1 Preface

Recent studies on tunable nano reactors with termosensitive polymer shell have shown curious effects in reaction rates. The state of the shell is presumalbly fluctuating between states with different permeability for the substrate. To investigate on this effect a simplified system containing a spherical sink shielded by a potential barrier is investigated. We derive an implicit solution for the diffusion controlled reaction rate and verify these results with brownian dynamics simulations.

This report contains the necessary spadework namely derivation of analytical solutions for non fluctuating potentials as well as development and testing of a numeric Brownian dynamics simulation of the system.

Contents

2 Analytic Considerations

This part will introduce the basic equations that are relevant for the handling of Brownian motion as a stochastic process (and its realisations in a computational model) as well as the derivation of an equivalent Focker-Planck equation to derive an analytic expression for the wanted density profiles and reaction rates.

2.1 The Focker Planck Equation for Brownian Particles

Brownian motion is a markovian process, i.e. each time step in the random motion of particles does only depend on their preceding position. This implies, that the conditional distribution of their coordinates obeys the following relation:

$$P(x,t|y,u;y,v) = P(x,t|y,v), \quad t > u > v \tag{1}$$

This relation implies, that for a Markov process every multi step probability distribution can be expressed as a hierarchy of a initial distribution and the two step transition probabilities. For $t_1 < t_2 < \cdots < t_n$:

$$\begin{aligned} P(x_1,t_1;x_2,t_2;\cdots;x_n,t_n) = P(x_n,t_n|x_{n-1},t_{n-1})P(x_{n-1},t_{n-1}|x_{n-2},t_{n-2})\cdots \\ \cdots P(x_2,t_2|x_1,t_1)P(x_1,t_1) \end{aligned} \tag{2}$$

So the entire realization of the process is determined by the initial distribution and the two step transition probability.
Integrating the three step joint probability distribution over the intermediate step leads to the Chapman Kolmogorov equation:

$$P(x,t|y,v) = \int P(x,t|z,u)P(z,u|y,v). \tag{3}$$

From this one can derive the Kramers Moyal expansion for $P(x,t)$:

$$\frac{\partial P(x,t)}{\partial t} = \sum_{m=1}^{\infty} \frac{(-1)^m}{m!} \frac{\partial^m}{\partial x^m} \left[a^{(m)}(x,t)P(x,t)\right] \tag{4}$$

with the *jump moments* of the transition probability $W(x,\Delta x,t,\Delta t) = P(x+\Delta x, t+\Delta t|x,t)$:

$$a^{(m)}(x,t) = \int \mathrm{d}W(x,r,t,\Delta t) r^m. \tag{5}$$

If the expansion is truncated after the second term, the result gives the well known Focker Planck Equation:

$$\frac{\partial P(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left[a^{(1)}P(x,t)\right] + \frac{\partial^2}{\partial x^2}\left[a^{(2)}P(x,t)\right] \tag{6}$$

These *jump moments* can be calculated from the Langewin equation, describing the Brownian motion of a Particle in solution:

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\gamma\frac{\mathrm{d}x}{\mathrm{d}t} + f(x) + \varepsilon(t) \tag{7}$$

in which $\varepsilon(t)$ is a Gaussian distributed random process describing the collision interaction of the particle and the solute. In the overdamped limit this expression can be discretized in time and transforms to:

$$x(t+\Delta t) = x(t) + \frac{1}{\gamma}f(x,t)\Delta t + \frac{1}{\gamma}\varepsilon'(t)\Delta t. \tag{8}$$

From the distribution of the random force:

$$P(\varepsilon') = \sqrt{\frac{\Delta t}{4\pi D\gamma^2}}\exp\left\{\frac{\gamma^2\Delta t}{4D\gamma^2}\right\} \tag{9}$$

one can compute the transitions probability for the Brownian particle as:

$$W(x,\Delta x,t,\Delta t) = \langle\delta\left(\Delta x - (x(t-\Delta t) - x(t))\right)\rangle \tag{10}$$

$$= \int \mathrm{d}\varepsilon'\delta\left(\Delta\mathrm{x} - (\mathrm{x}(\mathrm{t}-\Delta\mathrm{t}) - \mathrm{x}(\mathrm{t}))\right)\sqrt{\frac{\Delta\mathrm{t}}{4\pi\mathrm{D}\gamma^2}}\exp\left[\frac{\gamma^2\Delta\mathrm{t}}{4\mathrm{D}\gamma^2}\right] \tag{11}$$

$$= \sqrt{\frac{1}{4\pi D\Delta t}}\exp\left[\frac{-\left(\Delta x - f(x)\frac{\Delta t}{\gamma}\right)^2}{4D\Delta t}\right] \tag{12}$$

For this Gaussian transition probability the coefficients of the Kramers Moyal Expansion vanish after the second term, such that the resulting Focker Planck equation holds the full analytic solution for the time evolution of the distribution of particles.

$$\frac{\partial P(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left[f(x)P(x,t)\right] + D\frac{\partial^2}{\partial x^2}\left[P(x,t)\right] \tag{13}$$

2.2 Brownian Particles diffusing around a spherical Sink

The problem that shall be approached implies a spherical sink of radius R_s at the origin, that absorbes every particle, that crosses its boarder. Further the particle density at infinity is assumed to be constant, isotrop and homogeneous. The density for $t_o = 0$ is constant for all $r > R_s$. This leads to the following conditions:

$$\rho(r > R_s, t = 0) = \rho_o, \tag{14}$$

$$\rho(r = R_s, t) = 0, \tag{15}$$

$$\lim_{r\to\infty}\rho(r,t) = \rho_o. \tag{16}$$

In the following, the given Focker Planck Equation in terms of particle densities:

$$\frac{\partial \rho(x,t)}{\partial t} = -\vec{\nabla}\left[\vec{f}(x)\rho(x,t)\right] + D\vec{\nabla}^2\left[\rho(x,t)\right] \tag{17}$$

will be solved without external force, i.e. $\vec{f}(r) = 0$ and subject to the given boundary and initial conditions. With the substitution $r \cdot \rho(r,t) = u(r,t)$ and the assumption, that the problem is spherically symmetric the derivatives in the Focker Planck equation simplify to

$$\frac{\partial u(r,t)}{\partial t} = D\frac{\partial^2 u(r,t)}{\partial r^2} \tag{18}$$

Laplace transform of the equation yields:

$$\int_0^\infty e^{-st}\frac{\partial u(r,t)}{\partial t}\mathrm{dt} = D\frac{\partial^2}{\partial r^2}\int_0^\infty e^{-st}u(r,t)\mathrm{dt} \tag{19}$$

$$\left[e^{-st}u(r,t)\right]_0^\infty + s\int_0^\infty e^{-st}u(r,t)\mathrm{dt} = D\frac{\partial^2}{\partial r^2}\tilde{u}(r,s) \tag{20}$$

$$u(r,0) + s\tilde{u}(r,s) = D\frac{\mathrm{d}^2}{\mathrm{d}r^2}\tilde{u}(r,s). \tag{21}$$

This is an ordinary 2nd degree inhomogeneous differential equation with constant coefficients. For the standard ansatz $\tilde{u}(r,s) = \exp(\lambda(s)r)$ for the homogeneous solution we get the following characteristic polynomial:

$$\lambda(s)^2 - \frac{s}{D} = 0 \tag{22}$$

resulting in the following homogeneous solution:

$$\tilde{u}_h(r,s) = C_1 e^{-\sqrt{\frac{s}{D}}\cdot r} + C_2 e^{\sqrt{\frac{s}{D}}\cdot r} \tag{23}$$

We find the inhomogeneous solution using a polynomial ansatz of the form $\tilde{u}_i = C_3 r + C_4$ leading to the following relation:

$$s(C_3 r + C_4) = -u(r,0) \tag{24}$$

$$= -r\rho_o \tag{25}$$

$$\Rightarrow C_3 = \frac{r}{s}\rho_o \tag{26}$$

$$C_4 = 0 \tag{27}$$

Now the entire solution has to be fitted to the boundary conditions as in (16). The solution in Laplace space then reads:

$$\tilde{u}(r,s) = \rho_o\left(\frac{r}{s} + \frac{R_s}{s}e^{\sqrt{\frac{s}{D}}(R_s - r)}\right) \tag{28}$$

The inverse Laplace transform

$$u(r,t) = \frac{1}{2\pi i} \int\limits_{\gamma - i\infty}^{\gamma + i\infty} e^{st} \tilde{u}(r,s) \mathrm{d}t \tag{29}$$

$$= \frac{\rho_o}{2\pi i} \left\{ \int\limits_{\gamma - i\infty}^{\gamma + i\infty} \frac{r}{s} \mathrm{d}t + \int\limits_{\gamma - i\infty}^{\gamma + i\infty} \frac{R_s}{s} e^{\sqrt{\frac{s}{D}}(R_s - r)} \mathrm{d}t \right\} \tag{30}$$

is done using the residue theorem for the first integral:

$$\oint_\gamma \mathrm{d}z f(z) = 2\pi i \sum_{k=1}^{n} I(\gamma, a_k) \mathrm{Res}(f, a_k) \tag{31}$$

$$\mathrm{Res}(f, y_o) = \frac{1}{(m-1)!} \lim_{z \to z_o} \frac{\mathrm{d}^{m-1}}{\mathrm{d}z^{m-1}} \left[(z - z_o)^m f(z) \right] \tag{32}$$

and the following identity for the second:

$$\mathcal{L}\left[\mathrm{erfc}\left(\frac{\mathrm{a}}{2\sqrt{\mathrm{t}}} \right) \right] = \frac{1}{s} e^{a\sqrt{s}} \tag{33}$$

resulting in the following time dependent solution for $u(r,t)$ resp. the particle density $\rho(r,t)$:

$$u(r,t) = \rho_o \left\{ r - R_s \mathrm{erfc}\left(\frac{r - R_s}{\sqrt{4Dt}} \right) \right\} \tag{34}$$

$$\rho(r,t) = \rho_o \left\{ 1 - \frac{R_s}{r} + \mathrm{erf}\left(\frac{r - R_s}{\sqrt{4Dt}} \right) \right\}. \tag{35}$$

In the limit $t \leftarrow \infty$ this results in the steady state density profile:

$$\rho(r) = \rho_o \left(1 - \frac{R_s}{r} \right) \tag{36}$$

The reaction rate can be defined as the total flux of particles through the boundary Ω of the sink:

$$K = \int_\Omega \vec{J} \mathrm{d}\vec{A} \tag{37}$$

Using the differential continuity equation:

$$\frac{\partial \rho(\vec{r},t)}{\partial t} = \vec{\nabla} \vec{J}(\vec{r},t) \tag{38}$$

$$= \vec{\nabla} \left\{ \rho(\vec{r},t) \nabla \vec{U}(\vec{r}) + D \vec{\nabla} \rho(\vec{r},t) \right\} \tag{39}$$

and the spherical symmetry of the solution one can derive the time dependent reaction rate of the Brownian particles with the spherical sink of radius R_s as follows:

$$K(t) = \int_\Omega D\vec{\nabla}\rho(\vec{r},t) \tag{40}$$

$$= 4\pi D R_s^2\, \vec{\nabla}\rho(\vec{r},t)\Big|_{r=R_s} \tag{41}$$

$$= 4\pi D R_s \rho_o \left(1 + \frac{R_s}{\sqrt{4Dt}}\right) \tag{42}$$

Again in the limit of $t \to \infty$ this results in the steady state absorption rate:

$$K = 4\pi D R_s \rho_o \tag{43}$$

2.3 Steady State solution for Ideal Sink with Potential Barrier

The next section approaches the problem of Brownian particles diffusing around an ideal sink with a spherically symmetric potential barrier. The barrier is assumably smooth i.e. continuous and continuously differentiable. The boundary conditions are as in (16). The Focker Planck Equation thus is the following:

$$0 = \vec{\nabla}\left(\frac{1}{\gamma}\rho(\vec{r})\vec{\nabla}U(\vec{r}) + D\vec{\nabla}\rho(\vec{r})\right) \tag{44}$$

where γ is the friction constant. The solution for this problem was originally derived by Debye in 1949. From the spherical symmetry of the system, the Einstein Smoluchowski relation for friction and diffusion constant and the Gauss integration theorem follows:

$$\frac{K}{4\pi D r^2} = \rho(r)\frac{\mathrm{d}}{\mathrm{d}r}\frac{U(r)}{K_B T} + \frac{\mathrm{d}}{\mathrm{d}r}\rho(r) \tag{45}$$

where K denotes the total flux through the sink boundary that is by definition equal to the desired rate constant. The homogeneous Solution can be found by an exponential ansatz:

$$\rho_h(r) = Ce^{-\frac{U(r)}{K_B T}}. \tag{46}$$

The particular solution can be found via variation of the integration constant:

$$C(r) = C(R_s) + \frac{K}{4\pi D}\int_{R_s}^{r} \exp\left(\frac{U(r')}{K_B T}\right)\frac{r'^2}{\mathrm{d}}r. \tag{47}$$

To guarantee $\rho(R_s) = 0$, $C(R_s)$ has to be zero and for the solution holds:

$$\rho(r) = \frac{K}{4\pi D}\exp\left(-\frac{U(r)}{K_B T}\right)\int_{R_s}^{r} \exp\left(\frac{U(r')}{K_B T}\right)\frac{\mathrm{d}r'}{r'^2} \tag{48}$$

The rate constant K is obtained by normalization of the solution:

$$K = 4\pi D\left\{\int_{R_s}^{r} \exp\left(\frac{U(r')}{K_B T}\right)\frac{\mathrm{d}r'}{r'^2}\right\}^{-1} \tag{49}$$

2.4 Debye Solution for Boxcar Potential

The Debye solution for a spherical sink with a potential barrier only holds for smooth potential functions. In case of an unsteady potential the solution has to be obtained differently. The following section will consider the special case of a boxcar like potential barrier.

$$U(r) = \begin{cases} 0 & : R_s < r \leq a \\ U_o & : a < r \leq b \\ 0 & : b < r \leq R_d \end{cases} \tag{50}$$

To find a solution of the Focker Planck equation for this Potential, we integrate over a small area containing the saltus of the Potential.

$$0 = \vec{\nabla} \left\{ \frac{1}{\gamma} \rho(\vec{r}) \vec{\nabla} U(\vec{r}) + D \vec{\nabla} \rho(\vec{r}) \right\} \tag{51}$$

$$J(r) = \frac{1}{\gamma} \rho(\vec{r}) \vec{\nabla} U(\vec{r}) + D \vec{\nabla} \rho(\vec{r}) \tag{52}$$

$$\int_{a-\varepsilon}^{a+\varepsilon} \mathrm{d}r \frac{J(r)}{D\rho(r)} = \int_{a-\varepsilon}^{a+\varepsilon} \mathrm{d}r \delta(r-a) \frac{U_o}{K_B T} + \frac{1}{\rho(r)} \nabla \rho(r) \tag{53}$$

$$-\frac{U_o}{K_B T} = \lim_{\varepsilon \to 0} \left[\ln(\rho(r)) \right]_{a-\varepsilon}^{a+\varepsilon} - \lim_{\varepsilon \to 0} \int_{a-\varepsilon}^{a+\varepsilon} \mathrm{d}r \frac{J(r)}{D\rho(r)} \tag{54}$$

$$\rho_+(a) = \rho_-(a) e^{\frac{U_o}{K_B T}} \tag{55}$$

This way we obtain boundary conditions to fit the known solution for the force free Focker Planck equation (36) at the jump discontinuity.
We solve the emerging system of linear equations and obtain the following solution for steady state density profile and absorption rate:

$$\rho(r) = \rho_0 \cdot \begin{cases} \left(1 - \frac{R_s}{r}\right) & for \quad R_s < r \leq a \\ \left(1 - \frac{R_s}{a}\right) e^{-\frac{U_o}{K_B T}} + \frac{R_s}{a} - \frac{R_s}{r} & for \quad a < r \leq b \\ R_s \left(e^{\frac{U_0}{K_B T}} - 1\right) \left(\frac{1}{a} - \frac{1}{b}\right) + 1 - \frac{R_s}{r} & for \quad b < r \end{cases} \tag{56}$$

$$K = 4\pi R_s D \rho_0 \tag{57}$$

2.5 Treatment of fluctuating Boxcar Potential Barrier

The following section will treat the analytic solution of a boxcar potential barrier with fluctuating hight. The equations for the system are Focker Planck equations that are coupled by the transition rate matrix for the different states of the barrier.

$$\frac{\partial \boldsymbol{\rho}}{\partial t} = \nabla \boldsymbol{\rho} \nabla \hat{U} f(\vec{r}) + D \nabla^2 \boldsymbol{\rho} + \hat{W} \boldsymbol{\rho} \tag{58}$$

The Matrix $\hat{U}$ has diagonal elements U_{ii} equal to the different hight of the potential barrier in different states and is zero elsewhere. $f(\vec{r})$ describes the radial shape of the potential. For the rate matrix holds: $W_{ii} = -\sum_{i\neq j} W_{ij}$. The Matrix $\hat{W}$ is generally not symmetric. But one can show that it always has an eigenvector satisfying $\hat{W}\boldsymbol{\rho}_{eq} = 0$ and that for this eigenvector the following relation, also known as *detailed balance* is valid:

$$W_{ij}\boldsymbol{\rho}^{(eq)}[i] = W_{ji}\boldsymbol{\rho}^{(eq)}[j]. \tag{59}$$

Therefore the similarity transformation

$$\hat{T}^{-1}\hat{W}\hat{T} = \hat{S} \tag{60}$$

where

$$T_{ij} = \delta_{ij}\boldsymbol{\rho}^{(eq)}[i]^{-\frac{1}{2}} \tag{61}$$

symmetrizes $\hat{W}$ such that

$$\begin{aligned} S_{il} &= T_{ij}^{-1}W_{jk}T_{kl} \\ &= \sum_j \delta_{ij}\rho^{(eq)}[i]^{\frac{1}{2}}W_{jk}T_{kl} \\ &= \rho^{(eq)}[i]^{\frac{1}{2}}\sum_k W_{ik}\delta_{kl}\rho^{(eq)}[l]^{-\frac{1}{2}} \\ &= W_{il}^{\frac{1}{2}}\left(W_{il}\frac{\rho^{(eq)}[i]}{\rho^{(eq)}[l]}\right)^{\frac{1}{2}} \\ &= (W_{il}W_{li})^{\frac{1}{2}} \\ S_{ii} &= W_{ii} \end{aligned} \tag{62}$$

The resulting matrix can then be diagonalized by an orthogonal matrix $\hat{D}$:

$$\hat{D}^{\dagger}\hat{S}\hat{D} = -\hat{d}. \tag{63}$$

It can be shown that $d_i \leq 0$ and $d_1 = 0$ with the corresponding eigenvector:

$$D_{i1} = \rho^{(eq)}(i)^{\frac{1}{2}}. \tag{64}$$

Now we will treat the potential term in (58). Since $f(r)$ is constant except for the $r = a, r = b$ where it jumps from 0 to 1 and from 1 to 0 respectively, the barrier can be treated analogously to (55). Integration over an infinitesimally small interval containing the jump discontinuity of the potential results in fitting conditions for each component of $\boldsymbol{\rho}$:

$$\begin{aligned} \boldsymbol{\rho}^{(I)}(a) &= \mathrm{diag}[\exp(\frac{U_i}{K_BT})]\boldsymbol{\rho}^{(II)}(a), \\ \boldsymbol{\rho}'^{(I)}(a) &= \boldsymbol{\rho}'^{(II)}(a), \\ \boldsymbol{\rho}^{(III)}(b) &= \mathrm{diag}[\exp(\frac{U_i}{K_BT})]\boldsymbol{\rho}^{(II)}(b), \\ \boldsymbol{\rho}'^{(III)}(b) &= \boldsymbol{\rho}'^{(II)}(b). \end{aligned} \tag{65}$$

other boundary conditions are:

$$\begin{aligned} \boldsymbol{\rho}^{(I)}(R_s) &= 0, \\ \boldsymbol{\rho}^{(III)}(r \to \infty) &= \boldsymbol{\rho}^{(eq)}. \end{aligned} \tag{66}$$

Here I, II and III denote the different regions for I: $R_s < r \leq a$, II: $a < r \leq b$ and III: $b < r$. Now, to find a solution for the density profile, eq. (58) is transformed into new, independent coordinates via the transformations (61) and (63). Also the drift term from the potential is spared, since it is treated via fitting conditions only. For the steady state case this results in

$$0 = D\nabla^2 \tilde{\boldsymbol{\rho}} - \hat{d}\tilde{\boldsymbol{\rho}} \tag{67}$$

The solution then reads:

$$\begin{aligned} \tilde{\rho}_1^{(j)}(r) &= c_{1,1}^{(j)} + c_{1,2}^{(j)}\frac{1}{r} \\ \tilde{\rho}_{i\neq 1}^{(j)}(r) &= c_{i,1}^{j} \exp\left[-r\sqrt{\frac{d_i}{D}}\right] + c_{i,2}^{j} \exp\left[r\sqrt{\frac{d_i}{D}}\right] \end{aligned} \tag{68}$$

Now an efficient formulation for the calculation of the coefficients $c_{i,k}^{(j)}$ must be found. Therefore the fitting and boundary conditions from (65) and (66) are transformed to the new coordinates:

$$\begin{aligned} \hat{A}^{-1}\boldsymbol{\rho}^{(I)}(R_s) &= \tilde{\boldsymbol{\rho}}^{(I)}(R_s) = 0 \\ \tilde{\boldsymbol{\rho}}(r \to \infty) &= \hat{A}^{-1}\boldsymbol{\rho}^{(eq)} = (1, 0, \cdots, 0) \end{aligned} \tag{69}$$

and

$$\begin{aligned} \tilde{\boldsymbol{\rho}}^{(I)}(a) &= \hat{A}^{-1}\mathrm{diag}[\exp(\frac{U_i}{K_B T})]\hat{A}\tilde{\boldsymbol{\rho}}^{(II)}(a), \\ \tilde{\boldsymbol{\rho}}'^{(I)}(a) &= \tilde{\boldsymbol{\rho}}'^{(II)}(a), \\ \tilde{\boldsymbol{\rho}}^{(III)}(b) &= \hat{A}^{-1}\mathrm{diag}[\exp(\frac{U_i}{K_B T})]\hat{A}\tilde{\boldsymbol{\rho}}^{(II)}(b), \\ \tilde{\boldsymbol{\rho}}'^{(III)}(b) &= \tilde{\boldsymbol{\rho}}'^{(II)}(b) \end{aligned} \tag{70}$$

where $\hat{A} = \hat{D}\hat{T}$. The resulting system of linear equations is then used to determine the coefficients $c_{i,k}^{j}$. The absorption rate can still be calculated as

$$K = 4\pi R_s^2 \left.\frac{\partial}{\partial r}\right|_{R_s} \sum_i \rho_i^I \tag{71}$$

where ρ_i^I are the entries of the reverse transformed of $\tilde{\boldsymbol{\rho}}$ i.e. $\boldsymbol{\rho} = \hat{A}\tilde{\boldsymbol{\rho}} = \hat{D}\hat{T}\tilde{\boldsymbol{\rho}}$.

3 Computational Model

3.1 Equation of Motion

The implemented model to simulate the studied system is a particle simulation. In fact, this implies the simulation of a large number of realizations of the stochastic process as in (2) using the transition probability for discrete time Brownian motion as in (12).
The equation of motion for the particles thus becomes:

$$\vec{r_i}(t+\Delta t) = \vec{r_i}(t) + \sqrt{2D\Delta t}R(t) \tag{72}$$

3.2 Boundary Conditions

In addition one has to take into account the boundary and initial conditions for the system (16). These boundary conditions have to be modified to satisfy the finite domain of the simulation. For reasons of simplicity one is interested only in the steady state of the system. In this case, the continuity equation is used to obtain conditions for the domain boundaries:

$$\frac{\partial \rho(\vec{r},t)}{\partial t} = \vec{\nabla}\vec{J} = 0 \tag{73}$$

This implies:

$$\begin{aligned} 0 &= \int_V \vec{\nabla}\vec{J}\mathrm{d}V \\ &= \int_{\partial V} \vec{J}\mathrm{d}\vec{A} \\ &= 4\pi\left(R_s|\vec{J}(R_s)| - R_d|\vec{J}(R_d)|\right) \end{aligned} \tag{74}$$

To use this condition in the simulation, the domain is set to be spherical. If a particle that exits simulation domain $r_o > R_d$ is set inside the simulation domain again where the old and new radial coordinate of the particle is:

$$r_n = 2R_d - r_o \tag{75}$$

Similarly the particles that enter the sink $r_o < R_s$ are set to the boundary of the simulation domain with old an new radial coordinates as in

$$r_n = R_d - (r_o - R_s) \tag{76}$$

such that the total flux into the sink has to equal the total flux through the boundary of the simulation domain.

3.3 Initial Conditions

Since this boundary condition only holds for the steady state of the system, the initial conditions must be set accordingly. Therefore we use inverse sampling to initially distribute the particles in the system according to the following cumulative distribution function:

$$F(r) = C\int_{R_s}^{r} \mathrm{d}r' \left(1 - \frac{R_s}{r'}\right), \quad R_s \geq r \geq R_d \tag{77}$$

$$C = \int_{R_s}^{R_d} \mathrm{d}r \left(1 - \frac{R_s}{r}\right) \tag{78}$$

The absorption rate of the sink is simply calculated by counting the particles that enter the sink per time step.

3.4 Normalization in Finite Volume without Potential Barrier

To be able to compare the computational results to an analytic solution the normalization for the steady state density i.e. ρ_o has to be calculated for finite volume without potential barrier and given particle Number:

$$N = \int_{R_s}^{R_d} \mathrm{d}V \rho_o \left(1 - \frac{R_s}{r}\right) \tag{79}$$

$$\rho_o = \frac{N}{4\pi\left[\frac{1}{3}r^3 - \frac{R_s}{2}r^2\right]_{R_s}^{R_d}} \tag{80}$$

$$K = \frac{R_s D N}{\left[\frac{1}{3}r^3 - \frac{R_s}{2}r^2\right]_{R_s}^{R_d}} \tag{81}$$

3.5 Potential Barrier

The boxcar potential barrier is approximated by the following function:

$$U(r) = \frac{U_0}{\left(\frac{2}{b}(r-a)\right)^{2n} + 1} \tag{82}$$

where U_0 is the height and b is the width of the barrier and a is the distance of the middle of the barrier from the origin.
In the Limit for large n this becomes:

$$U(r) = \begin{cases} 0 & for \quad |r-a| > \frac{b}{2} \\ \frac{U_0}{2} & for \quad |r-a| = \frac{1}{2} \\ U_0 & for \quad |r-a| < \frac{b}{2} \end{cases} \tag{83}$$

3.6 Normalization in Finite Volume with Boxcar Potential Barrier

The normalization for the density profile in presence of a boxcar like potential barrier is the following:

$$N = \int_{R_s}^{R_d} \rho(r)\mathrm{d}V \tag{84}$$

$$\begin{aligned}=&4\pi\rho_o \left\{ \int_{R_s}^{a} \left(\alpha_1 - \frac{R_s}{r}\right) r^2 \mathrm{d}r \right.\\ &+ \int_{a}^{b} \left(\alpha_2 - \frac{R_s}{r}\right) r^2 \mathrm{d}r \\ &\left. + \int_{b}^{R_d} \left(\alpha_3 - \frac{R_s}{r}\right) r^2 \mathrm{d}r \right\}\end{aligned} \tag{85}$$

where the fit parameters α_i are taken from the solution derived in (57):

$$\alpha_1 = 1 \tag{86}$$

$$\alpha_2 = \left(1 - \frac{R_s}{a}\right) e^{-\frac{U_o}{K_B T}} + \frac{R_s}{a} \tag{87}$$

$$\alpha_3 = R_s \left(e^{\frac{U_0}{K_B T}} - 1\right) \left(\frac{1}{a} - \frac{1}{b}\right) + 1 \tag{88}$$

For the reaction Rate thus holds:

$$\begin{aligned} K &= 4\pi D R_s \rho_o \\ &= NDR_s \left\{ \left[\frac{\alpha_1}{3} r^3 - \frac{R_s}{2} r^2\right]_{R_s}^{a} + \left[\frac{\alpha_2}{3} r^3 - \frac{R_s}{2} r^2\right]_{a}^{b} + \left[\frac{\alpha_3}{3} r^3 - \frac{R_s}{2} r^2\right]_{b}^{R_d} \right\}^{-1} \\ &= NDR_s \left\{ \frac{\alpha_1}{3}\left(a^3 - R_s^3\right) + \frac{\alpha_2}{3}\left(b^3 - a^3\right) + \frac{\alpha_3}{3}\left(R_d^3 - b^3\right) + \frac{R_s}{2}\left(R_s^2 - R_d^2\right) \right\}^{-1} \end{aligned} \tag{89}$$

4 Results

If not stated differently the simulations use parameters as given in the table below:

N	10^5
D	$0,05$
R_s	1
R_d	10
$\mathrm{d}t$	10^{-3}

Table 1: Default simulation parameters without potential barrier

4.1 Ideal Sink without Potential Barrier

The following figures show results of simulations using the model explained before. First we examine the dependence of the reaction rate on the diffusion constant. Therefore the simulation results are compared to the steady state analytic solution as given in (81):

$$K = 4\pi R_s D \rho_o \tag{90}$$

$$\rho_o = N \left\{ \int_{R_s}^{R_d} \left(1 - \frac{R_s}{r}\right) \mathrm{d}r \right\}^{-1} \tag{91}$$

4.1.1 Varying Diffusion Coefficient D

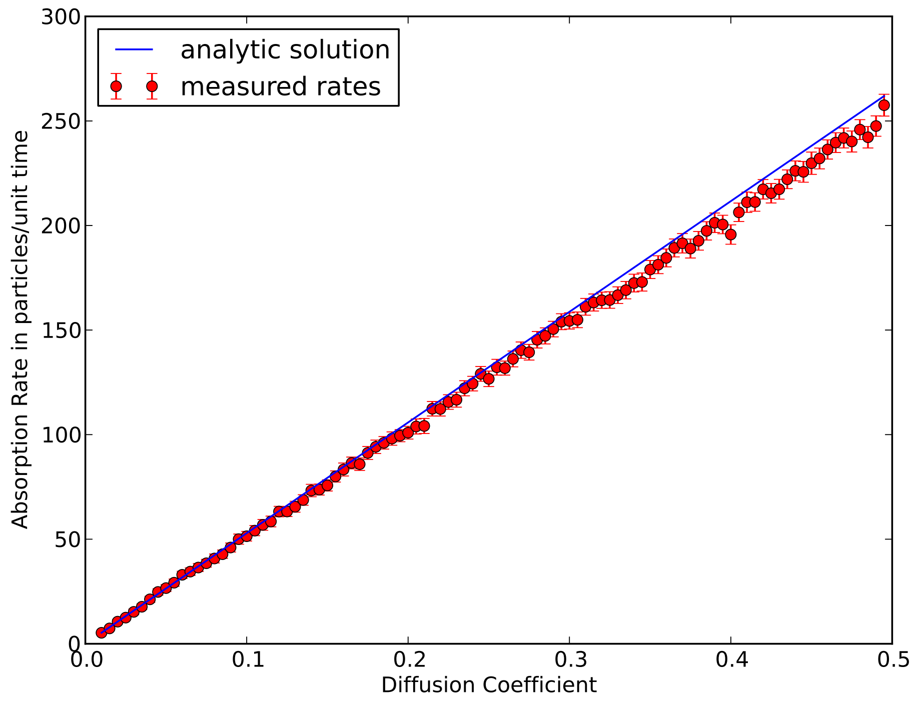

Figure 1: Reaction rates vs. Diffusion coefficient - Analytic solution and measured results

The preceding figure shows the results for several simulation runs with different diffusion coefficient. It is obvious that the simulation results show the correct linear behaviour for the reaction rate but have a systematic error of about $5-10\%$. To give a better impression of the relational dependence the following plot shows relative quantities for the results: or $D > 0.1$ the figure shows a systematic error of about 5% in the results for the reaction rates. For $D < 0.1$ the results are distributed around the analytic solution, mostly within the error bars, that are at 2σ.

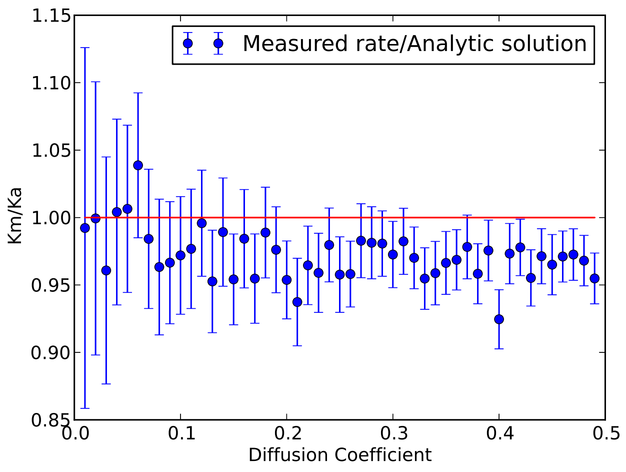

Figure 2: Relative quantities for Reaction rates

4.1.2 Varying Sink Radius R_s

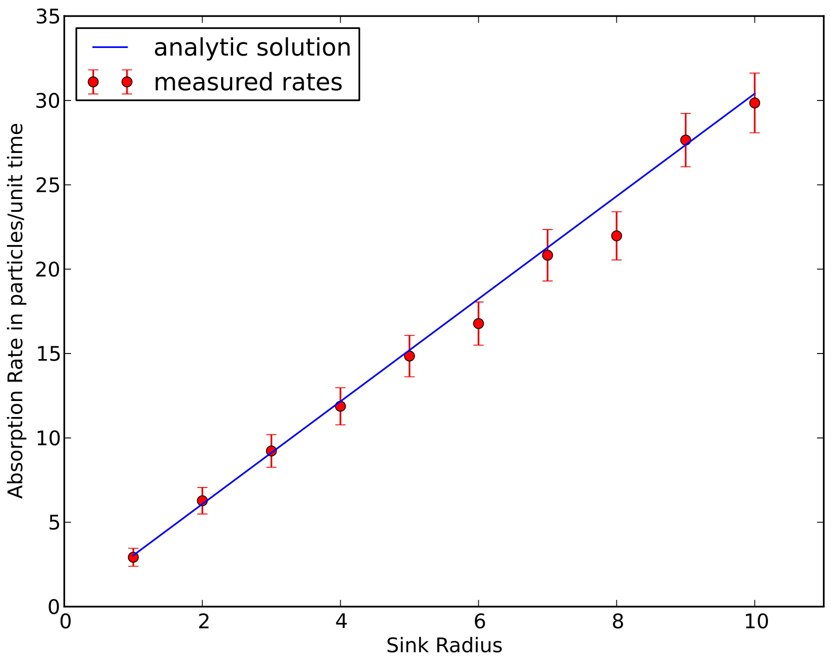

Figure 3: Reaction rate vs. Sink Radius

This section gives results for different sink radius R_s @ constant density ρ_o to test the obvious linear relationship given in (91). The results presented in the figure above do qualitatively support the linear relation between sink radius and reaction rate. The adjacent strongly suggests, that the simulation also quantitatively maps on the analytic solution as the results are correct within the calculated errors at 2σ. The following figures present the obtained density profiles from the simulation.

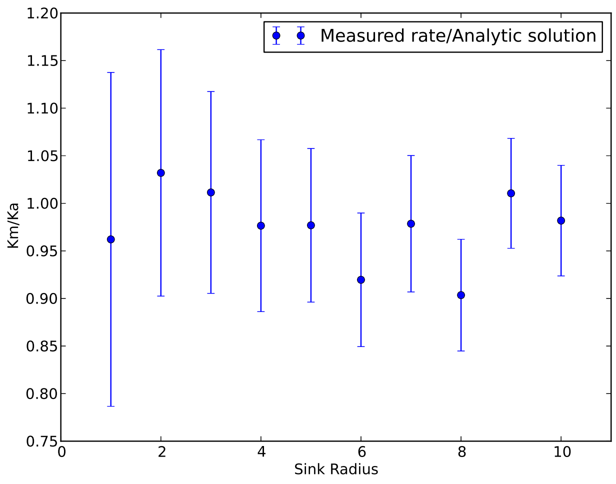

Figure 4: Normalized reaction rate vs. sink radius

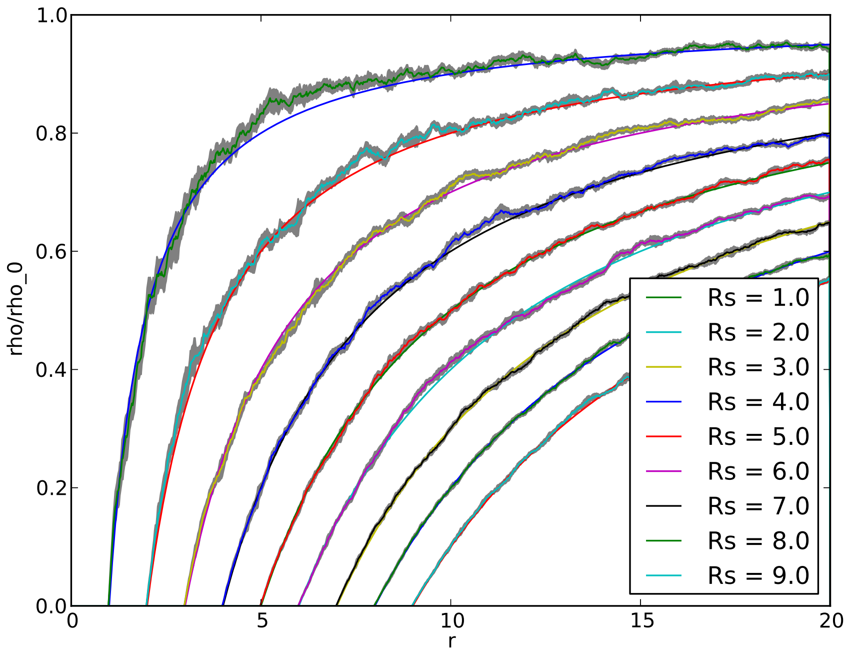

Figure 5: Normalized density profiles for different sink radius

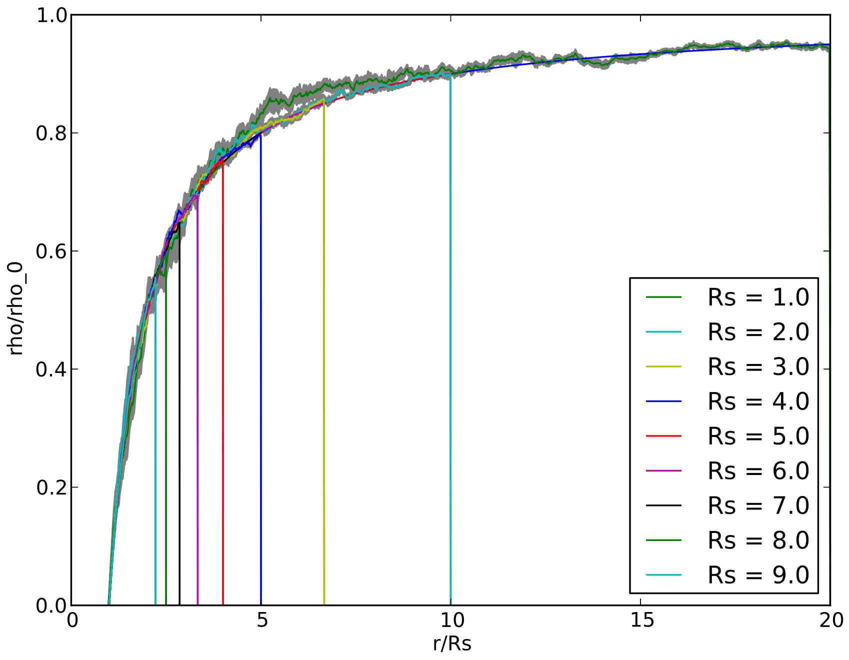

Figure 6: Normalized density profiles for different sink radius with rescaled radial coordinate

4.1.3 Finite size analysis - Varying Domain Radius R_d

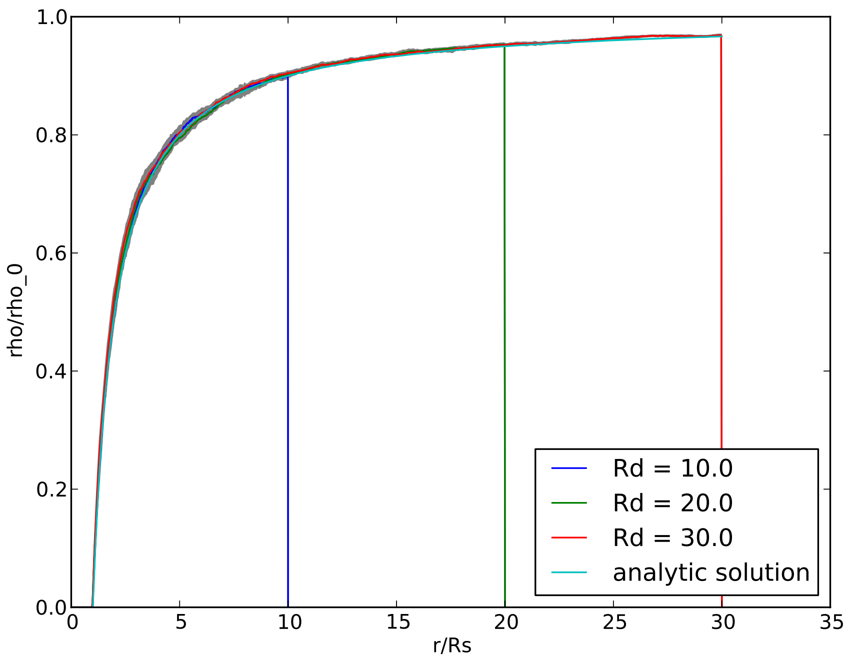

Figure 7: Density profiles for different domain Radius R_d

This section shows plots for different domain radius R_d @ constant density ρ_o. This will point out finite size effects in the simulation (if they exist). To account for the larger volume the number of particles is adjusted to keep the solution for density profile and rate constants unchanged. The plot above shows, that the variation of the domain radius does not qualitatively change the simulated density profile.

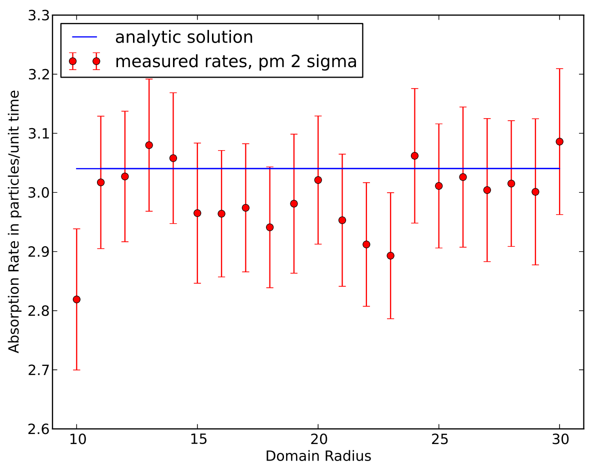

Figure 8: reaction rate vs. domain radius

The results for the rate constants does not show any qualitative changes for varying domain radius. These results suggest, that for the given boundary conditions and for ideal particles without external forces there are no finite size effects in this simulation.

4.2 Subsumption

The different results from simulation presented in the previous section show that:

- The density profiles from the BD simulation fit the analytical solution,
- the obtained rate constants are as expected within the calculated errors
- the diffusion constant and or time step has to be small enough for the particle trajectories to resolve the boundary conditions properly,
- finite size effects can be neglected.

4.3 Ideal Sink with Boxcar Potential Barrier

The following section contains simulation results for a potential barrier as derived in (57). If not stated differently, the following parameters are used:

N	10^4
D	$0,05$
R_s	1
R_d	10
U_0	2
U_a	2
U_b	2
U_n	50
$\mathrm{d}t$	10^{-4}

Table 2: Default simulation parameters for simulation with boxcar potential

4.3.1 Varying Distance from Sink to Barrier (U_a)

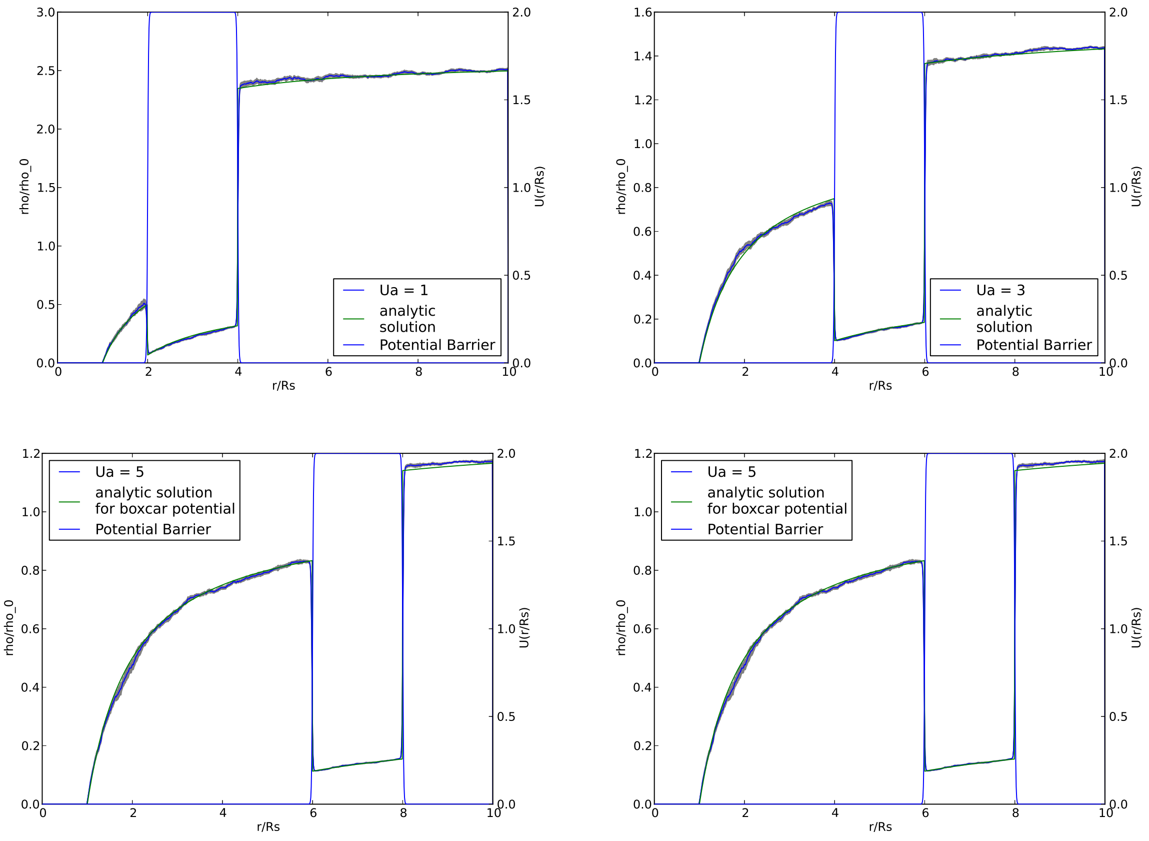

Figure 9: Density Profile for varying U_a

The simulation results obviously fit the analytic solution. Same holds for the calculated reaction rates as presented in the following Plot:

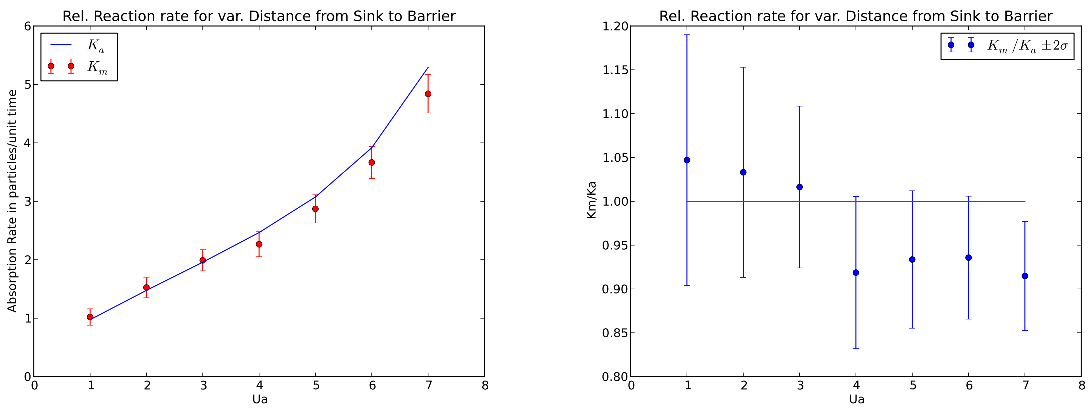

Figure 10: Absolute and relative Absorption rate for varying U_a

4.3.2 Varying Barrier Height (U_0)

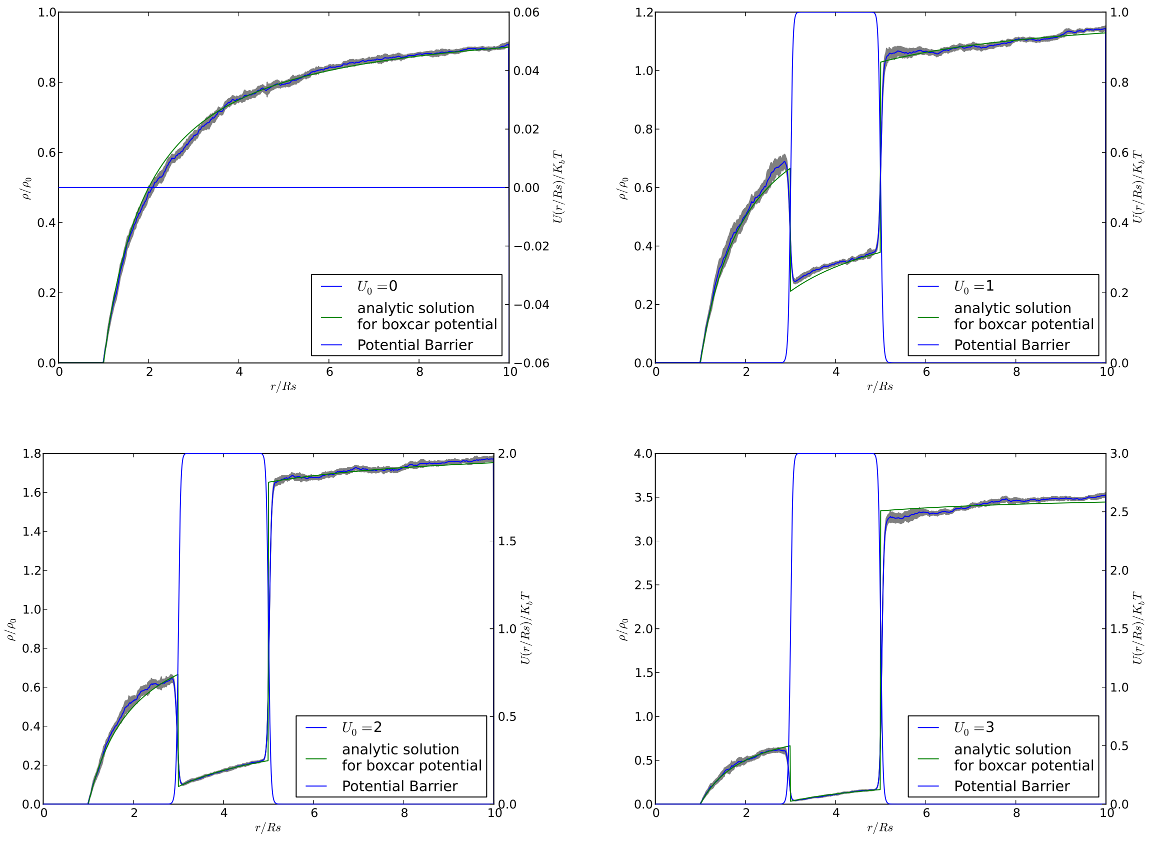

Figure 11: Density Profile for varying U_0

The simulation results obviously fit the analytic solution. Same holds for the calculated reaction rates as presented in the following Plot:

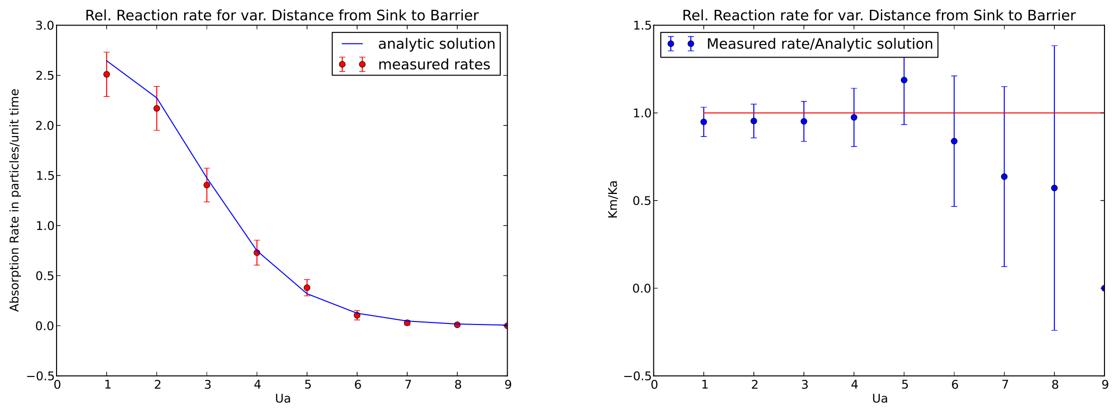

Figure 12: Absolute and relative Absorption rate for varying U_0

The simulation results obviously fit the analytic solution. Same holds for the calculated reaction rates as presented in the following Plot:

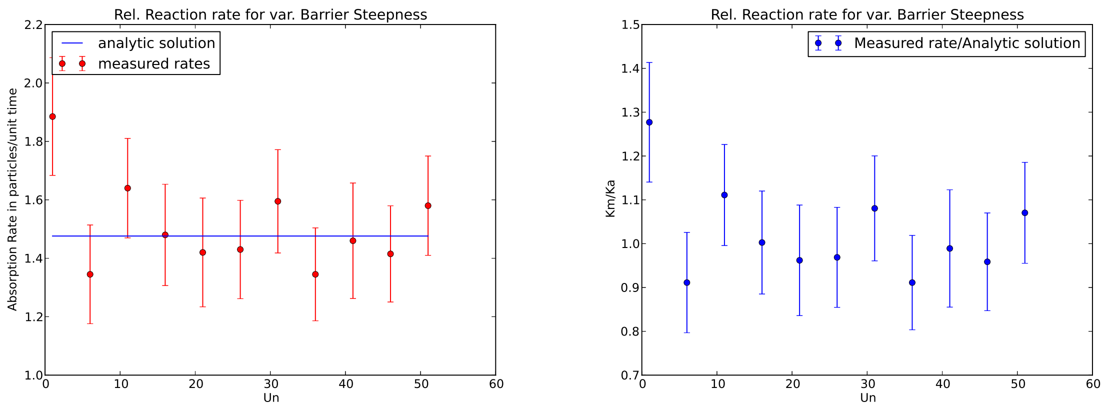

Figure 13: Absolute and relative Absorption rate for varying U_n

Appendices

A Code

A.1 Main Program

```
PROGRAM BDS

USE global
USE init
USE push
USE statistics

IMPLICIT NONE

      INTEGER         :: i, j
      INTEGER         :: counter=0
      REAL(8)         :: t=0, ct1, ct2, wt1, wt2
      REAL(8)         :: omp_get_wtime


      CALL RANDOM_SEED

      CALL open_output_files
```

```
    !read simulation parameters

    CALL init_parameters

    !Initialize particle possition randomly

    CALL init_particles

    !Initialize Statistics for histogramms and variable accumulation

    CALL init_statistics(nbins)

    !start iteration for particles
i = 0
    DO WHILE(t<=t1)
        i = i+1
        parold = par
        !Call statistic routines for density profile and absorption rate

        IF(t >= t0) THEN
            CALL dens_statistics_accum(nbins)
        ENDIF

        !Fluctuations of Potential/Substrate particles
        CALL update_state_of_potential

        !Move particles according to overdamped Langewin eq.
        CALL move_particles
        CALL maintain_boundary_conditions(counter)

        IF(t >= t0) THEN
            CALL rate_statistics_accum(counter, nbins)
        ENDIF
        t = t + dt

!          print*, 'msq/t - 2*d =', msqd/t - 6, ' md = ', md/t

    ENDDO

    !Output statistics to file

    CALL statistics_output(nbins)
    CALL close_output_files

END PROGRAM BDS
```

A.2 Initialization and Initial conditions

```
MODULE init

USE global
USE statistics

IMPLICIT NONE

CONTAINS

SUBROUTINE init_parameters

    USE global

    INTEGER :: in=10, tmp
    REAL(8) :: nparin, ntin, C
    CHARACTER(4)      :: trig, arg

    NAMELIST /PARAMETER/      nparin, D, KT, dt, t0, t1, Rd, Rs, U0,&
                              U1, Ua, Ub, Un, K01, K10, fmode, nbins

    !Read simulation parameters from file

    OPEN(unit=in, file='Parameters.in')
    READ(in,PARAMETER)
    CLOSE(in)

    npar = INT(nparin)

    !readin of terminal arguments and change simulation parameters

    CALL GETARG(2, arg)
    CALL GETARG(1, trig)

    read( arg, '(i10)') tmp

    IF(trig .EQ. 't0') t0 = REAL(tmp)
    IF(trig .EQ. 't1') t1 = REAL(tmp)
    IF(trig .EQ. 'D' ) D  = REAL(tmp)/1000.

    C = npar/(1/3.*Rd**3 - Rs/2.*Rd**2 - 1/3.*Rs**3 + Rs/2.*Rs**2)

    IF(trig .EQ. 'Rs') THEN
        npar = INT(C*(1/3.*Rd**3 - REAL(tmp)/2.*Rd**2 - 1/3.*REAL(tmp)**3
        Rs = REAL(tmp)
    ELSEIF(trig .EQ. 'Rd') THEN
        npar = INT(C*(1/3.*REAL(tmp)**3 - Rs/2.*REAL(tmp)**2 - 1/3.*Rs**3
        Rd = REAL(tmp)
    ENDIF

    IF(trig .EQ. 'U0' ) U0  = REAL(tmp)/10.0
    IF(trig .EQ. 'U1' ) U1  = REAL(tmp)/10.0
    IF(trig .EQ. 'Ua' ) Ua  = REAL(tmp)/10.0
    IF(trig .EQ. 'Ub' ) Ub  = REAL(tmp)/10.0
    IF(trig .EQ. 'Un' ) Un  = REAL(tmp)/10.0

    IF(trig .EQ. 'KDUb10')THEN
        K01 = REAL(tmp)*(D/Ub**2)
        K10 = K01
    ENDIF

    IF(trig .EQ. 'KDUb100')THEN
        K10 = REAL(tmp)*(Ub**2)
        K01 = K10
    ENDIF

    IF(trig .EQ. 'K100')THEN
        K01 = REAL(tmp)/(100.0)
        K10 = K01
    ENDIF

    IF(trig .EQ. 'K1000')THEN
        K10 = REAL(tmp)/(1000.0)
        K01 = K10
    ENDIF




    print*, 'npar = ', npar
    print*, 'D  = ', D
```

```
    print*, 'KT = ', KT
    print*, 'dt = ', dt
    print*, 't0 = ', t0
    print*, 't1 = ', t1
    print*, 'Rd = ', Rd
    print*, 'Rs = ', Rs
    print*, 'U0 = ', U0
    print*, 'U1 = ', U1
    print*, 'Ua = ', Ua
    print*, 'Ub = ', Ub
    print*, 'Un = ', Un
    print*, 'K01= ', K01
    print*, 'K10= ', K10
    print*, 'it = ', INT(t1/dt)

    msqd= 0
    md  = 0

END SUBROUTINE init_parameters

SUBROUTINE init_particles

    REAL(8), DIMENSION(3)      :: rand
    REAL(8)                    :: Rr, dr, theta, phi, a1, a2, a3, fracU0
    INTEGER                    :: i, j, n
    REAL(8), DIMENSION(nbins)  :: Cumm, r

    !Allocate particle array

    ALLOCATE(par(1:4,1:npar))
    ALLOCATE(parold(1:4,1:npar))

    !Initialize Particle Possitions - use inverse sampling to mime the or
    !debye solution for the density profile

    Cumm = 0
    Rr = Rs
    dr = (Rd-Rs)/(nbins)
    r = 0

    a1 = 1
    a2 = (1-Rs/(Rs+Ua))*exp(-U0/KT) + Rs/(Rs + Ua)
    a3 = Rs*(exp(U0/KT) - 1)*(1/(Rs+Ua) - 1/(Rs+Ua+Ub)) + 1
    DO i = 1,nbins
        r(i) = Rr
        IF(Rr > Rs .AND. Rr <= (Rs + Ua)) THEN
            Cumm(i) = Cumm(i-1) + (a1/3.*(r(i)+dr)**3 - Rs/2.*(r(i)+dr)**
        ELSEIF(Rr > (Rs + Ua) .AND. Rr <= (Rs + Ua + Ub)) THEN
            Cumm(i) = Cumm(i-1) + (a2/3.*(r(i) + dr)**3 - Rs/2.*(r(i) + d
        ELSEIF(Rr > (Rs + Ua + Ub) .AND. Rr <= Rd) THEN
            Cumm(i) = Cumm(i-1) + (a3/3.*(r(i) + dr)**3 - Rs/2.*(r(i) + d
        ENDIF
        IF(Cumm(i) .LE. Cumm(i-1)) print*, 'error!!', i, Rr, Cumm(i), Cum
        Rr = Rr + dr
    ENDDO
    Cumm = Cumm/Cumm(nbins)


    OPEN(unit=25, file='cummulative.out', action='write')
    DO i = 1,nbins-1
        WRITE(25,*) r(i), Cumm(i), Cumm(i+1) - Cumm(i)
    ENDDO
    CLOSE(25)


    DO i = 1,npar
        CALL RANDOM_NUMBER(rand)
        DO j = 1,nbins
            IF(rand(1) < Cumm(j)) THEN
                Rr = r(j) + dr*(rand(2)-0.5)
                EXIT
            ENDIF
        ENDDO
        theta = 2*pi*rand(2)
        phi   = ACOS(2*rand(3) - 1)
        par(1,i) = Rr*COS(phi)*SIN(theta)
        par(2,i) = Rr*SIN(phi)*SIN(theta)
        par(3,i) = Rr*COS(theta)
    ENDDO

!Initialize particle state according to detailled equilibrium

    IF(fmode == 0) THEN
        fracU0 = K10/(K10+K01)*npar
        fracU1 = K01/(K10+K01)*npar

        CALL RANDOM_NUMBER(randU)

        IF(randU <= fracU0) THEN
            par(4,:) = 0
        ELSEIF(randU > fracU0) THEN
            par(4,:) = 1
        ENDIF

    ELSEIF(fmode == 1) THEN
        fracU0 = K10/(K10+K01)*npar
        fracU1 = K01/(K10+K01)*npar

        DO i = 1,npar
            IF(i <= fracU0) THEN
                par(4,i) = 0
            ELSEIF(i > fracU0) THEN
                par(4,i) = 1
            ENDIF
        ENDDO
    ENDIF
END SUBROUTINE init_particles

SUBROUTINE init_statistics(bins)

    INTEGER, INTENT(in) :: bins

    CALL clear5(2*bins+1,500)

END SUBROUTINE init_statistics

SUBROUTINE open_output_files

    dens_final  = 20
    rate_final  = 21

    OPEN(unit=dens_final, file="dens_data.out", action="write")
    OPEN(unit=rate_final, file="rate_data.out", action="write")


END SUBROUTINE open_output_files
```

```
SUBROUTINE close_output_files

    CLOSE(dens_final)
    CLOSE(rate_final)

END SUBROUTINE close_output_files

END MODULE init
```

A.3 Equations of Motion and Boundary Conditions

```
MODULE push

USE global

IMPLICIT NONE

CONTAINS

SUBROUTINE move_particles

    REAL(8), DIMENSION(3,npar)    :: erand
    REAL(8), DIMENSION(3,npar)    :: nrand1, nrand2
    REAL(8), DIMENSION(3,npar)    :: f_eff
    INTEGER                       :: i, j

    CALL RANDOM_NUMBER(nrand1)
    CALL RANDOM_NUMBER(nrand2)

    !Compute gaussian random numbers for random force using Box Muller tr

    !$OMP PARALLEL DO
    DO i = 1,npar
        DO j = 1,3
            erand(j,i) = SQRT(-2*LOG(nrand1(j,i)))*COS(2*pi*nrand2(j,i))
        ENDDO
    ENDDO
    !$OMP END PARALLEL DO

    !Calculate effective interaction forces
    !f_eff = -D/KT*grad_U*dt

    f_eff = 0
    !$OMP PARALLEL DO
    DO i = 1,npar
        CALL grad_U(par(:,i), f_eff(:,i))
    ENDDO
    !$OMP END PARALLEL DO

    !Calculate sigma for random force using the Einstein Smoluchowski rel

    erand = SQRT(2*D*dt)*erand

    !Apply random force to Particles


    !$OMP PARALLEL DO
    DO i = 1,npar
        DO j = 1,3
            par(j,i) = par(j,i) +f_eff(j,i) + erand(j,i)
        ENDDO
    ENDDO
    !$OMP END PARALLEL DO


END SUBROUTINE move_particles

SUBROUTINE grad_U(X, f_eff)

    REAL(8), DIMENSION(4), INTENT(in)    :: X
    REAL(8), DIMENSION(3)                :: Rn, R
    REAL(8)                              :: Rr, grad_Ur, a, b
    REAL(8), DIMENSION(3), INTENT(out)   :: f_eff
    INTEGER                              :: state

    R = X(1:3)
    state = X(4)

    Rr = SQRT(DOT_PRODUCT(R,R))
    Rn = R/Rr

    a = Rs + Ua + 0.5*Ub
    b = Ub

    IF(state == 0) THEN
        grad_Ur = -4*Un*U0*((2/b*(Rr-a))**(2*Un-1))/b/((2/b*(Rr-a))**(2*U
    ELSEIF(state == 1) THEN
```

```
        grad_Ur = -4*Un*U1*((2/b*(Rr-a))**(2*Un-1))/b/((2/b*(Rr-a))**(2*U
    ENDIF
    f_eff = -D/KT*grad_Ur*dt*Rn

END SUBROUTINE grad_U

SUBROUTINE maintain_boundary_conditions(counter)


    REAL(8), DIMENSION(3)    :: px    !closest point on trajectory to sink
    REAL(8), DIMENSION(3)    :: A, B, AB !Work vectors
    REAL(8)                  :: Rr    !minimum distance of Particle to sink
    REAL(8), DIMENSION(3)    :: dr    !random displacement vector
    REAL(8), DIMENSION(4)    :: rand
    REAL(8)                  :: theta, phi !angles for random sphere point
    INTEGER, INTENT(out)     :: counter !counter for absorbed particles
    INTEGER                  :: i, j, icase
    REAL(8), DIMENSION(npar):: dmsqr
    REAL(8), DIMENSION(npar):: dmr
    counter = 0

    !calculate mean square displacement and mean displacement to check...

    DO i = 1,npar
        dmsqr(i) = DOT_PRODUCT(par(1:3,i) - parold(1:3,i), par(1:3,i) - p
        dmr(i)   = SUM(par(1:3,i) - parold(1:3,i))
    ENDDO

    msqd = msqd + SUM(dmsqr)/npar
    md   = md   + SUM(dmr)/npar


    !$OMP DO REDUCTION(+:counter) PRIVATE(Rr, rand, dr, A, B, AB, px, the
    DO i = 1,npar

        !Calculate closest point of particle trajectory to sink

        A = parold(1:3,i)
        B = par(1:3,i)
        AB = parold(1:3,i) - par(1:3,i)

        px = A + DOT_PRODUCT(A,AB)*AB/DOT_PRODUCT(AB,AB)
        IF( DOT_PRODUCT((px-A),(px-B)) < 0 )THEN
            Rr = SQRT(DOT_PRODUCT(px,px))
        ELSEIF( DOT_PRODUCT((px-A),(px-B)) >= 0 )THEN
            Rr = SQRT(DOT_PRODUCT(par(1:3,i),par(1:3,i)))
        ENDIF

        !Set particles to domain boundary (ensure steady state solution
        !when they hit the Sink and count them for rate statistics

        IF( Rr < Rs )THEN

            !increase counter for absorbed particles

            counter = counter + 1

            !calculate random possition at domain boundary

            CALL RANDOM_NUMBER(rand)
            theta = 2*pi*rand(2)
            phi   = ACOS(2*rand(3) - 1)
            dr(1) = COS(phi)*SIN(theta)
            dr(2) = SIN(phi)*SIN(theta)
            dr(3) = COS(theta)
            par(1:3,i) = (Rs + Rd - Rr)*dr
        ELSEIF( Rr > Rd )THEN

        !Reset particles to some random place at the boundary if they exc
        !simulation domain to have zero flux through domain boundary

            CALL RANDOM_NUMBER(rand)
            theta = 2*pi*rand(2)
            phi   = ACOS(2*rand(3) - 1)
            dr(1) = COS(phi)*SIN(theta)
            dr(2) = SIN(phi)*SIN(theta)
            dr(3) = COS(theta)
            par(1:3,i) = (2*Rd - Rr)*dr
        ENDIF
    ENDDO
    !$OMP END DO

END SUBROUTINE maintain_boundary_conditions

SUBROUTINE update_state_of_potential
```

```
    INTEGER                     :: i
    REAL, DIMENSION(npar)       :: rand
    REAL                        :: tmp

    IF(fmode == 0) THEN
        CALL RANDOM_NUMBER(tmp)
        rand = tmp
    ELSEIF(fmode == 1) THEN
        CALL RANDOM_NUMBER(rand)
    ENDIF
    !$OMP PARALLEL DO
    DO i = 1,npar
        IF(rand(i) < K01*dt .AND. par(4,i) == 0)THEN
            par(4,i) = 1
            CYCLE
        ELSEIF(rand(i) < K10*dt .AND. par(4,i) == 1)THEN
            par(4,i) = 0
        ENDIF
    ENDDO
    !$OMP END PARALLEL DO
END SUBROUTINE update_state_of_potential

END MODULE push
```

A.4 Variable calculation and I/O

```
MODULE statistics

USE global

IMPLICIT NONE

CONTAINS

    SUBROUTINE dens_statistics_accum(bins)

    INTEGER, INTENT(in) :: bins
    INTEGER             :: i, j
    REAL(8), DIMENSION(2,bins) :: hist

    !calculate histogramm of particle distance to sink

    CALL histogramm(par, hist, bins)
```

```
!accumulate density histrogramm in statistics module

DO i = 1,2
    DO j = 1,bins
        CALL accum5(j+(i-1)*bins,hist(i,j))
    ENDDO
ENDDO

END SUBROUTINE dens_statistics_accum

SUBROUTINE histogramm(X, Xhist, bins)

INTEGER, INTENT(in) :: bins
INTEGER :: i, imax, binnumber
REAL    :: r, vbin
REAL(8), DIMENSION(3)   :: POS
REAL(8), DIMENSION(2,bins), INTENT(out) :: Xhist
REAL(8), DIMENSION(:,:), INTENT(in)  :: X

Xhist = 0
imax = SIZE(X,2)

!build histogramm according to particle distance to sink

!$OMP PARALLEL DO REDUCTION(+:Xhist) PRIVATE(binnumber, r)
DO i = 1,imax
    r = SQRT(DOT_PRODUCT(X(1:3,i),X(1:3,i)))
    binnumber = INT(r/Rd*bins)
    IF(binnumber .LE. bins) THEN
        if(binnumber .LE. 0)print*, i, binnumber, r, Rd, X(:,i)
        IF(X(4,i) == 0) THEN
            Xhist(1,binnumber) = Xhist(1,binnumber) + 1
        ELSEIF(X(4,i) == 1) THEN
            Xhist(2,binnumber) = Xhist(2,binnumber) + 1
        ENDIF
    ENDIF
ENDDO
!$OMP END PARALLEL DO

END SUBROUTINE histogramm

SUBROUTINE rate_statistics_accum(counter, bins)
```

```
    INTEGER, INTENT(in) :: counter, bins
    REAL(8)              :: rate

    !calculate absorption rate: rate=apsorbed particles/unit time

    rate = counter/dt

    !accumulate absorption rate in statistics module

    CALL accum5(2*bins+1,rate)

    END SUBROUTINE rate_statistics_accum

    SUBROUTINE statistics_output(bins)

    INTEGER, INTENT(in) :: bins
    INTEGER              :: i
    REAL(8)              :: aver5, sigma5, a, b, Rr

    !Write statistics output to file


    a = Rs + Ua + 0.5*Ub
    b = Ub

    DO i = 1,bins
        Rr = real(i)*Rd/real(bins)
        WRITE(dens_final, "(7f15.4)")   (REAL(i))/REAL(bins)*Rd, aver5(i)
                                        aver5(i + bins), sigma5(i + bins)
                                  -4.*Un*U0*((2./b*(Rr-a))**(2.*Un-1.))/b/(
                                  U0/((2./b*(Rr-a))**(2.*Un) + 1.)
    ENDDO

    WRITE(rate_final, *) "this file contains simulation parameters and me
    WRITE(rate_final, *)
    WRITE(rate_final, "(14f15.4)") REAL(npar), D, Rs, Rd, t0, t1, U0, U1,

    END SUBROUTINE statistics_output

END MODULE statistics
```

A.5 Box averages, Jackknife Binning and Auto Correlation Correction

```
module stat5_data
!************************* stat5 ***********************************
!      statistical analysis using                          B Bunk 1993
!           o data blocking                                rev  2/2005
!           o autocorrelation analysis
!           o jackknife

!         nvar   : no. of variables
!         nbmax : max. no. of blocks per variable
! - - - - - - - - - - - - - - - - - - - - - - - - - - - -

      integer, save                              :: nvar = 0, nbmax = 0
      integer, save, dimension(:), allocatable   :: nbl, lbl, lnew
      real(8), save, dimension(:,:), allocatable :: blksum
      real(8), save, dimension(:), allocatable   :: sqsum
end module stat5_data
! - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!      allocate and clear counters
subroutine clear5(nvar1,nbmax1)

      use stat5_data

      real(8) zero
      parameter(zero=0.)

      if(nvar1.lt.1)     stop 'error in clear5: nvar is invalid'
      if(nbmax1.lt.2  .or. mod(nbmax1,2).ne.0) &
                         stop 'error in clear5: nbmax is invalid'

      if (nvar1 /= nvar .or. nbmax1 /= nbmax) then
         if (nvar > 0) then
            deallocate(nbl, lbl, lnew, blksum, sqsum, stat=ierror)
            if(ierror /= 0) stop 'error in clear5: deallocate failed'
         endif
         allocate(nbl(nvar1), lbl(nvar1), lnew(nvar1), &
            blksum(nvar1,nbmax1), sqsum(nvar1), stat=ierror)
         if (ierror /= 0) stop 'error in clear5: allocate failed'
      endif

      nvar=nvar1
      nbmax=nbmax1

      do 10 ivar=1,nvar
      nbl(ivar)=0
      lbl(ivar)=1
      lnew(ivar)=0
      sqsum(ivar)=zero
      do 10 ibl=1,nbmax
      blksum(ivar,ibl)=zero
10    continue
end subroutine
! ----------------------------
!     accumulate data

subroutine accum5(ivar,value)

      use stat5_data

      real(8) value,zero
      parameter(zero=0.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
                  stop 'error in accum5: ivar out of range'

      if(nbl(ivar) .eq. nbmax)then
            do 10 ibl=1,nbmax/2
            blksum(ivar,ibl)=blksum(ivar,2*ibl-1) + blksum(ivar,2*ibl)
10          continue
            do 20 ibl=nbmax/2+1,nbmax
            blksum(ivar,ibl)=zero
20          continue
            nbl(ivar)=nbmax/2
            lbl(ivar)=2*lbl(ivar)
      endif

      iblnew=nbl(ivar) + 1
      blksum(ivar,iblnew)=blksum(ivar,iblnew) + value
      sqsum(ivar)=sqsum(ivar) + value**2
      lnew(ivar)=lnew(ivar) + 1

      if(lnew(ivar) .eq. lbl(ivar))then
            nbl(ivar)=iblnew
            lnew(ivar)=0
      endif
end subroutine
! ----------------------------
!       compute averages

function aver5(ivar)

      use stat5_data

      real(8) aver5,zero
      parameter(zero=0.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
                  stop 'error in aver5: ivar out of range'

      aver5=zero
      nmeas=nbl(ivar)*lbl(ivar) + lnew(ivar)
      if(nmeas.eq.0) return
      do 10 ibl=1,min(nbl(ivar)+1,nbmax)
      aver5=aver5 + blksum(ivar,ibl)
10      continue
      aver5=aver5/nmeas
end function
! ------------------------------
!       compute errors

function sigma5(ivar)

      use stat5_data

      real(8) sigma5,var5,zero
      parameter(zero=0.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
                  stop 'error in sigma5: ivar out of range'

      sigma5=sqrt(max( var5(ivar) , zero ))
end function
! ------------------------------
!       compute variances

function var5(ivar)

      use stat5_data

      real(8) var5
      real(8) av,d,gam,gamvar,zero,one
      dimension d(nbmax)
      parameter(zero=0.,one=1.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
                  stop 'error in var5: ivar out of range'

      var5=zero
      nb=nbl(ivar)
      if(nb.lt.2)return
      nmeas=nb*lbl(ivar) + lnew(ivar)

      av=zero
      do 10 ib=1,nb
      av=av + blksum(ivar,ib)
10      continue
      av=av/nb

      do 20 ib=1,nb
      d(ib)=blksum(ivar,ib) - av
      var5=var5 + d(ib)**2
20      continue
      if(var5 .le. zero)return
      var5=var5/nb
      gamvar=var5**2

      do 30 it=1,nb-1
      gam=zero
      do 40 ib=1,nb-it
      gam=gam + d(ib)*d(ib+it)
40      continue
      gam=gam/(nb-it)
      if( gam .le. sqrt(gamvar/(nb-it)) )goto 31
      var5=var5 + 2*gam
      gamvar=gamvar + 2*gam**2
30      continue
31      continue

      var5=(var5/nmeas)/lbl(ivar)
end function
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!       compute elements of the covariance matrix
!               - sum off-diagonal gammas
!          - can violate positivity
!     better use covar5 !

function cov5(ivar,jvar)

      use stat5_data

      real(8) cov5
      real(8) var5,avi,avj,di,dj,gamij,gamji,gamii,gamjj,gamvar
      real(8) sgn,zero,one
      dimension di(nbmax),dj(nbmax)
      parameter(zero=0.,one=1.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
            stop 'error in cov5: ivar out of range'
      if(jvar.lt.1 .or. jvar.gt.nvar) &
            stop 'error in cov5: jvar out of range'

      if(ivar .eq. jvar)then
            cov5=var5(ivar)
            return
      endif

      cov5=zero
      nb=nbl(ivar)
      if(nb.lt.2) return
      nmeas=nb*lbl(ivar) + lnew(ivar)
      if(nmeas .ne. nbl(jvar)*lbl(jvar)+lnew(jvar) )then
            print*,'warning in cov5: mismatch of measurements'
            return
      endif

      avi=zero
      avj=zero
      do 10 ib=1,nb
      avi=avi + blksum(ivar,ib)
      avj=avj + blksum(jvar,ib)
10    continue
      avi=avi/nb
      avj=avj/nb

      gamii=zero
      gamjj=zero
```

```
       do 20 ib=1,nb
       di(ib)=blksum(ivar,ib) - avi
       dj(ib)=blksum(jvar,ib) - avj
       cov5=cov5 + di(ib)*dj(ib)
       gamii=gamii + di(ib)**2
       gamjj=gamjj + dj(ib)**2
20     continue
       cov5=cov5/nb
       sgn=sign(one,cov5)
       gamvar=gamii*gamjj/nb**2 + cov5**2

       do 30 it=1,nb-1
       gamij=zero
       gamji=zero
       gamii=zero
       gamjj=zero
       do 40 ib=1,nb-it
       gamij=gamij + di(ib+it)*dj(ib)
       gamji=gamji + dj(ib+it)*di(ib)
       gamii=gamii + di(ib)*di(ib+it)
       gamjj=gamjj + dj(ib)*dj(ib+it)
40     continue
       gamij=gamij/(nb-it)
       gamji=gamji/(nb-it)
       gamii=gamii/(nb-it)
       gamjj=gamjj/(nb-it)
       if( sgn*(gamij+gamji) .le. sqrt(2*gamvar/(nb-it)) )goto 31
       cov5=cov5 + gamij + gamji
       gamvar=gamvar + 2*(gamii*gamjj + gamij*gamji)
30     continue
31     continue

       cov5=(cov5/nmeas)/lbl(ivar)
end function
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!      compute elements of the covariance matrix
!            - rescale non-diagonal elements
!                  with autocorrelation factors
!                  from the diagonal
!            - positive matrix

function covar5(ivar,jvar)
```

```
      use stat5_data

      real(8) covar5
      real(8) var5,avi,avj,di,dj,vari,varj,zero
      parameter(zero=0.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
            stop 'error in covar5: ivar out of range'
      if(jvar.lt.1 .or. jvar.gt.nvar) &
            stop 'error in covar5: jvar out of range'

      if(ivar .eq. jvar)then
         covar5=var5(ivar)
         return
      endif

      covar5=zero
      nb=nbl(ivar)
      if(nb.lt.2) return
      nmeas=nb*lbl(ivar) + lnew(ivar)

      if(nmeas .ne. nbl(jvar)*lbl(jvar)+lnew(jvar) )then
         print*,'warning in covar5: mismatch of measurements'
         return
      endif

      avi=zero
      avj=zero
      do 10 ib=1,nb
      avi=avi + blksum(ivar,ib)
      avj=avj + blksum(jvar,ib)
10    continue
      avi=avi/nb
      avj=avj/nb

      vari=zero
      varj=zero
      do 20 ib=1,nb
      di=blksum(ivar,ib) - avi
      dj=blksum(jvar,ib) - avj
      covar5=covar5 + di*dj
      vari=vari + di**2
      varj=varj + dj**2
```

```
20      continue
        if(vari.le.zero .or. varj.le.zero)return

        covar5=covar5 * sqrt( var5(ivar)*var5(jvar)/(vari*varj) )
end function
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!       estimate integrated auto-correlations:
!               var(average) = var(single)/nmeas * rsq
!                           rsq = coth( 1/(2*tau) )
!                               = 2 * tauint

function tau5(ivar)

        use stat5_data

        real(8) tau5,rsq5,tauint5
        real(8) aver5,var5,var,rsq,zero,one,oneeps
        parameter(zero=0.,one=1.,oneeps=1.000001d0)

        if(ivar.lt.1 .or. ivar.gt.nvar) &
                stop 'error in tau5: ivar out of range'

        tau5=zero

        nmeas=nbl(ivar)*lbl(ivar) + lnew(ivar)
        if(nmeas.lt.2) return

        var=sqsum(ivar)/nmeas - aver5(ivar)**2
        if(var.le.zero) return

        rsq=var5(ivar) / var * nmeas
        if(rsq.le.oneeps) return
        tau5=one/log( (rsq+one)/(rsq-one) )
return
! . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
entry rsq5(ivar)

        if(ivar.lt.1 .or. ivar.gt.nvar) &
                stop 'error in rsq5: ivar out of range'

        rsq5=one

        nmeas=nbl(ivar)*lbl(ivar) + lnew(ivar)
```

```
      if(nmeas.lt.2) return

      var=sqsum(ivar)/nmeas - aver5(ivar)**2
      if(var.le.zero) return

      rsq5=var5(ivar) / var * nmeas
return
! . . . . . . . . . . . . . . . . . . . . . . . . . . . .
entry tauint5(ivar)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
            stop 'error in tauint5: ivar out of range'

      tauint5=.5

      nmeas=nbl(ivar)*lbl(ivar) + lnew(ivar)
      if(nmeas.lt.2) return

      var=sqsum(ivar)/nmeas - aver5(ivar)**2
      if(var.le.zero) return

      tauint5=.5 * var5(ivar) / var * nmeas
end function
! - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     compute jackknife blocks

!        ivar    : variable (input)
!        nb      : no. of blocks (output)
!        bj      : real(8) vector of jackknife blocks
!                  bj(ib), ib=1..nb (output)

subroutine jackout5(ivar,nb,bj)

      use stat5_data

      real(8) bj
      dimension bj(nbmax)
      real(8) bsum,zero,one
      parameter(zero=0.,one=1.)

      if(ivar.lt.1 .or. ivar.gt.nvar) &
            stop 'error in jackout5: ivar out of range'
      nb=nbl(ivar)
      if(nb .lt. 2) stop 'error in jackout5: nb < 2'

      bsum=zero
      do 10 ib=1,nb
      bsum=bsum + blksum(ivar,ib)
10    continue

      do 20 ib=1,nb
      bj(ib)=(bsum - blksum(ivar,ib))/(lbl(ivar)*(nb-1))
20    continue
end subroutine
! - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     jackknife analysis for a vector of (function) values

!        nb     : number of jackknife values, nb > 1 (input)
!        fj     : real(8) vector of jackknife values,
!                 fj(ib), ib=1..nb (input)
!        aver   : function average (output)
!        sigma  : error estimate (output)

subroutine jackeval5(nb,fj,aver,sigma)

      real(8) fj,aver,sigma
      dimension fj(nb)
      real(8) bsum,d,var,gam,gamvar,zero,one
      dimension d(nb)
      parameter(zero=0.,one=1.)

      aver=zero
      sigma=zero
      if(nb.lt.2)return

      bsum=zero
      do 10 ib=1,nb
      bsum=bsum + fj(ib)
10    continue
      aver=bsum/nb

      gam=zero
      do 20 ib=1,nb
      d(ib)=fj(ib) - aver
      gam=gam + d(ib)**2
```

```
20      continue
        if(gam .le. zero)return
        gam=gam/nb

        var=gam
        gamvar=gam**2
        do 30 it=1,nb-1
        gam=zero
        do 40 ib=1,nb-it
        gam=gam + d(ib)*d(ib+it)
40      continue
        gam=gam/(nb-it)
        if( gam .le. sqrt(gamvar/(nb-it)) )goto 31
        var=var + 2*gam
        gamvar=gamvar + 2*gam**2
30      continue
31      continue

        sigma=(nb-1)*sqrt(var/nb)
end subroutine
!- - - - - - - - - - - - - - - - - - - - - - - - - - - -
!       jackknife evaluation - custom call

!           fct     : function of statistical variables (user-defined)
!                           function fct(nvar, a)
!                           real(8) a(nvar)
!                     with a(i), i=1..nvar : variables, as in stat5
!           aver    : function average (output)
!           sigma   : error estimate (output)

!       calls jackeval5

!       note: is is assumed that all (relevant) variables were accumulated
!             in sync, with ivar=1 as prototype.

subroutine jack5(fct,aver,sigma)

        use stat5_data

        real(8) fct,aver,sigma
        real(8) bsum,bj,fj,zero,one
        dimension bsum(nvar),bj(nvar),fj(nbmax)
        parameter(zero=0.,one=1.)
        external fct

        aver=zero
        sigma=zero
        nb=nbl(1)
        lb=lbl(1)
        if(nb.lt.2)return

        do 10 ivar=1,nvar
        bsum(ivar)=zero
        do 11 ib=1,nb
        bsum(ivar)=bsum(ivar) + blksum(ivar,ib)
11      continue
10      continue

        do 15 ib=1,nb
        do 16 ivar=1,nvar
        bj(ivar)=(bsum(ivar) - blksum(ivar,ib))/(lb*(nb-1))
16      continue
        fj(ib)=fct(nvar,bj)
15      continue

        call jackeval5(nb,fj,aver,sigma)
end subroutine
! - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!       save and restore counters (unformatted/formatted)

subroutine save5(iunit)

        use stat5_data

        write(iunit)nvar,nbmax
        write(iunit)(nbl(i),lbl(i),lnew(i),i=1,nvar) &
                ,((blksum(i,ibl),ibl=1,nbmax),i=1,nvar) &
                ,(sqsum(i),i=1,nvar)
return

entry savef5(iunit)

        write(iunit,*)nvar,nbmax
        write(iunit,*)(nbl(i),lbl(i),lnew(i),i=1,nvar) &
                ,((blksum(i,ibl),ibl=1,nbmax),i=1,nvar) &
                ,(sqsum(i),i=1,nvar)
```

```
return

entry get5(iunit)
entry getst5(iunit)

      read(iunit)nvar1,nbmax1

      if(nvar1.lt.1)     stop 'error in get5: nvar is invalid'
      if(nbmax1.lt.2  .or. mod(nbmax1,2).ne.0) &
                         stop 'error in get5: nbmax is invalid'

      if (nvar1 /= nvar .or. nbmax1 /= nbmax) then
         if (nvar > 0) then
            deallocate(nbl, lbl, lnew, blksum, sqsum, stat=ierror)
            if(ierror /= 0) stop 'error in get5: deallocate failed'
         endif
         allocate(nbl(nvar1), lbl(nvar1), lnew(nvar1), &
            blksum(nvar1,nbmax1), sqsum(nvar1), stat=ierror)
         if (ierror /= 0) stop 'error in get5: allocate failed'
      endif

      nvar=nvar1
      nbmax=nbmax1

      read(iunit)(nbl(i),lbl(i),lnew(i),i=1,nvar) &
            ,((blksum(i,ibl),ibl=1,nbmax),i=1,nvar) &
            ,(sqsum(i),i=1,nvar)
return

entry getf5(iunit)

      read(iunit,*)nvar1,nbmax1

      if(nvar1.lt.1)     stop 'error in getf5: nvar is invalid'
      if(nbmax1.lt.2  .or. mod(nbmax1,2).ne.0) &
                         stop 'error in getf5: nbmax is invalid'

      if (nvar1 /= nvar .or. nbmax1 /= nbmax) then
         if (nvar > 0) then
            deallocate(nbl, lbl, lnew, blksum, sqsum, stat=ierror)
            if(ierror /= 0) stop 'error in getf5: deallocate failed'
         endif
         allocate(nbl(nvar1), lbl(nvar1), lnew(nvar1), &
           blksum(nvar1,nbmax1), sqsum(nvar1), stat=ierror)
        if (ierror /= 0) stop 'error in getf5: allocate failed'
    endif

    nvar=nvar1
    nbmax=nbmax1

    read(iunit,*)(nbl(i),lbl(i),lnew(i),i=1,nvar) &
           ,((blksum(i,ibl),ibl=1,nbmax),i=1,nvar) &
           ,(sqsum(i),i=1,nvar)
end subroutine
```