaceParam.c source code file are commented out, this will compile with no dependencies other than standard C library files. Doing this simply uses local versions of some functions and also uses the built-in random number generator instead of the SFMT one.

The file is divided into several sections, which are described below.

4 Functions for external use

The functions in this section are used by Smoldyn, and may be useful for other programs. They return adsorption and desorption probabilities, desorption initial separation distances, partial transmission probabilities, and partial transmission initial separation distances. Which of these items is returned is chosen with a parameter called `algo`, which is one of the `SurfParamAlgo` enumerated values.

Enumerated type

```
enum SurfParamAlgo {SPAirrTrans, SPAirrTransT, SPAirrTransQ, SPArevTrans, SPAirrAds,
    SPAirrAdsT, SPAirrAdsQ, SPAirrAdsEC, SPArevAds, SPArevAdsND, SPAirrDes, SPArevDes,
    SPAirrFlip, SPArevFlip, SPAirrDesC, SPArevAdsC;}
```

These enumerations are used as shown below.

Functions

```
double surfaceprob(double k1, double k2, double dt, double difc, double *p2ptr, enum
    SurfParamAlgo algo);
```

Returns the transition probabilities for molecules that interact with surfaces. `k1` and `k2` are interaction coefficients, `dt` is the simulation time step, `difc` is the molecule diffusion coefficient, and `p2ptr` is used for the function to return a second interaction probability. `algo` specifies which algorithm should be followed. Returns the probability, which is always between 0 and 1 inclusive, or -1 if `algo` isn't recognized. Enter `k1` and/or `k2` as -1 to indicate that they their values are infinite; in this case, the corresponding probability is typically set to 1 and the probability for the reverse action, if applicable, is set to achieve the correct equilibrium concentration. The algorithms and variable interpretations are:

algo	meaning	k1	k2	returns	*p2ptr
SPAirrTrans	irreversible transmission	κ_F	0	P_F	0
SPAirrTransT	" - table look-up	"	"	"	"
SPAirrTransQ	" - quartic equation	"	"	"	"
SPArevTrans	reversible transmission	κ_F	κ_B	P_F	P_B
SPAirrAds	irreversible adsorption	κ	0	P_a	0
SPAirrAdsT	" - table look-up	"	"	"	"
SPAirrAdsQ	" - quartic equation	"	"	"	"
SPAirrAdsEC	" - Erban-Chapman	"	"	"	"
SPArevAds	reversible adsorption	κ	k	P_a	P_d
SPArevAdsND	" - no displacement	"	"	"	"
SPAirrDes	irreversible desorption	k	sum	P_d	0
SPArevDes	reversible desorption	k	κ	P_d	P_a
SPAirrFlip	irreversible flipping	k	sum	P	0
SPArevFlip	reversible flipping	k_{fwd}	k_{rev}	P_{fwd}	P_{rev}

Several algorithms are listed with variants. For example, `IrrAds` uses the default method for calculating irreversible adsorption parameters (table interpolation), but one can also explicitly specify table interpolation with `IrrAdsT`, best-fit quartic with `IrrAdsQ`, or the Erban-Chapman equation with `IrrAdsEC`. Similarly, `IrrTransT` and `IrrTransQ` are table interpolation and best-fit quartic variants for irreversible transmission. `RevAdsND` is for reversible adsorption, but is for no initial displacement of desorbed molecules; this uses the irreversible desorption equation for desorption and then adjusts the adsorption probability to yield the correct equilibrium concentration.

Flipping means a state change for a surface-bound molecule, such as membrane-orientation flipping; this is identical to a simple first order reaction, but is included here for convenience. For irreversible flipping, k is the flipping reaction rate and sum is the sum of the rates of all first order reactions that the reactant can undergo. This sum value correctly adjusts the probability to account for competing processes (it is also required for irreversible desorption, with exactly the same interpretation). If the listed state change is the only one that is possible, then enter sum as either k or 0. For reversible flipping, k_{fwd} is the reaction rate constant for leaving the current state, while k_{rev} is the reaction rate constant for returning to the current state.

A minor bug was fixed in the `SPArevTrans` section on11/19/2010 to enable it to use a wider range of input values. Another bug was fixed in the `SPArevTrans` section 6/10/23 to clamp the output probabilities to the range from 0 to 1; this maintains the κ'_F/κ'_b ratio, so equilibrium constants are not affected, although actual transition rates will be lower than those entered here. These changes became moot on 7/12/23, when this portion of the function was commented out and sent to `surfacetransmit` instead.

```
double desorbdist(double step,enum SurfParamAlgo algo);
```

Returns a random distance away from a surface that a molecule should be desorbed to, according to the correct probability density for algorithm `algo`. `step` is the rms step length for a molecule (which is $\sqrt{2D\Delta t}$, where D is the diffusion coefficient and Δt is the time step). In addition, entering `algo` as `SPAirrDesC` or `SPArevAdsC` returns the (non-random) characteristic initial distances for these algorithms. Returns -1 for an unrecognized `algo` variable.

algo	meaning	returns
SPAirrDes	irreversible desorption	random distance
SPAirrDesC	”	characteristic distance
SPArevAds	reversible adsorption	random distance
SPArevAdsC	”	characteristic distance

`double surfacerate(double p1, double p2, double dt, double difc, double *k2ptr, enum SurfParamAlgo algo);`

This function is the inverse of `surfaceprob`. Enter the interaction probabilities in `p1` and `p2`, the time step in `dt`, the solution-phase diffusion coefficient in `difc`, and the algorithm in `algo`. This function then returns the rates directly and, if `k2ptr` is not `NULL`, pointed to by `k2ptr`.

algo	meaning	p1	p2	returns	*k2ptr
SPAirrTrans	irreversible transmission	P_F	0	κ_F	0
SPAirrTransT	” - table look-up	”	”	”	”
SPAirrTransQ	” - quartic equation	”	”	”	”
SPArevTrans	reversible transmission	P_F	P_B	κ_F	κ_B
SPAirrAds	irreversible adsorption	P_a	0	κ	0
SPAirrAdsT	” - table look-up	”	”	”	”
SPAirrAdsQ	” - quartic equation	”	”	”	”
SPAirrAdsEC	” - Erban-Chapman	”	”	”	”
SPArevAds	reversible adsorption	P_a	P_d	κ	k
SPArevAdsND	” - no displacement	”	”	”	”
SPAirrDes	irreversible desorption	P_d	sum	k	0
SPArevDes	reversible desorption	P_d	P_a	k	κ
SPAirrFlip	irreversible flipping	P	sum	k	0
SPArevFlip	reversible flipping	P_{fwd}	P_{rev}	k_{fwd}	k_{rev}

In `SPAirrFlip` and `SPAirrDes`, *sum* is the sum of the probabilities of transitioning away from the starting state. If there is only way to leave the starting state, then enter *sum* either equal to P or as 0.

`int surfacetransmit(double *kap1ptr, double *kap2ptr, double *p1ptr, double *p2ptr, double difc1, double difc2, double dt);`

This expands upon `surfaceprob` and `surfacerate` for partial transmission, offering additional functionality. This function allows for, but doesn't require, different diffusion coefficients on the two sides of the surface, entered in `difc1` and `difc2`. Also, `dt` is the simulation time step. The four transmission parameters are κ_1, κ_2, P_1, and P_2, of which one side 1 value and one side 2 value should be entered as inputs, and the other two values will be returned as outputs; the side number represents the side that a molecule diffuses *from*. All of these values are entered as pointers due to the fact that two of them will be outputs. Enter the output parameters with the value -2 to show that they are unknowns. The input parameters are not changed by this function.

For example:

```
kap1=kap2=1;
```

```
p1=p2=-2;
surfacetransmit(&kap1,&kap2,&p1,&p2,2,4,0.1);
```

This returns `p1=0.306` and `p2=0.216`, where the different values arise from the different diffusion coefficients.

Enter either κ value as -1 to show that it equals ∞. If both κ values are -1, then the probability for the side with the slower diffusion is set to 1 and the other probability is set to the appropriate value to produce equal concentrations on both sides. For example, if side 1 has slower diffusion, then $P_1 = 1$ and $P_2 = \sqrt{D_1/D_2}$. See my 2023 paper. Output probabilities are capped so that they are always between 0 and 1, inclusive. During this capping, the other computed parameter is also modified in order to achieve the correct equilibrium concentrations according to the equation

$$\frac{P_1}{P_2} = \frac{\kappa_1}{\kappa_2}\sqrt{\frac{D_2}{D_1}}.$$

This function returns 0 for correct operation, 1 for illegal input parameters (negative or zero diffusion coefficients or time step), or 2 if there are more than 2 unknowns. These errors are easily avoided with good code, so there's no need to check for them in most cases.

Internally, this function is divided into four sections: (1) κ_1 and κ_2 are known and P_1 and P_2 are computed, (2) P_1 and P_2 are known and κ_1 and κ_2 are determined, (3) P_1 and κ_2 are known and κ_1 and P_2 are determined, and (4) κ_1 and P_2 are known and P_1 and κ_2 are determined.

Section 1 considers several special cases first, such as the κ values being equal to 0 or ∞. Then, if those don't apply, it computes P_1 and P_2 using

$$P_1 = \frac{\kappa_1\sqrt{\pi\Delta t}}{c^2\sqrt{D_1}}\left(-1 + \frac{2c}{\sqrt{\pi}} + e^{c^2}\operatorname{erfc} c\right)$$

$$P_2 = \frac{\kappa_2\sqrt{\pi\Delta t}}{c^2\sqrt{D_2}}\left(-1 + \frac{2c}{\sqrt{\pi}} + e^{c^2}\operatorname{erfc} c\right)$$

where

$$c = \sqrt{\Delta t}\left(\frac{\kappa_1}{\sqrt{D_1}} + \frac{\kappa_2}{\sqrt{D_2}}\right).$$

Section 2 considers the special case of both probabilities equaling 0 and then, if that doesn't apply, it searches for the κ_1 and κ_2 values that yields the desired P_1 and P_2 values. This search uses a greedy random walk algorithm in which `diff` is equal to the squared distance value and is minimized. In each random step, trial κ_1 and κ_2 values are found by multiplying the best current κ_1 and κ_2 values by random values that range from 1−`deltakap` to 1+`deltakap`, where `deltakap` starts at 0.5 and generally decreases over time. Successful steps are kept, and `deltakap` is increased by 10%, and unsuccessful steps are discarded, and `deltakap` is reduced by 1%. I haven't optimized these search parameters, but they seem to work remarkably well, yielding good fits within about 20 successful steps. This section of the code turns off probability capping, by setting the static variable `capprob` to 0, in order to have a meaningful distance metric even when probability values exceed 1.

Section 3 considers a known P_1 and κ_2. After considering some special cases, it searches over κ_1 values that yield the correct P_1; it doesn't need to search over κ_2 because that's already known. This is only a 1D search, so it could be done with bracketing and subdividing.

However, it was easier to use essentially the same greedy random walk method as in section 2, but now in only 1D, so that's what I did.

Section 4 addresses the same problem as section 3 but with reversed knowns and unknowns. It works by reversing parameters and calling section 3.

5 Parameter calculation functions

The following functions perform the necessary table look-up for those in the preceding section. The `interpolate...` functions work on any tabular data and the lookup functions work with pre-computed tables of data that relate interaction coefficients and simulation probabilities. Reduced units are used here.

`double interpolate1D(double *xdata, double *ydata, int n, double x);`

Performs one-dimensional interpolation or extrapolation on tabulated data using an interpolating polynomial. Enter the tabulated x-values in `xdata`, the tabulated y-values in `ydata`, and the sizes of these vectors as `n`. This function requires `n` to be at least 4 and will return -1 if this is not the case. Enter the desired x value in `x`. This finds the four data points that are closest to x, calculates the interpolating polynomial for them, and then calculates and returns the y value that corresponds to x. The xdata vector does not need to be uniformly spaced, although it must increase monotonically.

`double interpolate2D(double *xdata, double *ydata, double *zdata, int nx, int ny, double x, double y);`

Performs two-dimensional interpolation or extrapolation on tabulated data using an interpolating polynomial. The two independent variables are listed in the vectors `xdata` and `ydata`, which have lengths `nx` and `ny`, respectively. The dependent variable is the table `zdata`, which has `nx` rows and `ny` columns (i.e. y is the fast-changing index). `zdata` needs to be a single array, such that `zdata[i*ny+j]` is the element at row `i` and column `j`. Enter the desired coordinate in `x` and `y`. This finds the 4x4 grid of `xdata` and `ydata` points that surround the desired (x,y) coordinate and interpolates on both axes to estimate the corresponding z value. If either `nx` or `ny` is less than 4, this returns the error code of -1.

Internally, this first calculates the polynomial coefficients for x (columns). Then, it uses these to interpolate a z value for each of the four nearest tabulated y values (i.e. for each row). Then, it calculates the polynomial coefficients for y and combines them with the 4 interpolated z values to yield a final interpolated z value, which is returned.

`double lookupirrevadsorb(double value,int pfromk);`

This uses a look-up table to find either the reduced adsorption coefficient from an adsorption probability, or the adsorption probability from the reduced adsorption coefficient. For the former behavior, enter `value` as the probability and set `pfromk` to 0 and for the latter, enter `value` as the reduced adsorption coefficient and set `pfromk` to 1.

A Smoldyn-like algorithm is assumed in which molecules diffuse and then can be adsorbed based on a fixed probability; only molecules that end up across the surface can be adsorbed, meaning that the Andrews-Bray correction is not implemented. The reduced adsorption coefficient, κ', is related to the actual adsorption coefficient with

$$\kappa' = \frac{\kappa \Delta t}{s} \qquad\qquad \kappa = \frac{\kappa' s}{\Delta t}$$

κ is the actual adsorption coefficient, Δt is the time step, and s is the rms step length of the adsorbant.

`double lookuprevadsorbnd(double probon,double proboff);`

This function uses a look-up table to find the equilibrium surface concentration that corresponds to the adsorption and desorption probabilities `probon` and `proboff`, respectively, and no initial separation for desorbed molecules. A Smoldyn-like algorithm is assumed in which molecules diffuse and then can be adsorbed or desorbed based on fixed probabilities; only molecules that end up across the surface can be adsorbed, meaning that the Andrews-Bray correction is not implemented. A molecule cannot both adsorb and desorb during the same time step. The returned surface concentration is the reduced value C'_{srf}, which is related to the actual surface concentration C_{srf}, the solution-phase concentration C_{soln}, and the rms step length of the molecules s, with

$$C'_{srf} = \frac{C_{srf}}{C_{soln} s}$$

`double lookuprevads(double value1,double value2,int pfromk,double *ans2ptr);`

This function uses look-up tables to convert reduced adsorption and desorption coefficients to probabilities or vice versa. To go from coefficients to probabilities, set `pfromk` to 1, enter the reduced adsorption coefficient (κ') in `value1`, and enter the reduced desorption rate (k') in `value2`. The adsorption probability will be returned directly and the desorption probability will be returned with `ans2ptr` if `ans2ptr` is not `NULL`. To go from probabilities to coefficients, enter `pfromk` as 0, enter the adsorption probability in `value1`, and enter the desorption probability in `value2`. The reduced adsorption coefficient will be returned directly and the reduced desorption rate will be returned with `ans2ptr` if `ans2ptr` is not `NULL`. Note that this function is designed for reversible adsorption at equilibrium and thus is not the correct function for irreversible adsorption or desorption, which do not attain equilibrium but only attain steady-state.

`double lookuprevtrans(double pf,double pb,double *kbptr);`

This finds the reduced partial transmission coefficients (κ'_F and κ'_B) that correspond to the front and back side partial transmission probabilities `pf` and `pb` (P_F and P_B), respectively, for reversible transmission. It returns the front side coefficient directly and the back side coefficient with `kbptr` if `kbptr` is not `NULL`.

6 Functions for investigating a partially adsorbing surface

The following functions are for determining the proper adsorption probability for a molecule that collides with a partially adsorbing surface.

While the x vector does not need to be uniformly spaced, and in fact is best with dense spacing near 0 and sparser spacing with increasing distance from 0, there are nevertheless some constraints. It must be symmetric about 0, but not include the value 0, meaning that it includes the values $-\Delta x/2$ and $\Delta x/2$. Symmetry is required for reflection to work properly. Also, the x vector must extend at least as far in the positive direction as it does in the negative direction.

Reduced units are used for most functions here: lengths are divided by the rms step length, $s = \sqrt{2D\Delta t}$; time is divided by the simulation time step, Δt; and concentrations are divided by the concentration far from the surface, C_∞. With these reduced units, the reduced adsorption coefficient, κ', is related to the regular adsorption coefficient with $\kappa' = \kappa\Delta t/s$. The reduced desorption rate constant, k', is related to the regular desorption rate constant with $k' = k\Delta t$.

`void xdfdiffuse(double *x,double *xdfa,double *xdfd,int n);`

This integrates the product of the x-distribution function, in `xdfa`, and the Green's function for simple diffusion (shown below) to implement diffusion over a fixed time step. `x` is a vector of distances, `xdfa` is the input `xdf`, `xdfd` is the output xdf, and `n` is the number of points in the vectors. The equation for the integral is:

$$d(x) = \int_{-\infty}^{\infty} a(x')\,\mathrm{grn}(x, x')dx$$

$$\mathrm{grn}(x, x') = G_s(x - x')$$

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

This function does not use analytical extensions to the integral for x values that are outside of the tabulated range. This is because Mathematica cannot integrate the product of an error function and the Green's function. Thus, for $x <$ `x[0]`, it is assumed that $a(x) = 0$, which results in an integrated area of 0. For $x >$ `x[n-1]`, it is assumed that $a(x) = 1$ (i.e. the normalized concentration value). That leads to the large-x integral result

$$\int_{x_{n-1}}^{\infty} 1 \cdot \mathrm{grn}(x, x')dx' = \frac{1}{2}\left(1 + \mathrm{erf}\,\frac{x - x_{n-1}}{s\sqrt{2}}\right)$$

`double xdfadsorb(double *x,double *xdf,int n,double probon);`

Adsorbs and reflects concentration for a one-dimensional system, assuming the rms step length is 1 and the surface is at $x = 0$. `x` is the list of position values, `xdf` is the list of concentrations (both input and output), `n` is the length of both vectors, and `probon` is the adsorption probability. Reflection and adsorption are carried out on all negative position values up to and including 0. Ideally, the x vector should not include a 0 value, but should have a value that is slightly less than 0 and another that is slightly greater than 0. Integration of the xdf function is performed with the trapezoid rule, which is simple because it is one-dimensional. At a distance value of 0, the upper left triangle of the trapezoid is measured and removed, while the lower right triangle is untouched, which ensures that an adsorption that follows a prior adsorption will yield a value of 0. The integration is extrapolated to negative infinity by assuming that the left end is an error function. The return value is the total amount of stuff that crossed $x = 0$, times the value `probon`; this is the amount that should be adsorbed by the surface. The rest of the flux over $x = 0$ is reflected by this function.

The integration from $-\infty$ to x_0 uses equation 2.14 in Crank, which is rewritten for initial concentration of C_0 for $x > 0$ and 0 for $x < 0$, and with an rms step length of s.

$$C(x, \Delta t) = \frac{C_0}{2}\left(1 + \mathrm{erf}\,\frac{x}{s\sqrt{2}}\right)$$

The integral from $-\infty$ to x_0 is

$$\int_{-\infty}^{x_0} C(x,\Delta t)dx = \frac{C_0}{2}\left[s\sqrt{\frac{2}{\pi}}e^{-\frac{x_0^2}{2s^2}} + x_0\left(1+\text{erf}\,\frac{x_0}{s\sqrt{2}}\right)\right]$$

`void xdfdesorb(double *x,double *xdf,int n,double b,double flux);`

This performs desorption to a fixed distance. `x` is the vector of input x values, `xdf` is the input and output xdf, `b` is the position to which the stuff will be desorbed, and `flux` is the amount of desorbed stuff to be added to the xdf. The desorbed stuff is added with a Gaussian distribution, centered at position `b`, for 1 time step after the desorption step, assuming an rms step length of 1. This desorbs to both sides of any surface at $x = 0$, without addressing any potential reflection considerations.

`void xdfdesorbdelta(double *x,double *xdf,int n,double b,double flux);`

This adds amount `flux` to the xdf `xdf` to create a delta function near x-position `b`. As usual, `x` is the vector of input x-values and `n` is the length of the x and xdf vectors. The delta function is not added exactly at `b`, but at the next higher x-value that is in the `x` vector. Also, because of the discrete nature of the `x` vector, the delta function is best represented by a triangle, and not by a pure delta function. This triangle has the correct area, regardless of the spacing of x values. If there is desorption and then diffusion, it is better to use `xdfdesorb`, but if there is only desorption with no subsequent diffusion, then `xdfdesorbdelta` will suffice.

`double xdfsteadystate(double *x, double *xdfa, double *xdfd,int n, double cs, double b, double probon, double proboff, double eps);`

This function is used to investigate either reversible or irreversible adsorption, depending on the desorption probability listed in `proboff` (set it to 0 for irreversible and non-zero for reversible). For reversible adsorption, it returns the steady-state surface concentration probability and for irreversible adsorption, it returns the steady-state amount of flux that is adsorbed over one time step. Either way, `probon` is the adsorption probability, `x` is a vector of x position values, `xdfa` and `xdfd` are xdf vectors, and `n` is the number of elements in these vectors. Send in both `xdfa` and `xdfd` with the same values, of which two especially useful starting points are (i) every value equal to 0, or (ii) a step function that equals 0 for $x < 0$ and 1 for $x > 1$. For reversible adsorption, `cs` is the input guess for the surface concentration, and `b` is the fixed desorption distance.

The function runs until the change in net flux from one step to the next is less than `eps` (0.001 is a reasonable number) or until problems are detected; if there are problems, -1 is returned. The returned xdf vectors are the xdf at steady-state, which are identical except that `xdfa` does not include the final desorption into the xdf, while xdfd does include it, using a delta function desorption at, or at least close to, `b`.

To improve desorption accuracy, it is performed after the next diffusion occurs, with an already diffused desorbed delta function (which is a Gaussian). In Smoldyn, the sequence of operations is: diffuse, adsorb and desorb, increment the simulation time, and report conditions. This function is identical, except that the desorbed molecules are not actually placed in the xdf vectors until after the diffusion step. The only way in which the behavior here differs from that in Smoldyn is that a desorption flux is added during the first time step that represents desorbed molecules from the previous time step; the purpose of this is that it means that two sequential calls to this function is identical to one call with more iterations.

`void xdfmaketableirrev(void);`

This makes a table of adsorption coefficients as a function of the adsorption probability for steady-state irreversible adsorption. This asks the user for the number of points to use in the concentration data (the xdf), the epsilon value, the total domain of the xdf, the domain of the xdf that should be fit with a straight line from which the effective surface concentration is extrapolated, and for whether output should be formatted in a list or in machine-readable format, which is a comma-separated list. In the latter case, the list is just the adsorption coefficients. This function calculates each value twice: one from an initial xdf that is a step function that steps from 0 to 1 at $x = 0$, and the other time from an initial xdf that is all zero. These xdfs asymptotically approach the result from opposite directions so as to bracket the actual result and give a measure of the maximum error.

`void xdfmaketable();`

Makes a table of steady-state surface concentrations as functions of the desorption probability (columns, the fast-changing index) and the adsorption probability (rows). This table has about 20 rows and 20 columns. It asks the user for the number of points to use in the concentration data (the xdf), the epsilon value, which determines the stopping point of the calculations, and for whether output should be formatted in a list or in machine-readable format, which is a comma-separated table.