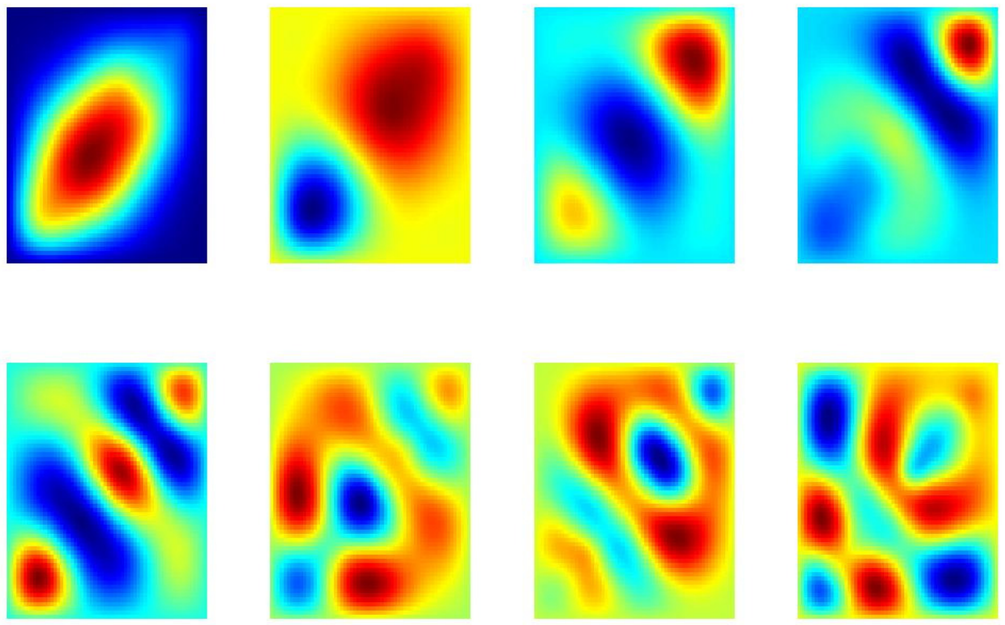
rijksuniversiteit groningen

faculteit Wiskunde en Natuurwetenschappen

Numerical Analysis of an Implicit Method Solving Non-Linear Stochastic Partial Differential Equations



Bachelor Thesis in Applied Mathematics

April 2012

Student: Erik Mulder

First supervisor: Dr. ir. F. W. Wubs

Second supervisor: Prof. Dr. E. C. Wit

Contents

1 Introduction

A multitude of problems in fields ranging from physics to economics deal with dynamical systems containing uncertain processes. Stochastic partial differential equations (SPDE's) are a popular tool in modelling such problems. In this project we will focus on SPDE's modelling linear diffusion and non-linear reaction-diffusion problems. The solutions to these problems will be partially stochastic, which means that they will contain several components of which only the parameters of their probability distributions are known. An example of such a decomposition is the velocity of a particle in a fluid, which can be thought of as having a deterministic "bulk" component and a stochastic part due to random collisions.

Only a small set of random variables will determine the stochastic part of a solution, which makes it easy to keep track of statistical properties and will reduce computational costs. This dimensional reduction will be compensated by defining the general influence of these stochastic variables with orthogonal matrices, which will prove to be a convenient approach when factoring the SPDE's into several simpler equations (sections 3 and 4). The idea and origin of this representation can be found in [7]. Additionally, the models we study will contain a time-dependent forcing, which we will divide into a deterministic and a stochastic component as well. For a large part of this project we will focus on the stochastic part and neglect any deterministic forcing.

To solve an SPDE we first transform it into a system of ordinary differential equations, which we then solve using a time-integrator. In section 3 we will briefly explain this approach; more can be found in [9]. We need a specific time-integrator that is able to integrate the stochastic part of the equation. This integration is performed along a *Wiener path*, on which we will elaborate in section 2. The numerical method we will use to implement this type of integration is known as an Euler-Maruyama method, which gives an explicit discretisation of the system of ordinary differential equations.

Explicitly solving the SPDE's is fast and easy, yet such methods are highly unstable. Therefore it is justified to derive a semi-implicit scheme, which computes the unknown solution via a combination of known and unknown solutions from the current and previous time-steps. A discussion on the choice of implicit methods can be found in section 3.1.1. The implicit approach is stable but may have a high computational cost. We will implement both methods and compare the results.

The stochastic variables and their probability distributions will be subject of extensive study. We will analyse their probability distributions and the correlations between them in models differing only by a diffusion coefficient. For this we will regard each step in a solution as a single experiment and perform these experiments over and over to obtain useful estimates of the probability density functions. Here we will make use of a great tool in data analysis, known as principal component analysis (PCA). It allows us to view the variables without correlations and order them by significance. In section 5.1 we will shortly explain its workings. More can be found in [4].

Finally, we are interested in the stability of solutions and whether we can indicate a steady state in the stochastic models. This can be difficult since there is always some random fluctuation present. We will therefore regard a state to be stable whenever the variance of each such fluctuation can be accurately predicted. Hence we will not obtain a true steady state, but something that is stable in a statistical way.

2 Preliminaries

As explained we will treat a variety of stochastic diffusion equations. The stochastic part of such equations is modelled using a random number generator. Although there is a deterministic algorithm determining the pseudo-random numbers, these are sufficiently random to simulate stochastic processes. To compare different methods we are able to *seed* this random number generator such that the sequence of pseudo-random numbers is equal in every simulation. We let the random variables be normally distributed, such that the probability density functions are determined by the mean (which we will denote μ) and variance (σ^2). To be able to perform a principal component analysis we need to be certain that the probability distributions of the random variables are completely determined by these two parameters.

Solving a differential equation implies that we are going to perform some integrations. In a deterministic case these are straightforward and known from basic calculus. In the stochastic case this is a bit more difficult. Here we will have to integrate a stochastic process with respect to another stochastic process. We will concern ourselves with techniques formulated in stochastic Ito calculus [3] and integrate with respect to a *Wiener process*, which is a non-differentiable stochastic process also known as Brownian motion. According to [1], a Wiener process is a random variable $W(t)$, with $t \in [0, T]$, satisfying the following conditions:

1. W(0) = 0
2. Given (s, t) such that $0 \leq s < t \leq T$, then $W(t) - W(s) \sim \sqrt{t - s}N(0, 1)$
3. Given (s, t, u, v) such that $0 \leq s < t < u < v \leq T$, then the differences $W(t) - W(s)$ and $W(v) - W(u)$ are independent.

Hence the Wiener process describes a path in which the increments are independent, identically distributed (*i.i.d.*) random variables. In the following sections we will deal with discretised differential equations which implies that we will have fixed size differentials, not infinitesimal ones. As the stochastic integrations are performed with respect to the Wiener path, this means that the discrete differential forms will contain increments from the Wiener path on a fixed interval, which we will denote $\Delta\mathcal{W} = W(\tau_i) - W(\tau_{i-1})$. Then we can say that each $\Delta\mathcal{W}$ is an i.i.d. random variable and we can let the pseudo-random number generator generate these increments.

To solve the differential equations we will focus on implicit and explicit Euler schemes. The stochastic variant of the explicit Euler scheme is known as the Euler-Maruyama method. This method incorporates the stochastic integration discussed above. Finding a suitable implicit form of the Euler-Maruayama method can be problematic. This will be discussed in section 3.1.1. We will now start with the derivation of several methods solving linear and non-linear SPDE's and state the first diffusion problem.

3 A Linear Stochastic Diffusion Equation

Let us consider the following one-dimensional problem:

$$\frac{\partial}{\partial t}u = a\frac{\partial^2}{\partial x^2}u - f \tag{3.1}$$

Which we define on an interval $[a, b]$, with initial condition $u(x, 0) = \phi(x)$ and Dirichlet boundary conditions $u(a, t) = D_a$, $u(b, t) = D_b$. The solution $u(x, t)$ and the forcing $f(x, t)$ will contain a deterministic and a stochastic part, on which we will elaborate in section 3.2. First we will give the basics of the numerical scheme solving equation (3.1).

3.1 The Basic Numerical Scheme

To numerically solve this equation we apply the method of lines, transforming the partial differential equation (PDE) into a system of ordinary differential equations (ODE's). We will shortly summarize what can also be found in [9]. We discretise in space using the finite difference method with central differences, using a uniform grid of size n with elements of size h. This discretisation gets rid of the spatial derivatives and we obtain a system of ordinary differential equations in matrix form:

$$\frac{d}{dt}\mathbf{u} = A\mathbf{u} - \mathbf{f} \tag{3.2}$$

$$\begin{aligned} \text{Where:} \quad \mathbf{u} &= [u_1, u_2, \dots, u_n]^T \\ \mathbf{f} &= [f_1, \dots, f_n]^T \quad \text{and} \\ A &\in \mathbb{R}^{n\times n} \qquad \text{given by:} \\ A &= \frac{a}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \end{aligned}$$

Note that in this spatial discretisation the Dirichlet boundary conditions have to be available to the solution through so called "ghost cells" or *fictive grid points* [9], which are implemented via the forcing $\mathbf{f}$. This will be clear from the implementation in appendix A.5.

Now we have a system of stochastic ODE's. First we will rewrite this into a form more consistent with the theory on stochastic differential equations. We have that equation (3.2) contains a random and a deterministic part. This encourages us to write (3.2) in the following stochastic differential form.

$$du^k = (\alpha\,(A\mathbf{u} - \mathbf{f}))^k dt + (\beta\,(A\mathbf{u} - \mathbf{f}))^k d\mathcal{W}^k$$

Where:

- $\alpha(t, \mathbf{u})$ denotes the deterministic part in the right hand side of equation (3.2).
- $\beta(t, \mathbf{u})$ denotes its stochastic part.
- $k \in \{1, \cdots, n\}$, denotes the k-th component in the system of ODE's.
- $d\mathcal{W}^k$ is an infinitesimal Wiener increment, since we need to integrate the stochastic part with respect to a Wiener process.

The reason we take the k-th component and not the entire matrix form, is to show that we will have to integrate each component along a different independent Wiener process. (In a matrix form we can take $d\mathcal{W}$ to be a square diagonal matrix with i.i.d. Wiener increments on the diagonal, which will be done in section 3.3.2.)

We can solve this system of stochastic ODE's using a multidimensional Euler-Maruyama time-integration method. Since we are planning to solve the problem both explicitly (forward) and implicitly (backward), we give both discretisations for the k-th component:

Forward:

$$u_{j+1}^k = u_j^k + \alpha \left(A\mathbf{u}_j - \mathbf{f}\right)^k \Delta t + \beta \left(A\mathbf{u}_j - \mathbf{f}\right)^k \Delta \mathcal{W}_j^k$$

Backward:

$$\begin{aligned} u_j^k &= u_{j-1}^k + \alpha \left(A\mathbf{u}_j - \mathbf{f}\right)^k \Delta t + \beta \left(A\mathbf{u}_{j-1} - \mathbf{f}\right)^k \Delta \mathcal{W}_j^k \qquad \text{or, equivalently:} \\ u_{j+1}^k &= u_j^k + \alpha \left(A\mathbf{u}_{j+1} - \mathbf{f}\right)^k \Delta t + \beta \left(A\mathbf{u}_j - \mathbf{f}\right)^k \Delta \mathcal{W}_j^k \end{aligned}$$

Where:

- $\Delta t = T/s$, for some end-time T and number of time-steps s.
- $j \in \{0, \dots, s\}$ denotes a single time-step. We can write: $t_j = jdt$.
- $\Delta \mathcal{W}_j^k \sim N(0, \Delta t)$ is the increment of a Wiener process on the interval $[t_j, t_{j+1}]$. There are some considerations on the nature of this increment. These will be discussed in section 5.2.

3.1.1 Numerical Stability: Semi- vs. Fully Implicit Methods

The implicit Euler scheme given above is not *fully* implicit. In the stochastic term we do not take coefficients from the $(j+1)$-th step, but from the j-th step. We could use a fully implicit Euler scheme (p.336 of [3]):

$$u_{j+1}^k = u_j^k + \alpha(\mathbf{u}_{j+1})^k \Delta t + \beta(\mathbf{u}_{j+1})^k \Delta \mathcal{W}_j$$

We test the fully implicit method with a single standard stochastic differential equation:

$$dX = aXdt + bXd\mathcal{W}$$

Its discretisation becomes:

$$\begin{aligned} X_{j+1} &= X_j + aX_{j+1}\Delta t + bX_{j+1}\Delta \mathcal{W}_j \quad \text{or:} \\ X_{j+1} &= \left(\frac{1}{1 - a\Delta t - b\Delta \mathcal{W}_j}\right) X_j \end{aligned}$$

This last expression defines a sequence which we say is *mean square stable* [2] if $\lim_{j\to\infty} \mathrm{E}\left[|X_j|^2\right] = 0$, where $\mathrm{E}\left[\cdot\right]$ denotes the expected value. We have that:

$$\mathrm{E}\left[|X_{j+1}|^2\right] = \mathrm{E}\left[\frac{1}{|1 - a\Delta t - b\Delta \mathcal{W}_j|^2} |X_j|^2\right]$$

For this equation to become zero as j gets large we need that:

$$\frac{1}{|1 - a\Delta t - b\Delta \mathcal{W}_j|} < 1$$

This is difficult due to the random term in the denominator. It is possible for the denominator to become really small, resulting in divergence. Such a stability issue can be solved by using a bounded increment $\Delta \widetilde{\mathcal{W}}_j$ (see p. 337 of [3] and [2]), which reduces the increment to a two-point random variable such that:

$$P(\Delta \widetilde{\mathcal{W}}_j = \sqrt{\Delta t}) = P(\Delta \widetilde{\mathcal{W}}_j = -\sqrt{\Delta t}) = \frac{1}{2}$$

We are, however, not interested in this approach since it neglects the Wiener process along which we want to integrate. Furthermore, it stabilises the fully implicit method only in the *asymptotic* sense, which is a weaker notion than the mean square one [1].

We can do a similar analysis with the semi-implicit method:

$$u_{j+1}^k = u_j^k + \alpha(\mathbf{u}_{j+1})^k \Delta t + \beta(\mathbf{u}_j)^k \Delta \mathcal{W}_j$$

The test equation becomes:

$$X_{j+1} = \left(\frac{1 + b\Delta \mathcal{W}_j}{1 - a\Delta t} \right) X_j$$

In [6] it is shown that for this sequence:

$$\lim_{j\to\infty} \mathrm{E}\left[|X_{j+1}|^2\right] = 0 \iff \frac{1 + |b|^2}{|1 - a\Delta t|^2} < 1$$

Hence, by choosing our coefficients wisely, we can obtain an implicit numerical scheme which is stable in the mean square sense. A similar statement holds for the explicit Euler Maruyama method applied to the test equation:

$$\lim_{j\to\infty} \mathrm{E}\left[|X_{j+1}|^2\right] = 0 \iff |1 + a\Delta t|^2 + |b|^2 < 1$$

Thus, in this project we will make use of a semi-implicit and an explicit Euler-Maryama scheme. A more thorough stability analysis of the methods discussed above can be found in [1, 2, 3, 6]

We have given the numerical scheme for solving problem (3.2) and justified the choice of our methods. Now we shall elaborate on the different components of (3.2) and derive several equations necessary to solve the problem.

3.2 Further Development of the Stochastic Diffusion Equation

In equation (3.2) we have $\mathbf{u}(t) \in \mathbb{R}^n$. This term consists of a deterministic and a stochastic part, in which the stochastic part is governed by an orthogonal matrix. Similar to [7], we will write this as $\mathbf{u} = \bar{\mathbf{u}} + V\mathbf{y}$, where $V \in \mathbb{R}^{n\times m}$ is an orthogonal matrix and $\mathbf{y} \in \mathbb{R}^m$ is a vector containing m normally distributed random variables with expectation $E[y_i] = 0$, which implies that $E[\mathbf{y}] = 0$. For the orthogonal matrix V we will also require that $V^T\left(\frac{d}{dt}V\right) = 0$. As indicated before, by using this approach we are able to approximate a situation containing stochastic information with a relatively small amount of random variables. We will do the same for the stochastic forcing.

Let $\mathbf{f}$ consist of a deterministic and a stochastic part governed by a matrix W: $\boldsymbol{f} = \mathbf{b} + W\mathbf{z}$, where $W \in \mathbb{R}^{n\times m}$ and $\mathbf{z} \in \mathbb{R}^m$ is a stochastic process in m dimensions, with expectation $E[\mathbf{z}] = 0$. The columns of W will determine the regions where the random variables in $\mathbf{z}$ will act, approximating a situation where there would be independent stochastic input at every position. Now we have three time-dependent components, $\bar{\mathbf{u}}$, $\mathbf{y}$ and V, for which we need three equations.

We substitute the expressions for $\mathbf{u}$ and $\mathbf{f}$ in equation (3.2), which gives us the following equation:

$$\frac{d}{dt}\bar{\mathbf{u}} + \left(\frac{d}{dt}V\right)\mathbf{y} + V\frac{d}{dt}\mathbf{y} = A\bar{\mathbf{u}} - \mathbf{b} + AV\mathbf{y} - W\mathbf{z} \tag{3.3}$$

We have that $E[\mathbf{y}] = 0$, $E[\mathbf{z}] = 0$ and A,V,W,$\bar{\mathbf{u}}$ and $\mathbf{b}$ are deterministic. Taking the expectation of both sides of this equation gives us:

$$\mathrm{E}\left[\frac{d}{dt}\bar{\mathbf{u}}\right] + \mathrm{E}\left[\left(\frac{d}{dt}V\right)\mathbf{y}\right] + \mathrm{E}\left[V\frac{d}{dt}\mathbf{y}\right] = \mathrm{E}\left[A\bar{\mathbf{u}}\right] - \mathrm{E}\left[\mathbf{b}\right] + \mathrm{E}\left[AV\mathbf{y}\right] - \mathrm{E}\left[W\mathbf{z}\right] \quad \Rightarrow$$

$$\frac{d}{dt}\bar{\mathbf{u}} = A\bar{\mathbf{u}} - \mathbf{b} \tag{3.4}$$

Now we have isolated the deterministic part of the problem. Equation (3.4) does not contain any stochastic elements, which means that for this equation we can use an ordinary Euler scheme.

We can simplify equation (3.3) by subtracting equation (3.4) from (3.3). This gives us:

$$\left(\frac{d}{dt}V\right)\mathbf{y} + V\frac{d}{dt}\mathbf{y} = AV\mathbf{y} - W\mathbf{z} \tag{3.5}$$

Recall that V is an orthogonal matrix. This enables us to further simplify equation (3.5), by premultiplying both sides of the equation with the transpose of V:

$$V^T\left(\frac{d}{dt}V\right)\mathbf{y} + V^TV\frac{d}{dt}\mathbf{y} = V^TAV\mathbf{y} - V^TW\mathbf{z}$$

We have that $V^TV = I$ and we will require that $V^T\left(\frac{d}{dt}V\right) = 0$. This leads to the following simplification of equation (3.5):

$$\frac{d}{dt}\mathbf{y} = V^TAV\mathbf{y} - V^TW\mathbf{z} \tag{3.6}$$

This is the second equation necessary to solve problem (3.2). Here we will need an Euler-Maruyama scheme. The solution $\mathbf{y}$ is initially purely random, but depending on the forcing W and the diffusion A this will change. Having little forcing and a high diffusion should result in $\mathbf{y}$ getting more deterministic in time.

Now we only need to find an equation for the matrix V. We start by postmultiplying equation (3.5) with $\mathbf{y}^T$. We get:

$$\left(\frac{d}{dt}V\right)\mathbf{y}\mathbf{y}^T + V\left(\frac{d}{dt}\mathbf{y}\right)\mathbf{y}^T = AV\mathbf{y}\mathbf{y}^T - W\mathbf{z}\mathbf{y}^T$$

Like before, we take the expectation of both sides of the equation. By setting $C := E\left[\mathbf{y}\mathbf{y}^T\right]$ we get:

$$\left(\frac{d}{dt}V\right)C + VE\left[\left(\frac{d}{dt}\mathbf{y}\right)\mathbf{y}^T\right] = AVC - WE\left[\mathbf{z}\mathbf{y}^T\right] \tag{3.7}$$

Equation (3.6) gives us an an expression for $\frac{d}{dt}\mathbf{y}$. After substituting this expression in equation (3.7) we get:

$$\left(\frac{d}{dt}V\right)C + VV^TAVC + V^TWE\left[\mathbf{z}\mathbf{y}^T\right] = AVC - WE\left[\mathbf{z}\mathbf{y}^T\right]$$

Or, after some rearrangements:

$$\frac{d}{dt}V = \left(I - VV^T\right)\left(AV - WE\left[\mathbf{z}\mathbf{y}^T\right]C^{-1}\right) \tag{3.8}$$

This is the third equation necessary to solve problem (3.2). Note that this one is, like equation (3.4), entirely deterministic. So we can solve this using an ordinary implicit Euler scheme. However, equation (3.8) is non-linear. This means we have to apply an approximation. This will be explained in the next section, along with the specific implementations of the other equations.

3.3 Implementations using an Euler Scheme

We have seen that solving problem (3.2) requires us to solve equations (3.8), (3.6) and (3.4) in that order, since (3.6) depends on V, which is obtained in (3.8). For equation (3.4) the order is unimportant, since it is entirely independent of the other equations (this will change in section 4). In this section we will treat the discretisations given by the Euler-Maruyama scheme.

First we take a look at equation (3.8). To simplify things we let:

$$F(V) := \left(AV - WE\left[\mathbf{z}\mathbf{y}^T\right]C^{-1}\right)$$

Equation (3.8) becomes:

$$\frac{d}{dt}V = \left(I - VV^T\right)F(V)$$

To solve this equation numerically we use, as discussed in section 3.1, an Euler scheme. The explicit discretisation is fairly straightforward:

$$\begin{aligned} \frac{V_{j+1} - V_j}{\Delta t} &= \left(I - V_jV_j^T\right)F(V_j) \quad \Rightarrow \\ V_{j+1} &= V_j + \left(I - V_jV_j^T\right)F(V_j)\Delta t \end{aligned} \tag{3.9}$$

Where:

- $\Delta t = T/s$, for some end-time T and number of time-steps s.
- $j \in \{0, \dots, s\}$, $t = jdt$.

This is the necessary expression for implementing (3.8), since the unknown $(j+1)$-th step can be directly computed from the known j-th step. The implicit discretisation is given by:

$$\frac{V_{j+1} - V_j}{\Delta t} = \left(I - V_{j+1}V_{j+1}^T\right)F(V_{j+1}) \tag{3.10}$$

This poses a problem, since we cannot simply write this equation in terms of V_{j+1}. A solution is to approximate V_{j+1}, by cleverly rewriting equation (3.10).

First we define $\delta V := V_{j+1} - V_j$, which denotes a small time-variation of V. Then we have that $V_{j+1} = V_j + \delta V$. Substituting this into the right hand side of equation 3.10 gives us:

$$\begin{aligned} \frac{V_{j+1} - V_j}{\Delta t} &= \left(I - (V_j + \delta V)(V_j + \delta V)^T\right)F(V_j + \delta V) \\ &= \left(I - \left(V_jV_j^T + \delta V V_j^T + V_j\delta V^T + \delta V\delta V^T\right)\right)F(V_j + \delta V) \\ &\approx \left(I - V_jV_j^T\right)F(V_j + \delta V) \quad \text{(since } \delta V \text{ is small)} \end{aligned} \tag{3.11}$$

We have that:

$$\begin{aligned} F(V_j + \delta V) &= AV_j + A\delta V - W\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]C^{-1} \\ &= F(V_j) + A\delta V \end{aligned}$$

Substituting this expression in (3.11) gives us a usable discretisation for equation (3.8):

$$\begin{aligned} \frac{V_{j+1} - V_j}{\Delta t} &= \left(I - V_jV_j^T\right)\left(F(V_j) + A\delta V\right) \quad \text{or:} \\ \delta V &= \Delta t\left(I - V_jV_j^T\right)\left(F(V_j) + A\delta V\right) \quad \Rightarrow \\ \delta V - \Delta t\left(I - V_jV_j^T\right)A\delta V &= \Delta t\left(I - V_jV_j^T\right)F(V_j) \qquad \Rightarrow \\ \left(I - \Delta t\left(I - V_jV_j^T\right)A\right)\delta V &= \Delta t\left(I - V_jV_j^T\right)F(V_j) \end{aligned} \tag{3.12}$$

This is an expression of the form $A\mathbf{x} = \mathbf{b}$ which we can solve using, for example, an LU decomposition [5]. We can also add a constraint to this system. If we premultiply equation (3.12) with V_j^T we get:

$$\begin{aligned} \left(V_j^T - \Delta t \left(V_j^T - V_j^T V_j V_j^T\right) A\right) \delta V &= \Delta t \left(V_j^T - V_j^T V_j V_j^T\right) F(V_j) \\ \Rightarrow \quad V_j^T \delta V &= 0 \end{aligned}$$

Equation (3.12) combined with this added constraint can be handled by viewing (3.12) as an optimisation problem and using the method of Lagrange multipliers [9]. First we introduce a Lagrange multiplier γ. Using γ we can bring the constraint inside equation (3.12). Then, minimising over δV and γ is equivalent to solving the following system:

$$\left(I - \Delta t \left(I - V_j V_j^T\right) A\right) \delta V + V_j \gamma = \Delta t \left(I - V_j V_j^T\right) F(V_j) \tag{3.13a}$$

$$V_j^T \delta V = 0 \tag{3.13b}$$

We can further simplify equation (3.13a), by taking $I = I - V_j V_j^T + V_j V_j^T \quad \Rightarrow$

$$\left(I - V_j V_j^T + V_j V_j^T - \Delta t \left(I - V_j V_j^T\right) A\right) \delta V + V_j \gamma = \Delta t \left(I - V_j V_j^T\right) F(V_j)$$

And since $V_j(V_j^T \delta V) = 0$, we have:

$$\begin{gathered} \left(I - V_j V_j^T - \Delta t \left(I - V_j V_j^T\right) A\right) \delta V + V_j \gamma = \Delta t \left(I - V_j V_j^T\right) F(V_j) \quad \text{or} \\ \left(I - V_j V_j^T\right) (I - \Delta t A) \delta V + V_j \gamma = \Delta t \left(I - V_j V_j^T\right) F(V_j) \end{gathered}$$

Here $\left(I - V_j V_j^T\right)(I - \Delta t A)\delta V$ is equivalent to $(I - \Delta t A)\delta V + \zeta(V_j)$, for some term $\zeta(\cdot)$ depending on V_j. We already have such a term and so we let $\zeta(V_j) = V_j \gamma$. This greatly simplifies our system of equations. We will write the result in matrix form:

$$\begin{bmatrix} (I - \Delta t A) & V_j \\ V_j^T & 0 \end{bmatrix} \begin{bmatrix} \delta V \\ \gamma \end{bmatrix} = \begin{bmatrix} \Delta t \left(I - V_j V_j^T\right) F(V_j) \\ 0 \end{bmatrix} \tag{3.14}$$

Where 0 denotes an $m \times m$ null matrix. This will be our system dealing with solving (3.8) implicitly.

3.3.1 Obtaining the Expected Values

Note that in equation (3.8) we have an expected value: $\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]$. In the implementation of (3.14) we will not obtain this expected value analytically, since we then need to know the unknown probability distributions in advance[1]. Therefore we will approximate these values by using the law of large numbers and thus computing the mean over a large amount of trials, that is: $\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right] \approx \frac{1}{K}\sum_{i=1}^{K} \mathbf{z}_i \mathbf{y}_i^T$, for large K.

3.3.2 Implementing the Euler-Maruyama Method

Let us now turn to equation (3.6)

$$\frac{d}{dt}\mathbf{y} = V^T A V \mathbf{y} - V^T W \mathbf{z} \tag{3.6}$$

[1]Finding the probability distributions is possible using the Fokker-Planck equation, but this would dramatically increase the computational costs.

This equation contains a stochastic process $\mathbf{z}$ and has some stochastic initial condition. Whether the solution stays purely stochastic will depend on the nature of A and W. The Euler-Maruyama method gives us the following explicit discretisation:

$$y_{j+1}^{k} = y_j^k + \left(V_j^T A V_j \mathbf{y}_j\right)^k \Delta t - \left(V_j^T W \mathbf{z}\right)^k \Delta \mathcal{W}_j^k$$

Where, again, $j \in \{0, \ldots, s\}$ denotes a single time-step, $k \in \{1, \cdots, m\}$ denotes the k-th component in the system, Δt is the step-size and $\Delta \mathcal{W}_j^k$ is an increment in a Wiener process on the interval $[t_j, t_{j+1}]$. For ease of use we want this system in a matrix form. This is easily done if we create a diagonal matrix $\mathcal{D}$, containing the i.i.d. random variables $\Delta \mathcal{W}_j^k \sim N(0, \Delta t)$ on its diagonal. The system becomes:

$$\mathbf{y}_{j+1} = \mathbf{y}_j + V_j^T A V_j \mathbf{y}_j \Delta t - V_j^T W \mathcal{D} \mathbf{z} \tag{3.15}$$

This expression is ready to be implemented.

Obtaining a workable implicit discretisation is, luckily, not as much work as in the previous section. The semi-implicit Euler-Maruyama scheme gives us:

$$\mathbf{y}_{j+1} = \mathbf{y}_j + V_{j+1}^T A V_{j+1} \mathbf{y}_{j+1} \Delta t - V_j^T W \mathcal{D} \mathbf{z}$$

Note that the step V_{j+1} is necessary to solve equation (3.6) implicitly. We do not take implicit coefficients at the random term for reasons discussed in section 3.1. As in the explicit case, $\mathcal{D}$ is a diagonal matrix containing independent identically distributed random variables with mean 0 and variance Δt. Rearranging this expression gives us:

$$\left(I - V_{j+1}^T A V_{j+1} \Delta t\right) \mathbf{y}_{j+1} = \mathbf{y}_j - V_j^T W \mathcal{D} \mathbf{z} \tag{3.16}$$

This system has the form $Ax = b$ and is therefore ready to be implemented. This leaves us with equation (3.4), which can be solved independent of (3.6) and (3.8).

3.3.3 Implementing the Euler Method

Consider equation (3.4):

$$\frac{d}{dt}\bar{\mathbf{u}} = A\bar{\mathbf{u}} - \mathbf{b} \tag{3.4}$$

The forward Euler method gives us the following explicit discretisation:

$$\bar{\mathbf{u}}_{j+1} = \bar{\mathbf{u}}_j + \Delta t \left(A\bar{\mathbf{u}}_j - \mathbf{b}\right) \tag{3.17}$$

Which is ready to be implemented. The implicit discretisation needs little work as well:

$$\begin{aligned} \bar{\mathbf{u}}_{j+1} &= \bar{\mathbf{u}}_j + \Delta t \left(A\bar{\mathbf{u}}_{j+1} - \mathbf{b}\right) \\ \Rightarrow \quad (I - \Delta t A)\, \bar{\mathbf{u}}_{j+1} &= \bar{\mathbf{u}}_j - \Delta t \mathbf{b} \end{aligned} \tag{3.18}$$

Which is in the form $Ax = b$. Now we are ready to solve problem (3.2) by implementing equations (3.14), (3.16) and (3.18) respectively. Technical details and the `MATLAB` code are available in appendix A.5.

4 A Non-Linear Stochastic Reaction-Diffusion Equation

Here we derive a method to solve a non-linear variant of the problem given in section 3. The steps taken are therefore very similar to the previous ones. Consider equation (3.1). We add a bilinear form $\langle u, u\rangle$, which gives us the following non-linear problem:

$$\frac{\partial}{\partial t}u = a\frac{\partial^2 u}{\partial x^2} + \langle u, u\rangle - f \tag{4.1}$$

Physically, the bilinear form may be regarded as a reaction. Then this equation resembles a reaction-diffusion system with a partially stochastic initial state and forcing. In this section we will not choose a specific bilinear form, since we want the developed method to be somewhat generally applicable.

Similar to section 3.1 we get rid of the spatial derivatives using the finite difference method. Again, we obtain a system of stochastic ODE's in matrix form:

$$\frac{d}{dt}\mathbf{u} = A\mathbf{u} + \langle \mathbf{u}, \mathbf{u}\rangle - \mathbf{f} \tag{4.2}$$

As in section 3.2 we have that $\mathbf{u} = \bar{\mathbf{u}} + V\mathbf{y}$ and $\mathbf{f} = \mathbf{b} + W\mathbf{z}$, where $\mathbf{y}, \mathbf{z} \in \mathbb{R}^m$, containing normally distributed random variables with expected value $\mathrm{E}[y_i] = \mathrm{E}[z_i] = 0$, $i \in \{1, \ldots, m\}$ and $V, W \in \mathbb{R}^{n\times m}$ with V orthogonal. Substituting these expressions in (4.2) we get:

$$\begin{aligned}\frac{d}{dt}\bar{\mathbf{u}} + \left(\frac{d}{dt}V\right)\mathbf{y} + V\frac{d}{dt}\mathbf{y} \quad &= \quad A\bar{\mathbf{u}} + AV\mathbf{y} + \langle \bar{\mathbf{u}}, \bar{\mathbf{u}}\rangle + \langle V\mathbf{y}, \bar{\mathbf{u}}\rangle \\ &\quad + \langle \bar{\mathbf{u}}, V\mathbf{y}\rangle + \langle V\mathbf{y}, V\mathbf{y}\rangle - \mathbf{b} - W\mathbf{z}\end{aligned} \tag{4.3}$$

Now we take the expected value on both sides of equation (4.3) to find an expression for the deterministic part $\bar{\mathbf{u}}$ of $\mathbf{u}$:

$$\frac{d}{dt}\bar{\mathbf{u}} = A\bar{\mathbf{u}} + \langle \bar{\mathbf{u}}, \bar{\mathbf{u}}\rangle + \mathrm{E}[\langle V\mathbf{y}, \bar{\mathbf{u}}\rangle] + \mathrm{E}[\langle \bar{\mathbf{u}}, V\mathbf{y}\rangle] + \mathrm{E}[\langle V\mathbf{y}, V\mathbf{y}\rangle] - \mathbf{b} \tag{4.4}$$

We can get rid of a couple of the bilinear forms. Let $f(\mathbf{y})$ be the multidimensional probability density function of the random variable $\mathbf{y}$. Since we can take an integral to the inside of a bilinear form we have that:

$$\begin{aligned}\mathrm{E}[\langle V\mathbf{y}, \bar{\mathbf{u}}\rangle] &= \int_{-\infty}^{\infty} \langle V\mathbf{y}, \bar{\mathbf{u}}\rangle\, f(\mathbf{y})d\mathbf{y} \\ &= \left\langle \int_{-\infty}^{\infty} V\mathbf{y}f(\mathbf{y})d\mathbf{y}, \bar{\mathbf{u}}\right\rangle \\ &= \langle V\mathrm{E}[\mathbf{y}], \bar{\mathbf{u}}\rangle = 0\end{aligned}$$

The same holds for $\langle \bar{\mathbf{u}}, V\mathbf{y}\rangle$. Thus we get rid of these terms in (4.4). Our equation for the deterministic part becomes:

$$\frac{d}{dt}\bar{\mathbf{u}} = A\bar{\mathbf{u}} + \langle \bar{\mathbf{u}}, \bar{\mathbf{u}}\rangle + \mathrm{E}[\langle V\mathbf{y}, V\mathbf{y}\rangle] - \mathbf{b} \tag{4.5}$$

In the next section we will discuss the implementation of a forward and a backward discretisation of this equation. Now we will find an equation for $\mathbf{y}$. Subtracting (4.5) from (4.3) gives us:

$$\begin{aligned}\left(\frac{d}{dt}V\right)\mathbf{y} + V\frac{d}{dt}\mathbf{y} \quad &= \quad AV\mathbf{y} + \langle V\mathbf{y}, \bar{\mathbf{u}}\rangle + \langle \bar{\mathbf{u}}, V\mathbf{y}\rangle + \langle V\mathbf{y}, V\mathbf{y}\rangle \\ &\quad - \mathrm{E}[\langle V\mathbf{y}, V\mathbf{y}\rangle] - W\mathbf{z}\end{aligned} \tag{4.6}$$

Premultiplying with V^T gives us an equation for $\mathbf{y}$:

$$\begin{aligned}\frac{d}{dt}\mathbf{y} &= V^T AV\mathbf{y} + V^T\big(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle \\ &\quad + \langle V\mathbf{y},V\mathbf{y}\rangle - \mathrm{E}\left[\langle V\mathbf{y},V\mathbf{y}\rangle\right]\big) - V^T W\mathbf{z} \qquad (4.7)\end{aligned}$$

We are left with finding an expression for V. Postmultiplying equation (4.6) with $\mathbf{y}^T$ gives us:

$$\begin{aligned}\left(\frac{d}{dt}V\right)\mathbf{y}\mathbf{y}^T + V\left(\frac{d}{dt}\mathbf{y}\right)\mathbf{y}^T &= AV\mathbf{y}\mathbf{y}^T + \big(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle + \langle V\mathbf{y},V\mathbf{y}\rangle \\ &\quad - \mathrm{E}\left[\langle V\mathbf{y},V\mathbf{y}\rangle\right]\big)\mathbf{y}^T - W\mathbf{z}\mathbf{y}^T\end{aligned}$$

We take the expected value on both sides of this equation and define $C := \mathrm{E}\left[\mathbf{y}\mathbf{y}^T\right]$.

$$\begin{aligned}\left(\frac{d}{dt}V\right)C + V\mathrm{E}\left[\left(\frac{d}{dt}\mathbf{y}\right)\mathbf{y}^T\right] &= AVC + E\big[\big(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle + \langle V\mathbf{y},V\mathbf{y}\rangle \\ &\quad - \mathrm{E}\left[\langle V\mathbf{y},V\mathbf{y}\rangle\right]\big)\mathbf{y}^T\big] - W\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]\end{aligned}$$

And since $\mathrm{E}\left[\mathrm{E}\left[\langle V\mathbf{y},V\mathbf{y}\rangle\right]\mathbf{y}^T\right] = \mathrm{E}\left[\langle V\mathbf{y},V\mathbf{y}\rangle\right]\mathrm{E}\left[\mathbf{y}^T\right] = 0$, we have:

$$\begin{aligned}\left(\frac{d}{dt}V\right)C + V\mathrm{E}\left[\left(\frac{d}{dt}\mathbf{y}\right)\mathbf{y}^T\right] &= AVC + E\big[\big(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle + \langle V\mathbf{y},V\mathbf{y}\rangle\big)\mathbf{y}^T\big] \\ &\quad - W\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]\end{aligned}$$

Now we can substitute equation (4.7) for $\frac{d}{dt}\mathbf{y}$ in the left hand side of this expression and after some manipulation we get:

$$\left(\frac{d}{dt}V\right) = \left(I - VV^T\right)\left(AVC + \mathrm{E}\left[\left(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle + \langle V\mathbf{y},V\mathbf{y}\rangle\right)\mathbf{y}^T\right] - W\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]\right)C^{-1} \qquad (4.8)$$

If we let:

$$F(V) := \left(AVC + \mathrm{E}\left[\left(\langle V\mathbf{y},\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V\mathbf{y}\rangle + \langle V\mathbf{y},V\mathbf{y}\rangle\right)\mathbf{y}^T\right] - W\mathrm{E}\left[\mathbf{z}\mathbf{y}^T\right]\right)C^{-1}$$

We have:

$$\frac{d}{dt}V = \left(I - VV^T\right)F(V) \qquad (4.9)$$

Now we have expressions for $\bar{\mathbf{u}}, \mathbf{y}$, and V. We can numerically solve equation (4.9) in the same way as explained in section 3.3, since we again have that the Jacobian of F is A. Hence, equation (3.9) gives the explicit and equation (3.14) gives the implicit implementation of equation (4.9). Equations (4.7) and (4.5) are not that similar to their linear counterparts and need some further attention.

4.1 Implementing the Euler-Maruyama Method

The Euler-Maruyama scheme gives us the following explicit and semi-implicit discretisations for the k-th component of equation (4.7):

Explicit:

$$\begin{aligned}y^k_{j+1} &= y^k_j + \Delta t\, V_j^T\Big(AV_j\mathbf{y}_j + \langle V_j\mathbf{y}_j,\bar{\mathbf{u}}\rangle + \langle\bar{\mathbf{u}},V_j\mathbf{y}_j\rangle \\ &\quad + \langle V_j\mathbf{y}_j, V\mathbf{y}_j\rangle - \mathrm{E}\left[\langle V_j\mathbf{y}_j,V_j\mathbf{y}_j\rangle\right]\Big)^k - \left(V_j^T W\mathbf{z}\right)^k\Delta\mathcal{W}^k_j \qquad (4.10)\end{aligned}$$

Implicit:

$$\begin{aligned} y_{j+1}^k &= y_j^k + \Delta t\, V_{j+1}^T \Big(A V_{j+1} \mathbf{y}_{j+1} + \langle V_{j+1}\mathbf{y}_{j+1}, \bar{\mathbf{u}} \rangle + \langle \bar{\mathbf{u}}, V_{j+1}\mathbf{y}_{j+1} \rangle \\ &\quad + \langle V_{j+1}\mathbf{y}_{j+1}, V\mathbf{y}_{j+1} \rangle - \mathrm{E}\left[\langle V_{j+1}\mathbf{y}_{j+1}, V_{j+1}\mathbf{y}_{j+1} \rangle \right] \Big)^k - \left(V_j^T W \mathbf{z} \right)^k \Delta \mathcal{W}_j^k \end{aligned} \tag{4.11}$$

We see that the explicit discretisation is ready to be implemented. In the implicit discretisation we have a higher order function, which we need to solve for y_{j+1}. Depending on the bilinear form this can become extremely difficult and even impossible to solve analytically. We will therefore transform the equation into a root finding problem and approximate this root. In the same way as in section 3.3.2, we write the system in matrix form. Then let us define the following function:

$$\begin{aligned} G(\mathbf{x}) :=\ & \mathbf{x} - \mathbf{y}_j - \Delta t\, V_{j+1}^T \Big(A V_{j+1}\mathbf{x} + \langle V_{j+1}\mathbf{x}, \bar{\mathbf{u}} \rangle + \langle \bar{\mathbf{u}}, V_{j+1}\mathbf{x} \rangle + \langle V_{j+1}\mathbf{x}, V_{j+1}\mathbf{x} \rangle \\ & -\ \mathrm{E}\left[\langle V_{j+1}\mathbf{x}, V_{j+1}\mathbf{x} \rangle \right] \Big) + V_j^T W \mathcal{D}\mathbf{z} \qquad (4.12) \\ G(\mathbf{x}) =\ & 0 \end{aligned}$$

One of the possibly several roots of this function is the required $\mathbf{y}_{j+1}$. We can use a multidimensional version of the Newton-Raphson algorithm (see section 9.6 of [5]) to approximate this specific root. Let $\mathbf{y}_j$ be an initial guess $\mathbf{x}_{\text{init}} \in \mathbb{R}^m$, then an approximation to $\mathbf{y}_{j+1}$ is found by solving the following system:

$$J(\mathbf{x}_{\text{init}})(\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{init}}) = -G(\mathbf{x}_{\text{init}}) \tag{4.13}$$

Where J is the Jacobian matrix of G: $J_{ij} = \frac{\partial G_i}{\partial x_j}$ and $\mathbf{x}_{\text{new}} \in \mathbb{R}^m$ is an approximation of a root of G close to $\mathbf{x}_{\text{init}} = \mathbf{y}_j$. We can iterate this process a few times to get a good approximation of $\mathbf{y}_{j+1}$. We will discuss a method to find the Jacobian numerically in section 4.1.1.

Due to the stochastic and non-linear nature of equation 4.12, subsequent solutions may vary greatly. This will cause the initial guess to be a possibly bad approximation to the sought after root. As a result the algorithm might overshoot and give a bad solution. To prevent this overshooting we will implement a safeguard by using *backtracking* (see section 9.7.1 of [5]). This method makes use of the fact that a solution to the Newton algorithm should satisfy $|G(\mathbf{x}_{\text{new}})| \leq |G(\mathbf{x}_{\text{init}})|$. If this inequality does not hold, the algorithm is repeated, but with an extra parameter λ:

$$J(\mathbf{x}_{\text{init}})(\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{init}}) = -\lambda G(\mathbf{x}_{\text{init}})$$

Where $0 < \lambda \leq 1$. Now if a solution seems to overshoot we restart the algorithm with a smaller value for λ. Since we have a negative sign in the left hand side it is possible to stabilise every solution with some possibly very small λ, but to prevent the algorithm from running indefinitely we shall restrict this parameter to: $\lambda \geq 0.05$. An algorithm performing the Newton-Raphson approximation with backtracking can be found in appendix A.3

4.1.1 Approximating a Jacobian Matrix

We can find the Jacobian analytically but, for compatibility reasons we will do this numerically. For this we approximate the partial derivatives of G in the following way:

Since $G : \mathbb{R}^m \to \mathbb{R}^m$, we write:

$$\frac{\partial G(\mathbf{x})}{\partial x_j} = \left[\frac{\partial G_1(\mathbf{x})}{\partial x_j}, \ldots, \frac{\partial G_m(\mathbf{x})}{\partial x_j} \right]^T \quad \text{for } j \in \{1, \ldots, m\}$$

We can approximate each derivative:

$$\begin{aligned}\frac{\partial G_i(\mathbf{x})}{\partial x_j} &= \lim_{\delta\to 0}\frac{G_i(x_1,\dots,x_j+\delta,\dots,x_m)-G_i(x_1,\dots,x_m)}{\delta}\\ &\approx \frac{G_i(x_1,\dots,x_j+\delta,\dots,x_m)-G_i(x_1,\dots,x_m)}{\delta}\quad\text{for small }\delta\end{aligned}$$

Now we obtain the Jacobian by assembling the vectors with approximated derivatives into an $m\times m$-matrix:

$$J(\mathbf{x})=\left[\frac{\partial G(\mathbf{x})}{\partial x_1},\dots,\frac{\partial G(\mathbf{x})}{\partial x_m}\right]$$

Hence, by using some very small δ we can approximate the Jacobian of G. An algorithm performing this process is given in appendix A.2. Using this method we can find a good approximation of $\mathbf{y}_{j+1}$. We now turn to the last equation to be implemented.

4.1.2 Implementing the Euler Method

We are left with equation (4.5), which, like (3.4), does not contain any stochastic part, hence we can use an ordinary Euler scheme. Let us first give the explicit discretisation:

$$\bar{\mathbf{u}}_{j+1}=\bar{\mathbf{u}}_j+\Delta t\big(A\bar{\mathbf{u}}_j+\langle\bar{\mathbf{u}}_j,\bar{\mathbf{u}}_j\rangle+\mathrm{E}\left[\langle V_j\mathbf{y}_j,V\mathbf{y}_j\rangle\right]-\mathbf{b}\big)$$

This form is clearly ready to be implemented. For the implicit discretisation we have:

$$\bar{\mathbf{u}}_{j+1}=\bar{\mathbf{u}}_j+\Delta t\big(A\bar{\mathbf{u}}_{j+1}+\langle\bar{\mathbf{u}}_{j+1},\bar{\mathbf{u}}_{j+1}\rangle+\mathrm{E}\left[\langle V_{j+1}\mathbf{y}_{j+1},V\mathbf{y}_{j+1}\rangle\right]-\mathbf{b}\big)$$

Again, we cannot simply solve for $\mathbf{u}_{j+1}$, but have to approximate this term using the Newton-Raphson method. Consider the following function:

$$\begin{aligned}H\left(\mathbf{x}\right) &:= \mathbf{x}-\bar{\mathbf{u}}_j-\Delta t\big(A\mathbf{x}+\langle\mathbf{x},\mathbf{x}\rangle+\mathrm{E}\left[\langle V_{n+1}\mathbf{y}_{n+1},V\mathbf{y}_{n+1}\rangle\right]-\mathbf{b}\big)\\ H\left(\mathbf{x}\right) &= 0\end{aligned}$$

If we now let $\bar{\mathbf{u}}_j$ be an initial guess $\mathbf{x}_{\text{init}}$, then we can obtain an approximation $\mathbf{x}_{\text{new}}$ of $\bar{\mathbf{u}}_{j+1}$ by solving the following system similar to (4.13):

$$J(\mathbf{x}_{\text{init}})(\mathbf{x}_{\text{new}}-\mathbf{x}_{\text{init}})=-H(\mathbf{x}_{\text{init}})\tag{4.14}$$

Where J is the Jacobian matrix of H. Again, we need to iterate this process a few times to get a good approximation of $\mathbf{y}_{j+1}$. Now we have equations (4.9), (4.13) and (4.14), which, if we consider the dependencies, we need to implement in that order to solve the problem, equation (4.2). The `MATLAB` implementation is available in appendix A.5.

4.2 Two Dimensions

Of course we want to take the method developed in sections 3 and 4 to a higher dimension. Luckily this involves only a few minor changes. In this section we will discuss some adaptions for the two-dimensional case. Consider equation (4.1):

$$\frac{\partial}{\partial t}u=a\Delta u+\langle u,u\rangle-f\tag{4.1}$$

We now have that $\Delta=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}$. The solution $u(x,y,t)$ is defined on a two-dimensional domain with some initial condition $u(x,y,0)=\phi(x,y)$. On the boundary we will have one or several

Dirichlet conditions.

The Laplacian yields the following finite difference scheme:

$$(\Delta u)_{i,j} = \frac{u_{i+1,j}(t) + u_{i,j+1}(t) - 4u_{i,j}(t) + u_{i-1,j}(t) + u_{i,j-1}(t)}{h^2} + \mathcal{O}(h^2) \tag{4.15}$$

Assuming we have a uniform, rectangular grid such that i and j denote grid points in the x- and y-direction respectively. This discretisation transforms the PDE into a system of ODE's and we would like to get this system into a matrix form. Therefore we need to reshape the solution $u(x, y, t)$, which is initially defined on a two-dimensional grid, into a vector $\mathbf{u} \in \mathbb{R}^{n^2}$:

$$\mathbf{u} = [u_{1,1}, u_{2,1}, \ldots, u_{n,1}, u_{1,2}, u_{2,2}, \ldots, u_{n,2}, \ldots, u_{1,n}, u_{2,n}, \ldots, u_{n,n}]^T$$

We can create a matrix $A \in \mathbb{R}^{n^2 \times n^2}$ such that the term $A\mathbf{u}$ meets the dependencies from equation (4.15). This discretised Laplacian has the following structure:

$$A = \frac{a}{h^2} \begin{bmatrix}
-4 & 1 & & & 1 & & & & & \\
1 & -4 & \ddots & & & 1 & & & & \\
 & \ddots & \ddots & 1 & & & \ddots & & & \\
 & & 1 & -4 & & & & \ddots & & \\
1 & & & & -4 & 1 & & & \ddots & \\
 & 1 & & & 1 & -4 & \ddots & & & 1 \\
 & & \ddots & & & \ddots & \ddots & \ddots & & \\
 & & & \ddots & & & \ddots & \ddots & \ddots & \\
 & & & & \ddots & & & \ddots & \ddots & 1 \\
 & & & & & 1 & & & 1 & -4
\end{bmatrix}$$

Now we have a suitable matrix A and vector $\mathbf{u}$, such that we can write the system in matrix form:

$$\frac{d}{dt}\mathbf{u} = A\mathbf{u} + \langle \mathbf{u}, \mathbf{u} \rangle - \mathbf{f} \tag{4.16}$$

Where $\mathbf{b}$ and δ are vectors in $\mathbb{R}^{n^2}$. Again we have that $\mathbf{u} = \bar{\mathbf{u}} + V\mathbf{y}$ and $\boldsymbol{\delta} = W\mathbf{z}$. But, now we have that $V, W \in \mathbb{R}^{n^2 \times m}$. Hence, these matrices will define regions of stochastic influence in the two-dimensional domain. From here we can continue using the method developed in section 4.

5 Preliminaries to the Analysis

In the previous sections we derived the necessary equations to numerically solve different stochastic diffusion and reaction-diffusion problems. In this section we will discuss the methods we use to analyse the results we will get from the numerical experiments.

5.1 Analysing the Stochastic Components

As previously explained, the solution of the SPDE contains a deterministic and a stochastic part: $\mathbf{u} = \bar{\mathbf{u}} + V\mathbf{y}$, where the stochastic part is a vector $\mathbf{y} \in \mathbb{R}^m$. This vector is initialised with m independent normally distributed random variables with some given mean and variance. In time the probability distributions of these random variables will become affected by the stochastic part of the forcing, which is a time-dependent stochastic process $z \in \mathbb{R}^m$, containing independent normally distributed random variables. At every time-step the vector z contains different random variables, independent of the other time-steps. The stochastic part of the solution will further be under the influence of the diffusion and, in the non-linear case, the term arising from the bilinear form. We can say that, due to all these influences, the initial mean and the variance of the random variables in $\mathbf{y}$ are bound to change.

To keep track of these changing parameters we will need to be able to estimate them. For this we will regard the computation of the vector $\mathbf{y}$ in the Euler-Maruyama method as a single trial in m variables. Hence, to be able to estimate the mean and the variance of each random variable in $\mathbf{y}$, we need to repeat the trials a considerable amount of times. Then we can obtain enough data to be able to accurately estimate the parameters describing the probability distributions.

Suppose we repeat a single time-step p times. We can store each result and create a matrix $G \in \mathbb{R}^{m \times p}$, containing p experiments in m variables which is called a *matrix of observations.* From each row we can easily estimate the first (crude) moment and the second (central) moment with the usual approximations using the law of large numbers. If we subtract the obtained mean from each row we bring the data in *mean deviation form.* Then we can easily compute a covariance matrix:

$$\mathrm{Cov}_G := \frac{1}{p-1} G G^T$$

This is a symmetric $m \times m$ matrix containing the variances of the variables on the diagonal and the covariances among them on the off-diagonals, for example:

$$\mathrm{Cov}_{G_{(1,2)}} = \frac{1}{p-1} G_{(1,:)} \cdot \left(G^T\right)_{(:,2)} = \frac{1}{p-1} G_{(1,:)} \cdot \left(G_{(2,:)}\right)^T$$

Which is an estimate of the covariance between the first and second random variable in $\mathbf{y}$. Here $G_{(1,:)}$ denotes the first row of G and $G_{(:,2)}$ its second column. Note that, with the fraction $1/(p-1)$, we take the unbiased estimator of the variance. Taking $1/p$ would give a biased estimator. The expected values of both estimators differ by a factor σ^2/p, where σ^2 denotes the true variance. We see that, when increasing the number of experiments, this difference will diminish.

At each time-step in the Euler-Maruyama scheme, the approach will thus be to repeat the computation of $\mathbf{y}$ a large amount of times. From the collected data we will be able to estimate the mean, variance and covariances in the components of $\mathbf{y}$. If we also closely monitor the structure of the orthogonal matrix V, acting on the random variables, we will be able to obtain detailed information on the stochastic part of the solution.

It is very likely that we find that the m variables in G become correlated when time progresses. When this happens it becomes difficult to tell which combination of random variables and columns of V are responsible for specific parts of the solution. To get a grip on a set of highly correlated data, there exists a method known as *principal component analysis*. A thorough explanation of this method can be found in [4] and [8]. The main idea is to find a set of basis vectors in which we can represent the matrix of observations, such that the correlations between the variables are minimised.

We will now briefly summarise the procedure. The principal components are a set of orthogonal basis vectors. We will express them as rows of a matrix P. The goal is to find a matrix P such that the change of basis $PG = Y$ results in a re-expressed matrix of observations Y with a diagonal covariance matrix Cov_Y, thus with zero covariances on the off-diagonals. Diagonalisation of the covariance matrix is performed by a similarity transformation with a matrix containing the eigenvectors of the matrix GG^T. By letting the rows of P contain the eigenvectors of GG^T, we have that $P^{-1} = P^T$ and since $PG = Y$:

$$\begin{aligned}\mathrm{Cov}_Y &= \frac{1}{p-1} YY^T \\ &= \frac{1}{p-1} PGG^TP^T \\ &= \frac{1}{p-1}\Lambda \quad \text{diagonal}\end{aligned}$$

Thus we find the principal components to be the eigenvectors of the covariance matrix times its transpose. This concludes the procedure of finding an appropriate basis. In the transformed matrix of observations we now have uncorrelated data. By estimating the variance we can indicate the actual influence of each transformed random variable on the solution. We are also able to perform the same change of basis on the matrix V, such that we find the re-expressed columns of V acting on the the uncorrelated random variables. Combining this information we will be able to exactly determine the cause of each random fluctuation in the solution.

In the implementation of the principal component analysis in appendix A.1, we use a slightly different, more sophisticated method of computing the principal components of the matrix of observations. If we let $H = G^T/\sqrt{p-1}$ such that $H^TH = \mathrm{Cov}_G$, then we compute the singular value decomposition of H, $H = U\Sigma V^T$, where the columns of V will contain the eigenvectors of $H^TH = \mathrm{Cov}_G$. Hence, choosing $P = V^T$ gives us the principal components of G.

5.2 Comparing an Implicit with an Explicit Scheme

A major subject of our interest is whether we can find any large differences between the explicit and the implicit implementation of each problem we previously discussed. In general an explicit Euler method is fast and easily implemented but unstable, whereas the implicit variant is very stable but has a high computational cost. Here we are interested in whether it is possible to achieve similar results, while using random variables and having to perform approximations in the implicit cases.

To compare the implicit with the explicit scheme we have to make sure that all conditions are maintained equal, especially the stochastic ones. Luckily, we are able to *seed* our random number generator such that in every experiment it is returning the exact same sequence of numbers. More important is to make sure that in both situations the Euler-Maruyama method integrates the same stochastic process along the same Wiener increments.

Of course we need the forward Euler method to have a really small time-step for it to be stable. The implicit method however, will not be suitable to run on the same time-interval with the same time-step. Because of its high computational cost this will take ages. This discrepancy in time-steps has repercussions for the Wiener path, which depends on the time-steps taken. Each increment in the Wiener process is a normally distributed random variable with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\Delta t}$. From this it seems that we cannot run both schemes with the same Wiener increments.

In [1] an approach is given that hints at a practical solution to this problem. Suppose we use a time-step of size Δt_{expl} in the explicit scheme. Let us then choose the larger time-step for the implicit scheme Δt_{impl} to be an integer multiple of the former one: $\Delta t_{\text{impl}} = k \cdot \Delta t_{\text{expl}}$. If we make sure that Δt_{impl} is an integer k multiple of Δt_{impl}, then, in the explicit scheme, we can halt the stochastic integration at every k-th iteration for $k-1$ time-steps. That is, we do not compute any new random variables for $k-1$ iterations. We only compute new random variables when the number of iterations has reached an integer multiple of k. With this approach the explicit scheme performs the same operation as the implicit one. To show this we compare the variance of the sum of k Wiener increments $\Delta\mathcal{W}_{\text{expl}} \sim N(0, \Delta t)$, with the variance of $\Delta\mathcal{W}_{\text{impl}} \sim N(0, \Delta t_{\text{impl}})$, which is $k \cdot \Delta t_{\text{expl}}$. The question becomes whether the following statement is true:

$$\operatorname{Var}\left[\sum_{i=1}^{k} \Delta\mathcal{W}_{\text{expl}_i}\right] = k \cdot \Delta t_{\text{expl}} = \operatorname{Var}\left[\Delta\mathcal{W}_{\text{impl}}\right] \tag{5.1}$$

We know that in the addition of independent random variables we have that: $X + Y \sim N(\mu_X + \mu_X, \sigma_X^2 + \sigma_Y^2)$, where $X \sim N(\mu_X, \sigma_X^2)$ and $Y \sim N(\mu_Y, \sigma_Y^2)$. Extending this notion into k-dimensions we obtain the following justification of statement (5.1):

$$\sum_{i=1}^{k} \Delta\mathcal{W}_{\text{expl}_i} \sim N(0 + \cdots + 0, \Delta t_{\text{expl}} + \cdots + \Delta t_{\text{expl}}) = N(0, k \cdot \Delta t_{\text{expl}})$$

$$\Rightarrow \quad \operatorname{Var}\left[\sum_{i=1}^{k} \Delta\mathcal{W}_{\text{expl}_i}\right] = \operatorname{Var}\left[\Delta\mathcal{W}_{\text{impl}}\right]$$

This means that in both situations we will integrate the same stochastic information along the same independent random variables. Now we have a method enabling us to compare an implicit with an explicit Euler scheme.

6 Results

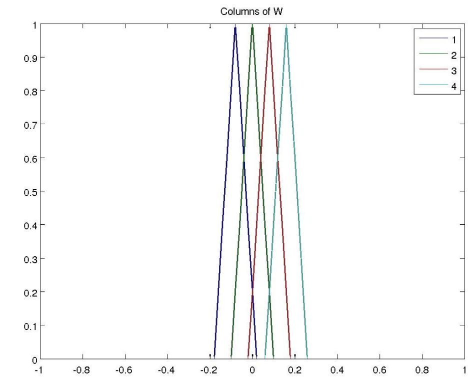
6.1 Running the Linear Stochastic Diffusion Model

In our first experiment we will model the linear one-dimensional problem from section 3. Since we want to do a thorough comparison of different situations, we will run the model with several different parameters and conditions. First we need to specify an initial condition, some boundary conditions, the forcing and a diffusion coefficient.

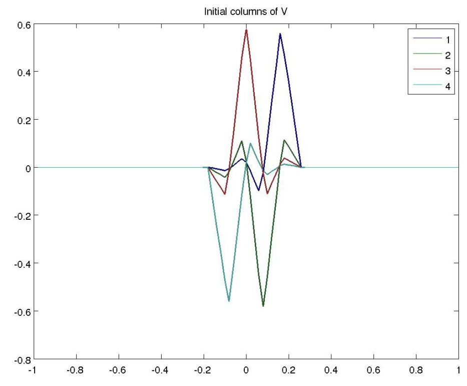
We let the deterministic forcing be $b = 0$. The stochastic part of the forcing will consist of $m = 4$ normal random variables with standard deviation $\sigma = 1$ and mean $\mu = 0$. The region of influence of these stochastic variables is located closely around the centre of the interval. This region is defined by the matrix W (see figure 1). To keep things simple we let the initial condition of the solution's deterministic part be a peak at $x = 0$, where $x \in [-1, 1]$:

$$u(x,0) = \begin{cases} 0 & \text{for} \quad x \in [-1, 0) \\ 1 & \text{for} \quad x = 0 \\ 0 & \text{for} \quad x \in (0, 1] \end{cases}$$

The initial condition for the stochastic part of the solution is similar to the stochastic forcing. We let $m = 4$ normal random variables have $\sigma = 1$ and $\mu = 0$. Their region of influence is defined by the matrix V. We let V be the nearest orthogonal matrix to W[2] (see figure 1). Last but not least, at the boundaries we define the homogeneous conditions $u(0, t) = u(1, t) = 0$.



(a) A plot of the 4 columns of W which define the regions in which the stochastic forcing has its effect.

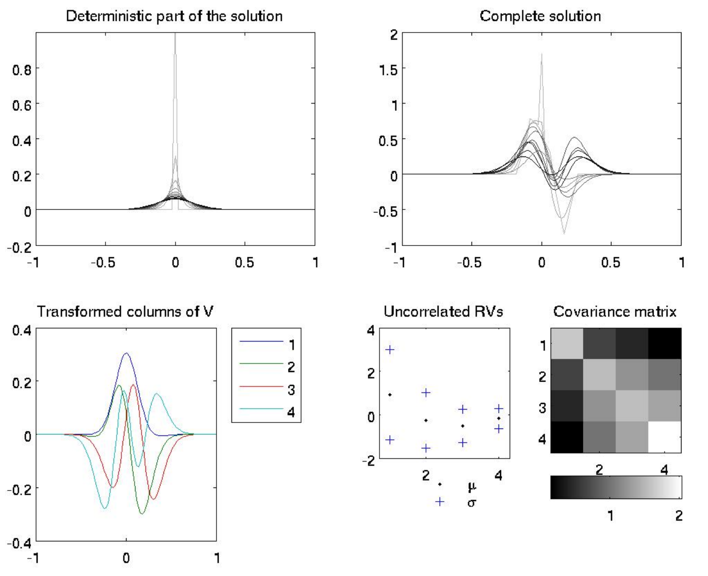


(b) The initial columns of V, such that V is the nearest orthogonal matrix to W. These columns define the region of the stochastic initial condition.

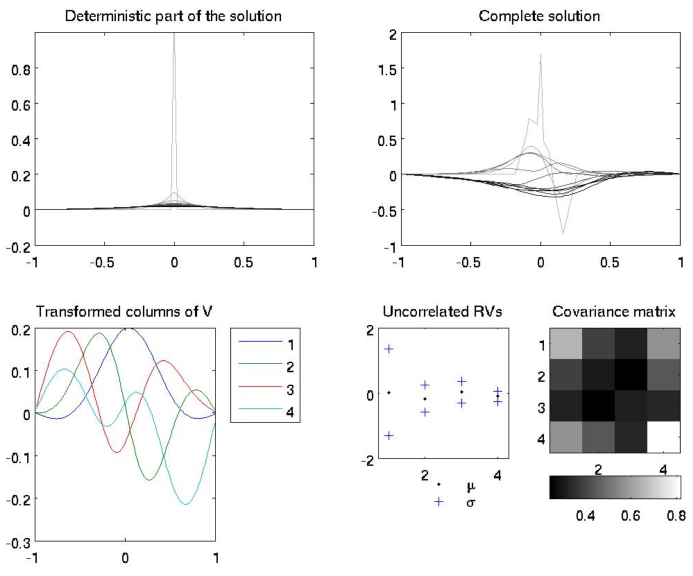
Figure 1: Plots of the matrices W and V, which define the region of stochastic influence.

We run the implicit algorithm on a grid of size $n = 100$, at different rates of diffusion: $a \in \{0.01, 0.1, 1, 10\}$, for $t \in [0, T]$ letting $T = 1$. In figure 2 we show the development of the solution for $t = k\Delta t$ with $k \in \{0, \ldots, s\}$. Here we choose a time-step $\Delta t = 0.1$. We also plot the mean and standard deviation of the uncorrelated random variables in $\mathbf{y}$ at $t = T = 1$, together with the transformed columns of V acting on these random variables.

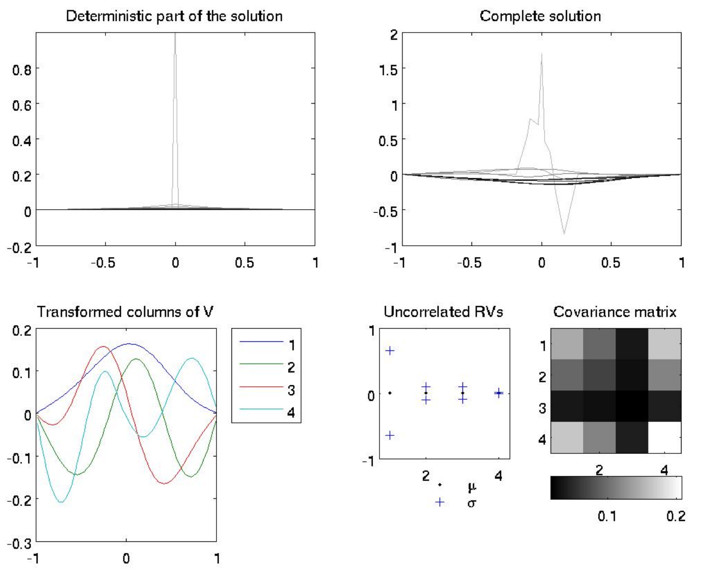
[2]An algorithm finding such a nearest orthogonal matrix is given in appendix A.4.



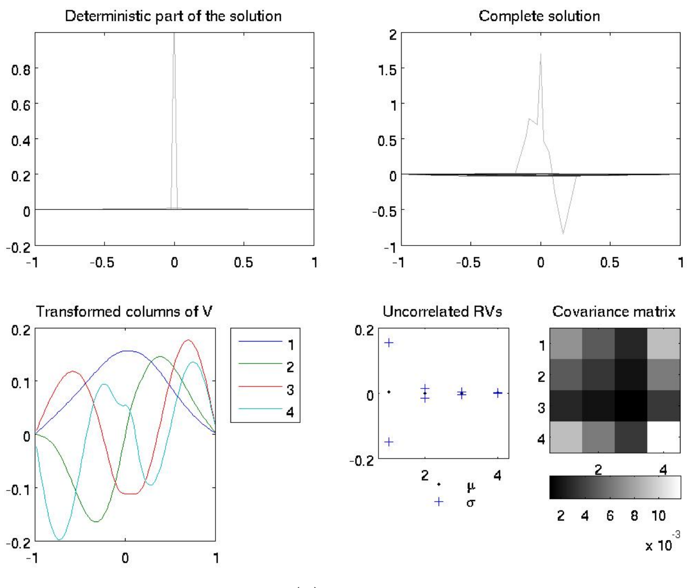
(a) $a = 0.01$



(b) $a = 0.1$



(c) $a = 1$



(d) $a = 10$

Figure 2: Numerical solutions of the linear diffusion model for different diffusion coefficients a. When progressing in time, the plot of a solution becomes darker. Hence the light-gray plots resemble the initial and early stages of the solution and the black plot shows the solution's final state for $T = 1$. The mean and variance of each random variable are computed by repeating each computation involving these variables 200 times.

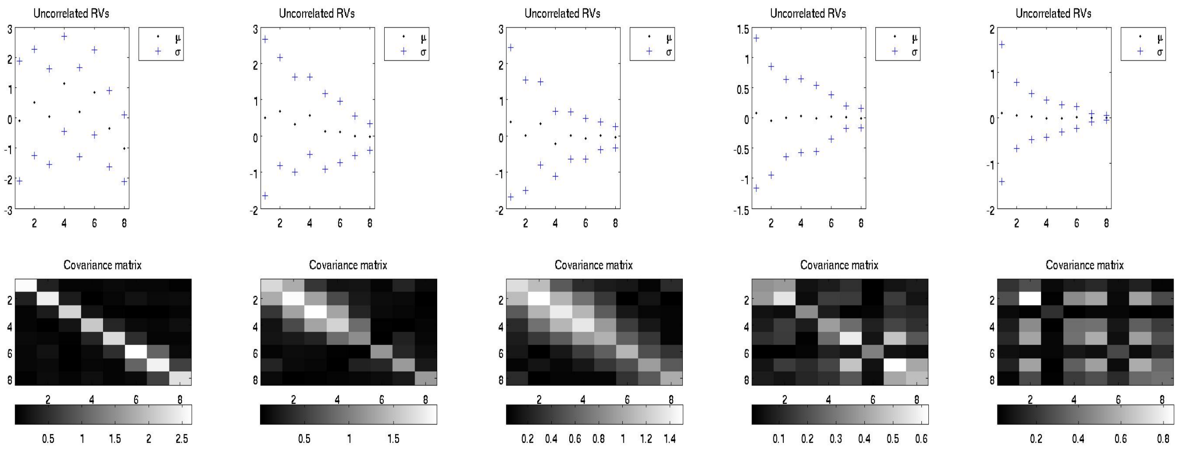
The figures 2a-2d show the effect diffusion has on the solution and especially on the probability distributions of the random variables in $\mathbf{y}$. For a small diffusion coefficient (figure 2a and

2b) we see that the deterministic solution slowly moves to its steady state, which should be $u(x,t) = 0 \ \forall \ x \in [-1, 1]$. If we combine this with the stochastic part we see that the complete solution behaves chaotically. The figures show that here the diffusion does not get the upper hand. As with the complete solution, the mean and standard deviation of the variables in $\mathbf{y}$ do not change much from their initial values. Due to the lack of diffusion these have mainly been under the influence of the stochastic forcing, which is equally distributed. We can also see this behaviour in the columns of V, which are not as spread as in the cases where there is more diffusion.

If we now consider the figures in which the diffusion gets higher (figures 2c-2d) we see that both the deterministic and the complete solution quickly tend to a steady state. The diffusion clearly gets the upper hand and manages to control the erratic behaviour of the stochastic forcing.

In the uncorrelated random variables we see the remarkable effect the diffusion has on the probability distributions. As the random variables get heavily correlated we can identify the first and most principal random variable which, combined with its respective column of V, almost solely determines the entire stochastic part of the solution. We furthermore notice that a higher diffusion forces the mean of the distributions to zero and decreases the standard deviations.

We investigate this effect in greater detail by running the model with $m = 8$ random variables, at diffusion rates $a \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$ and plot the covariance matrix together with the μ and σ of the uncorrelated random variables. The result is given in figure 3.



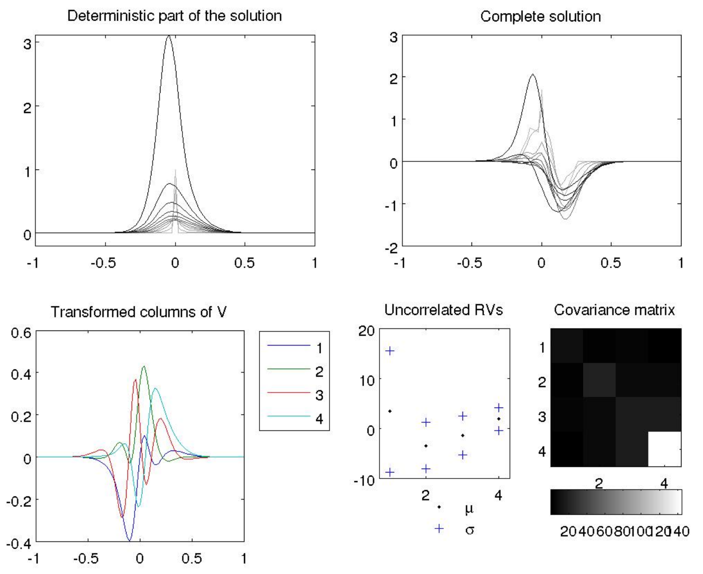
(a) $a = 0.001$ (b) $a = 0.005$ (c) $a = 0.01$ (d) $a = 0.05$ (e) $a = 0.1$

Figure 3: Distributions and covariance matrices at different values for a. Note that these images are scaled.

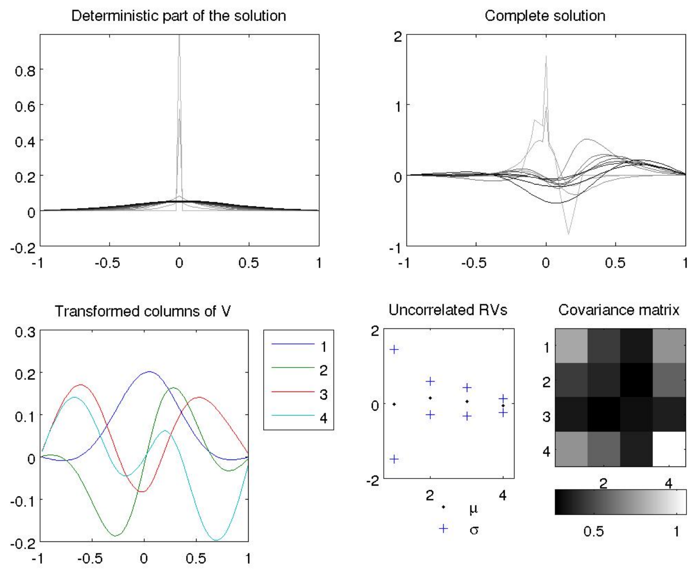
We see that the increase in diffusion results in an increase in correlation between the random variables. When these variables become more dependent it becomes easier to identify the subset defining the majority of the stochastic influence.

6.2 Running the Non-linear Stochastic Reaction Diffusion Model

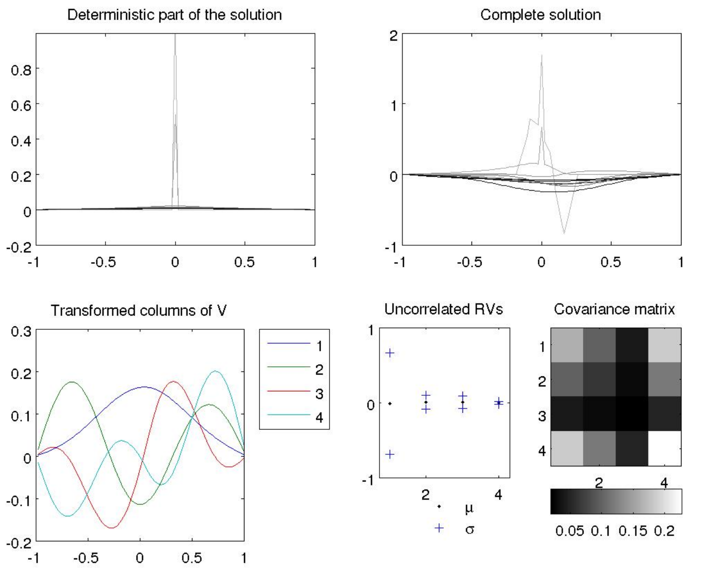
Our second experiment will be modelling the non-linear one-dimensional problem from section 4. As a bilinear form we take a square: $\langle u, u\rangle = u^2$. The other parameters and conditions remain equal to the situation in section 6.1. The results are given in figure 4.



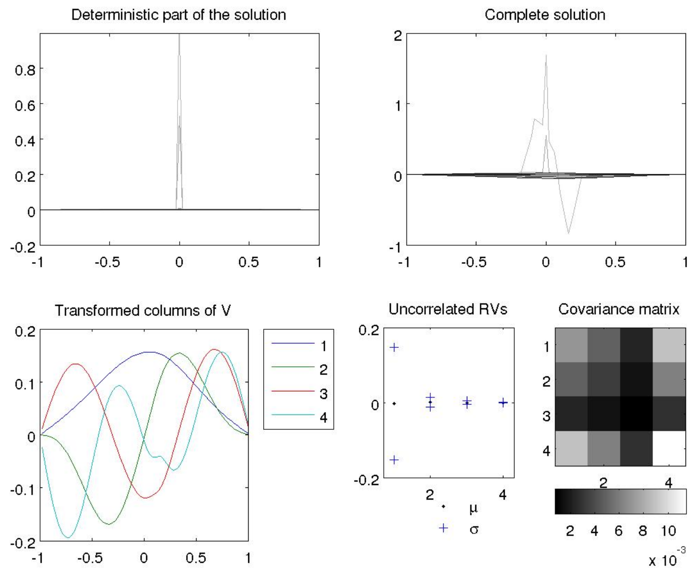
(a) $a = 0.01$



(b) $a = 0.1$



(c) $a = 1$

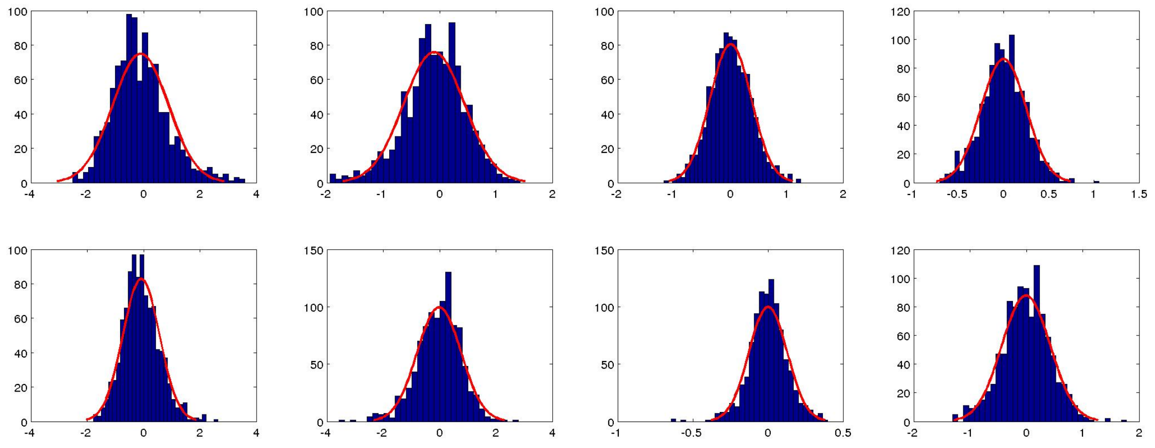


(d) $a = 10$

Figure 4: Numerical solutions of the non-linear reaction diffusion model with $\langle u, u\rangle = u^2$ for different diffusion coefficients a. The parameters and conditions in section 6.1 apply here as well.

We immediately notice that for the smallest diffusion $a = 0.01$ the method is unstable. Although the initial condition is positive and hence, in an unstable region (see section 6.5.1), we would expect the Newton-Raphson algorithm to perform better, yet it is unable to approximate the root of equation (4.12) (even with some modest *backtracking*, see section 4.1 and appendix A.3. The stochastic influence in the non-linear equation results in deviations large enough to make the algorithm's initial guess, which is the solution in the previous time-step, a really *bad* one. For higher diffusion coefficients we encounter more or less the same behaviour as in the results of section 6.1. From the covariance matrices it can still be seen that the random variables become increasingly dependent as the rate of diffusion rises. Consequently, the primary principal component becomes increasingly responsible for a major part of the stochastic information.

To justify the use of a principal component analysis we have to show that the random variables are still normally distributed. We first do this in a descriptive manner by using the function `histfit()` from `MATLAB`'s statistics toolbox. We plot the histograms with fitted normal pdf for diffusion rates $a = 0.1$ and $a = 1$. We do not consider the model for $a = 0.01$ since it is unstable. To get an accurate approximation to the true distributions of the random variables we repeat each computation 1000 times. That is, the data consists of 1000 trials for each variable. The result is clear form figure 5.



(a) $a = 0.1$ (b) $a = 1$

Figure 5: Histograms of the $m = 4$ random variables at different values for a.

If we want to test for normality in a more rigorous way, we cannot simply perform a χ^2 goodness of fit test, since we are dealing with highly correlated data. However, by using PCA, we are able to perform a similarity transformation to the matrix of observations obtaining the uncorrelated measurements. In this transformed set of data the χ^2 test is applicable. Taking the null hypothesis to be a normal distribution, then `MATLAB`'s implementation of this test, `chi2gof()`, returns, as expected, that this null hypothesis cannot be rejected at a 5% statistical significance level. We conclude that the original data in the matrix of observations is jointly normal. For PCA to work we need that the probability distributions are entirely described by the two parameters μ and σ [8]. Since the data is jointly normally distributed, this condition is satisfied and we can perform a principal component analysis.

6.3 A Non-linear Stochastic Reaction Diffusion Model in 2D

Let us now solve a two-dimensional reaction diffusion problem. Again we take the bilinear form $\langle u, u\rangle = u^2$. The problem becomes:

$$\frac{\partial}{\partial t} u = a\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u + u^2 - f$$

We want an initial condition to be similar to the one used in sections 6.1 and 6.2. We therefore take it to be a peak at the exact center of the domain. Thus, taking the domain to be $(x, y) \in [-1, 1] \times [-1, 1]$, we have that:

$$u(x, y, 0) = \begin{cases} 0 & \text{for} \quad (x, y) \in \{[-1, 1] \times [-1, 1]\} \setminus (0, 0) \\ 1 & \text{for} \quad (x, y) = (0, 0) \end{cases}$$

Taking $n = 50$ we have a uniform grid of size 50×50. In figure 6a, the initial condition is plotted as the image of a 50×50 matrix. Similar to sections 6.1 and 6.2 we neglect any deterministic forcing, $b = 0$ and let the stochastic forcing consist of $m = 4$ random variables with $\mu = 0$ and $\sigma = 1$. In the previous sections we had that the region of stochastic influence was defined by the columns of W and V describing triangular functions. We will extend this idea into two dimenstion by creating columns of W and V that, reshaped into $n \times n$-matrices, resemble pyramids. V is again initially chosen to be the nearest orthogonal matrix to W. The reshaped columns of W and the initial V are given in figure 6. At the boundaries we, again, take homogeneous Dirichlet conditions.

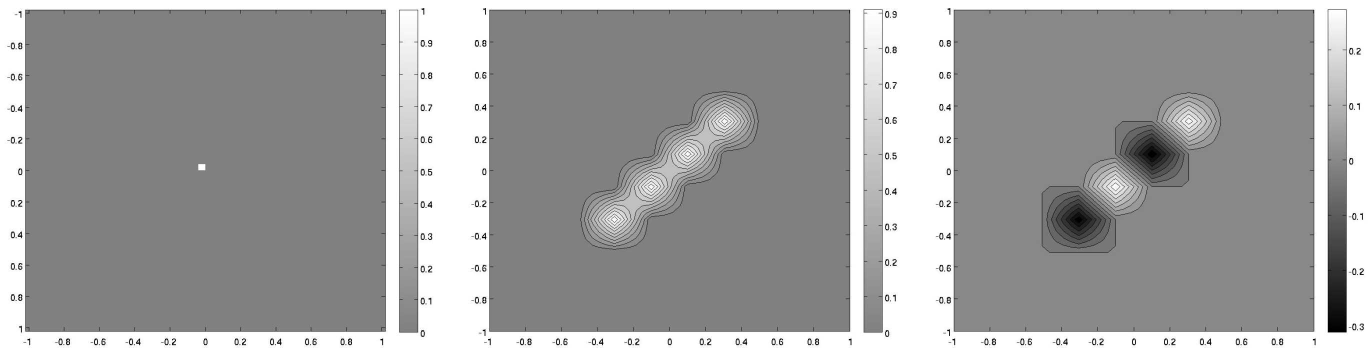
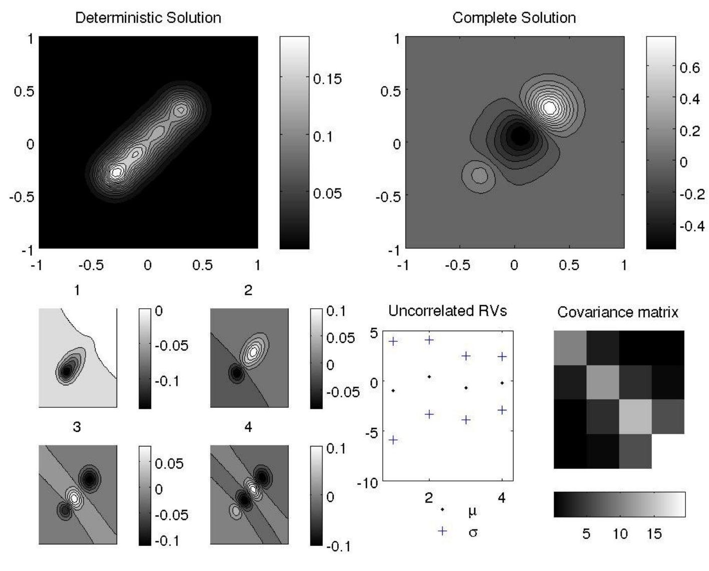

(a) (b) (c)

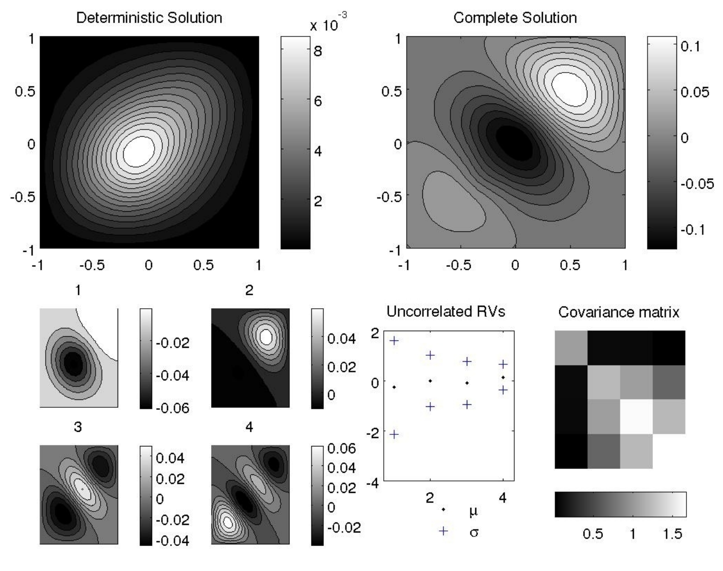
Figure 6: The initial condition $u(x, y, 0)$ (6a), the regions of stochastic influence in the forcing (6b) and in the initial condition (6c). Here the pyramids are combined into a single figure, but it should be clear that each pyramid acts on a single random variable.

We run the implicit algorithm at rates of diffusion $a \in \{0.01, 0.1, 1, 10\}$ and plot the deterministic and complete solution at $t = T = 1$. We furthermore plot the principal components together with the reshaped columns of V acting on them. Last but not least we plot a covariance matrix to keep track of the degree of correlation. The results are shown in figure 7.

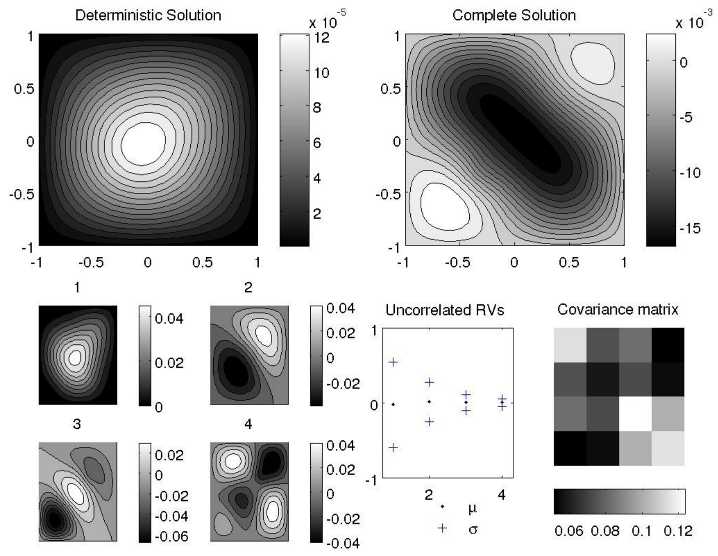
The advantage of performing this two-dimensional simulation is that we can observe the influence of V in greater detail. If we look at figure 7b we see the direct effect of the first and



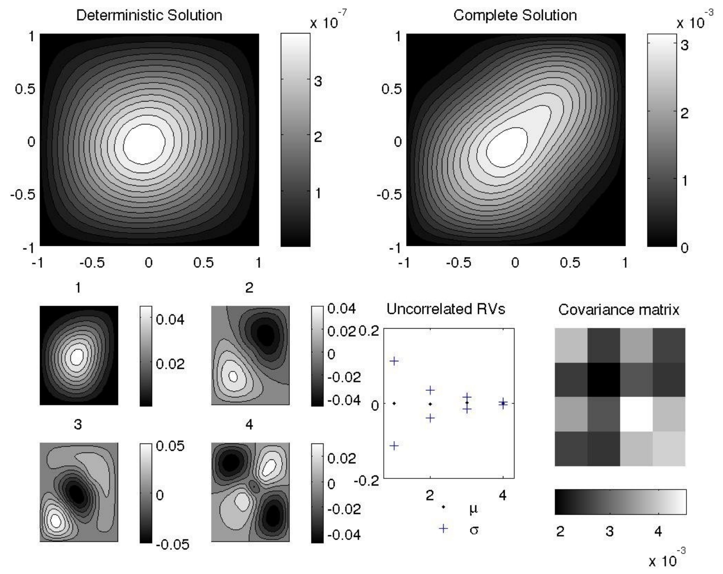
(a) $a = 0.01$



(b) $a = 0.1$



(c) $a = 1$

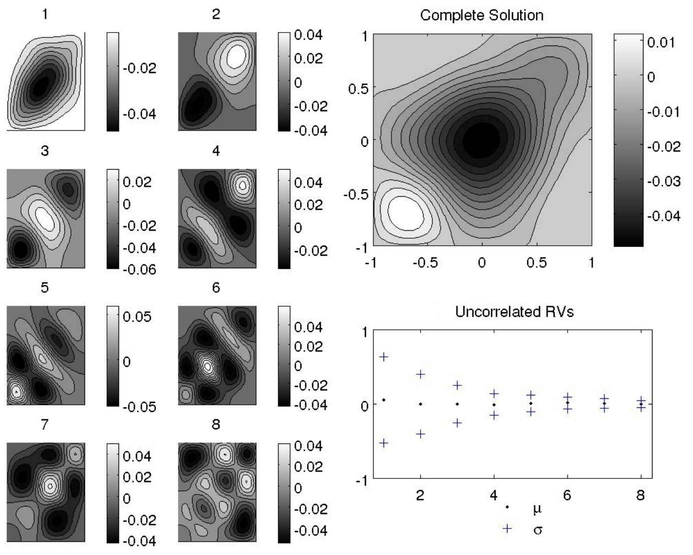


(d) $a = 10$

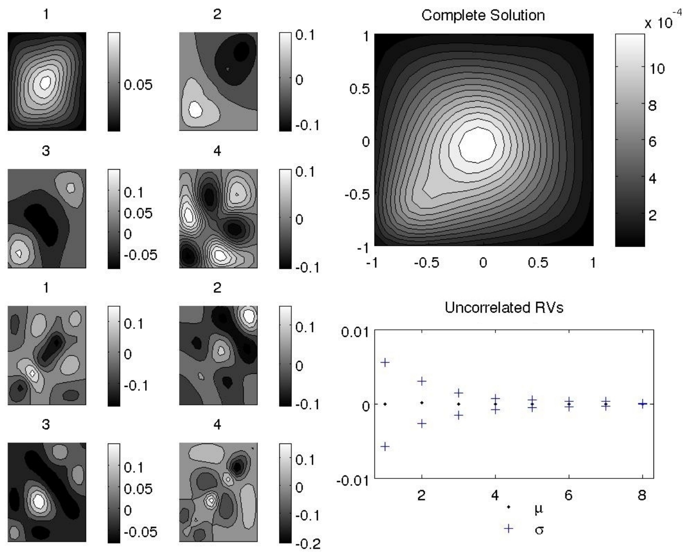
Figure 7: Numerical solutions of the 2D non-linear reaction diffusion model with $\langle u, u\rangle = u^2$ for different diffusion coefficients a. We choose contourplots to show the two-dimensional shape of the solutions.

second column of V combined in the complete solution. This similarity can also be seen in figures 7a and 7c. In figure 7d the effect is not noticable since the diffusion is too strong and the solution has reached a steady state at $T = 1$. Note that te complete solution shown in the upper right corner of every figure is a particular one. To obtain a complete solution we need to choose a vector $\mathbf{y}$ from the 200 vectors we have used to obtain the principal components.

Since we basically apply the same equation as in section 6.2 we can assume that the random variables are again jointly normally distributed. When the diffusion rises, the uncorrelated random variables exhibit the same behaviour as we have seen in the previous experiments and again, the covariance matrices show how the random variables become increasingly correlated. To show how the principal random variables affect the solution, we will now perform a simulation with $m = 8$ random variables and with an initial condition $u(x, y, 0) = 0 \ \forall \ (x, y) \in [-1, 1] \times [-1, 1]$. We will take relatively high diffusions $a = 1$ and $a = 10$. The results are shown in figure 8.



(a) $a = 1$



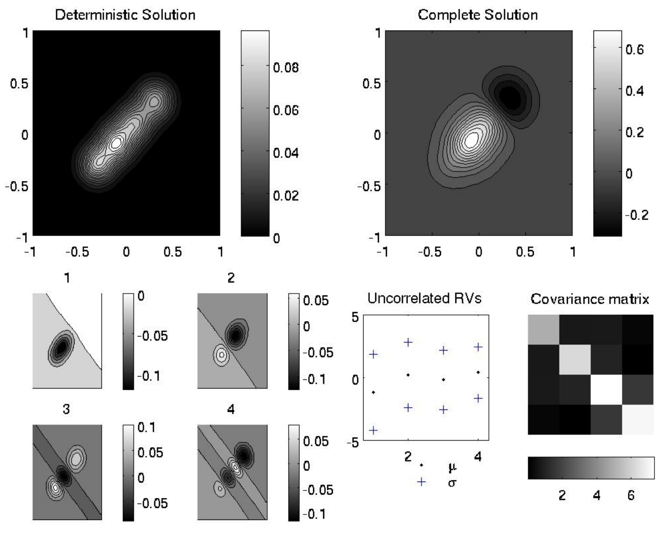
(b) $a = 10$

Figure 8: Plots of the 8 columns of V, a particular complete solution and the principal components.

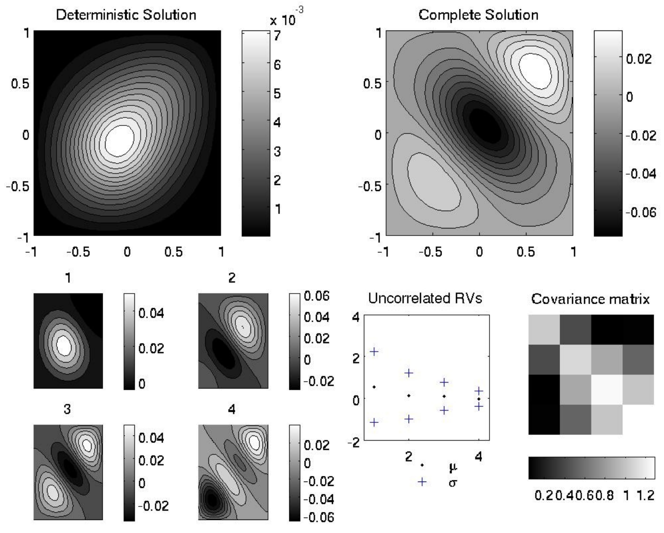
From figure 8 we can see that the complete solution is roughly given by the combination of the first two principal components with the first and second columns of V displayed in the upper left of each figure.

6.4 Running an Explicit Scheme

In this section we will investigate the possibility of achieving the result from section 6.3 with an explicit algorithm. This will be a test for the implicit algorithm, such that we may know whether the deviations in the solution due to several approximations are at an acceptable level. The explicit algorithm is adapted such that we integrate the same random variables along the same Wiener path, as discussed in section 5.2, while maintaining stability. We use the exact same parameters and conditions as in section 6.3. To perform a stable explicit computation for $t \in [0, T]$, where $T = 1$, with a grid of size 50×50, we need a small time-step. This significantly increases the computation time. We therefore restrict the computations to the situations where the diffusion is $a = 0.01$ and $a = 0.1$. These are the relevant cases since the implicit scheme performed questionably at these diffusion rates. The results are given in figure 9.



(a) $a = 0.01$, $\Delta t_{\text{expl}} = \Delta t_{\text{impl}}/10$



(b) $a = 0.1$, $\Delta t_{\text{expl}} = \Delta t_{\text{impl}}/25$

Figure 9: Explicit solutions of the 2D non-linear reaction diffusion model with, except for the size of Δt, the same parameters as in section 6.3

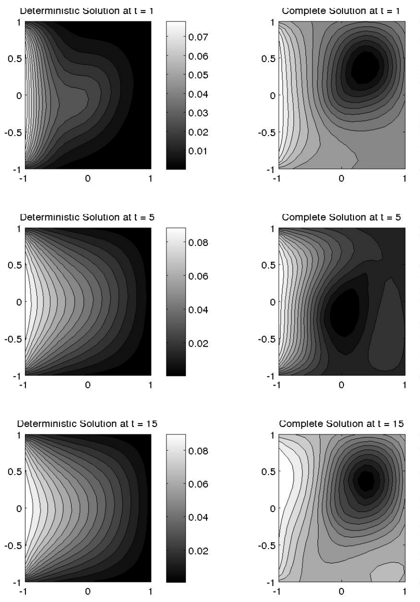
Let us first compare figure 9a with figure 7a. There is a slight difference in the structure of the complete solutions. However, the maximum in these solution at $T = 1$ is in both cases around 0.6. If we compare the structure of the columns of V we find that these are almost exactly similar. This also holds for the magnitude of the variances in the principal components and for the structure of the covariance matrices.

Now we compare figure 9b with figure 7b. Here it becomes hard to find any large discrepancies, since the overall structure of the complete solution and the columns of V are practically equal. Yet, here we find that the maximum height of the implicit solution is somewhat higher than in the explicit case. However, the principal components and covariance matrices exhibit the same expected behaviour. If we regard the explicit model as the "true" solution of the problem, then we can be fairly satisfied with the results from the implicit method. In spite of the necessary approximations, the implicit solution behaves more or less in the way it should behave.

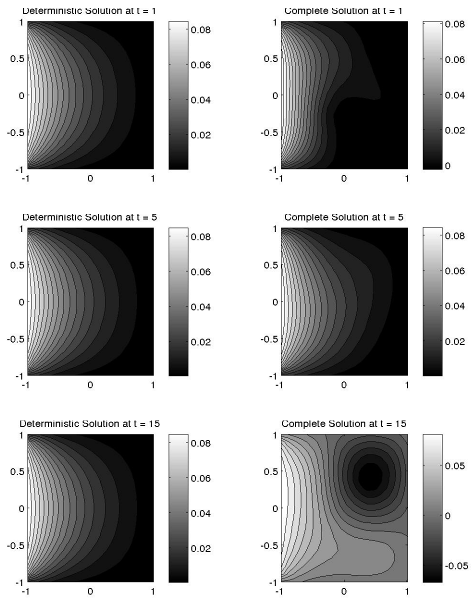
6.5 Results in the Steady State

What happens when the solution seems to have converged? Of course there is no strict convergence since we have the ongoing influence of the Brownian increments, but we can tell when the influence of the diffusion and the forcing, together with the stochastic part have reached an equilibrium. We can talk about a steady state when the probability distributions of the random variables remain unchanged in time. The changing variance of $\mathbf{y}$'s components will be the main indicator of this phenomenon. Note that this equilibrium does not imply that there are not going to be any variations in the complete solution. These will still be clearly visible due to the random fluctuations in the forcing. We will, however require a truly stable deterministic solution.

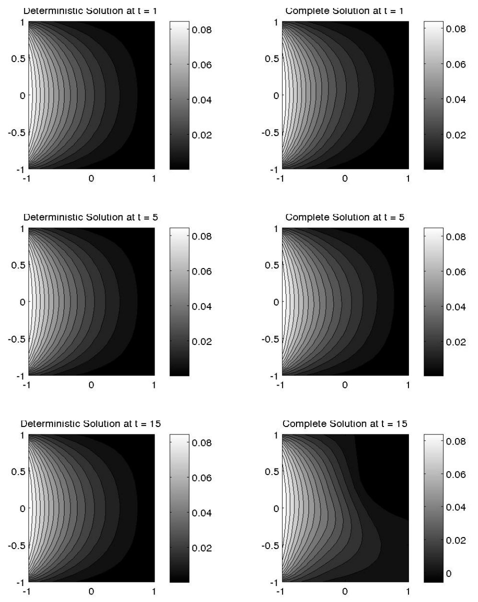
We will plot the deterministic and the complete solution of the non-linear two-dimensional model at times $t = 1$, $t = 5$ and $t = 15$. In each computation we will take $\Delta t_{\text{impl}} = 0.1$, to maintain an equal stochastic integration. We choose to implement an inhomogeneous boundary condition: $u(0, y, t) = 0.1$ for $t \in [0, T]$ and $y \in [-1, 1]$. This is convenient since this situation has an easily recognisable steady state. We will perform this analysis for $a = 0.1$, $a = 1$ and $a = 10$, rates of diffusion for which the implicit method has shown to be stable. The results are shown in figure 10.



(a) $a = 0.1$



(b) $a = 1$



(c) $a = 10$

Figure 10: Deterministic and complete solutions at $t = 1$, $t = 5$ and $t = 15$.

As expected, for a high diffusion ($a = 10$) the solution quickly reaches a steady state. At $t = 5$ this is already the case. The simulations with a lower diffusion rate take some more time to reach an equilibrium. To show the evolution of the variances we plot the largest measured variance against the time. If the size of the variances and the deterministic solution remain unchanged we consider the state to be stable. The results are shown in figure 11

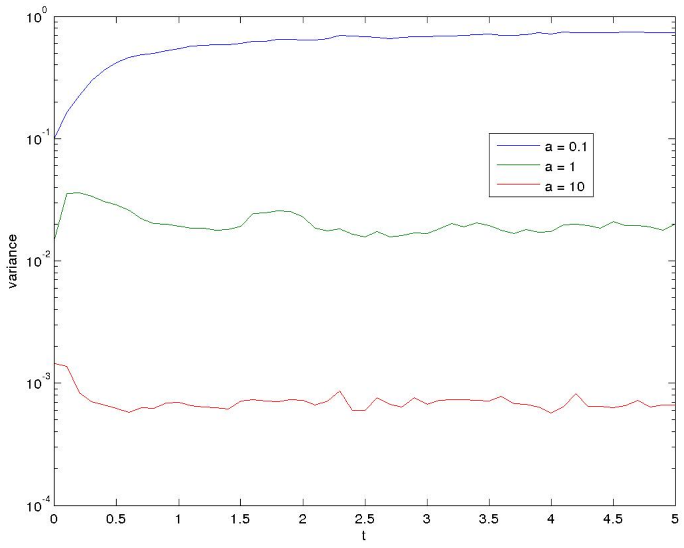


Figure 11: Time-evolution of the largest variance obtained from the matrix of observations.

From figure 11 we see that the variance in the model with the smallest diffusion stabilises at $t = 4$. The variances in the other models reach a steady state at around $t = 3$. Combining this with the information from figure 10 we can roughly state that for $a = 0.1$ we enter a steady state at $t = 5$. For $a = 1$ and $a = 10$ the variance takes more time to stabilise than the deterministic solution. At about $t = 3$ we can say that the complete solution has become stable. This implies that for a high rate of diffusion, the stochastic components cause the system to reach its equilibrium after a longer period of time than it would in a deterministic situation.

The long term solution at a low diffusion rate will exhibit a fairly regular behaviour. That is, the occurring fluctuations due to the stochastic forcing will be within the range of the constant variance. In such a situation we can say that the system is going into a *statistical steady state*. A higher diffusion will suppress the forcing even more, which leads to a quicker entering into this equilibrium. Yet, in this case we see that the stochastic forcing has a negative effect on the speed of entering the steady state.

6.5.1 Stability Analysis in models with Forcing

Up until now we have been focusing on stable situations. Finally we are going to bring some variations into the initial condition and the forcing to observe the instabilities at a low diffusion rate. We have already seen the instability of the non-linear equation in the results for low diffusion rates in sections 6.2 and 6.3. We will see that if we would have taken a negative initial condition, the solution would remain stable.

From the theory on bifurcations we know where to expect certain stability points. These can be estimated from equation (4.1) by letting $\frac{\partial}{\partial t}u = 0$. If we plug in the bilinear form $\langle u, u\rangle = u^2$ and neglect the diffusion term we find that the stability points should be approximately at $u = \pm\sqrt{f}$, where $-\sqrt{f}$ is a sink and $+\sqrt{f}$ is source for $f > 0$ (see figure 12). The diffusion term will have a stabilising effect, thus we can expect the true stability points to be slightly further away.

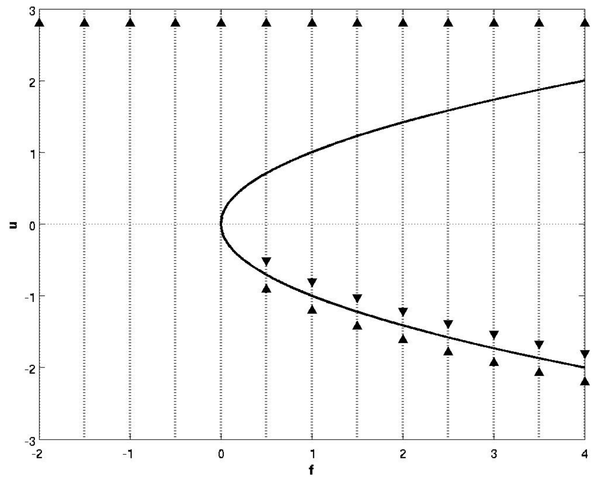


Figure 12: Bifurcation diagram of the equation $\frac{du}{dt} = u^2 - f$.

We have that the solution and the forcing are partially stochastic. This means that near the unstable equilibrium point it will be a matter of chance whether the solution will move into the region of attraction or diverge into infinity. We will now try to find the bounds of this region of uncertainty by testing the model with different initial conditions u and forcing constants f. We will keep track of the solution's extrema to see whether the solution has found the stable sink at $-\sqrt{f}$, or is diverging into infinity. Whether a solution converges is tested for the parameters (f, u), where $u \in \{0, \ldots, 3\}$ and $f \in \{0, \ldots, 4\}$. The result is shown in figure 13.

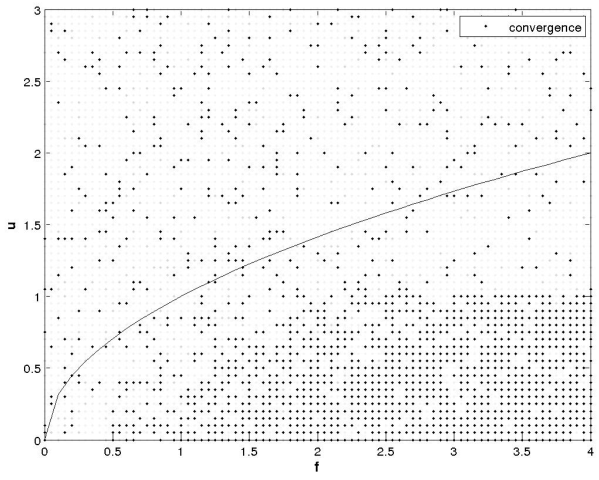
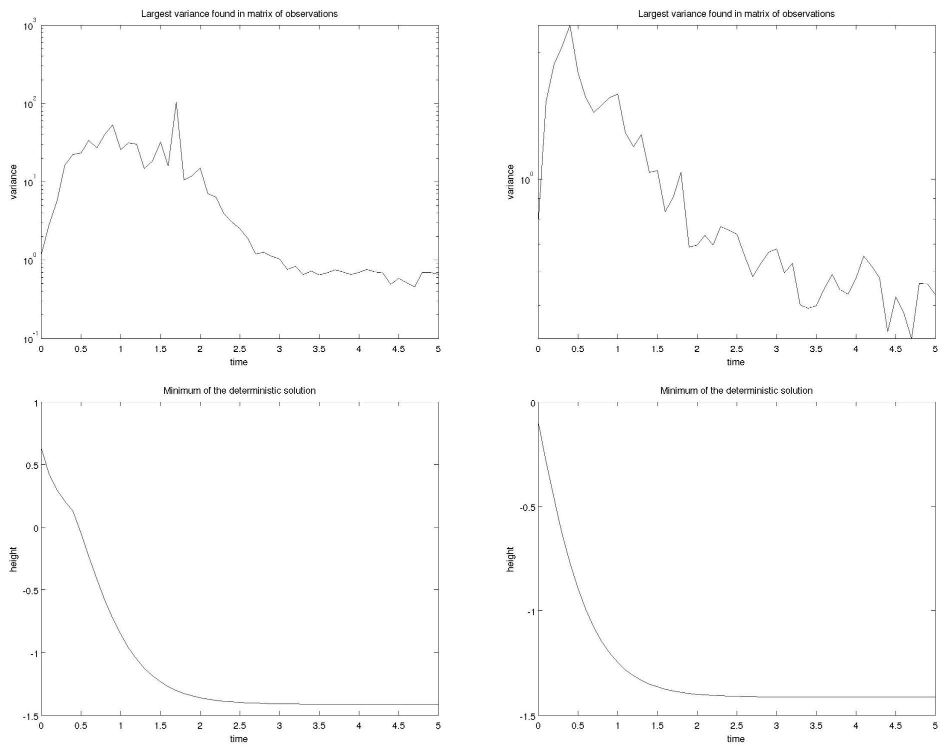


Figure 13: Coordinates (f, u) that show stability are plotted as black dots. Each dot represents a simulation of the non-linear reaction-diffusion equation with diffusion coefficient $a = 0.01$.

As we can see from figure 13, there exists a region within the bound of the positive square root of f, where it is uncertain whether the given initial condition leads to a converging solution. Now we are able to make an educated guess on which values for u and f will give a converging solution, which enables us to perform the study of the steady state in section 6.5 for a smaller value of a. We will choose combinations (f, u) that are close to the region of instability and observe the stabilisation process.



(a) $(f, u) = (2, 1)$ (b) $(f, u) = (2, 0)$

Figure 14: Plots of the minimum of the deterministic solution and the evolution of the variance for $a = 0.01$.

If we look at the choice of $(2, 1)$ in figure 13, we see that this coordinate lies on the edge of what is probably a stable region. Thus it may be difficult for the solution to reach the steady state in the sink. From figure 14a we see that, at a low rate of diffusion and this choice of parameters we have that the deterministic solution and the variance both reach their stable state at about $t = 3$. In figure 14b the experiment is repeated for $(f, u) = (2, 0)$, which should be a more stable choice. We see that the deterministic solution takes a time-period slightly less than $t = 3$ to find the sink at $-\sqrt{2}$. As expected, this is a more stable configuration. If we now look at the evolution of the variance we see that these are at $t = 3$ not yet converging. This means that in this stable configuration it is the stochastic part that is, again, responsible for the slow convergence to the steady state. We can conclude that, either the deterministic and stochastic part achieve their stable state at the same time, which is the case in unstable configurations, or the stochastic part converges slower than the deterministic part.

7 Discussion and Conclusions

First of all, whether it is possible to model the linear and especially the non-linear SPDE's implicitly should be clear by now. But in the non-linear case there are some difficulties in situations where there is little diffusion, a lot of stochastic forcing and inconveniently chosen initial conditions. These situations, discussed in section 6.5, give rise to questioning the approximations in the implicit scheme. The Newton-Raphson algorithm computes a root of a function containing the entire set of information necessary to obtain the solution. This can be problematic since there is a stochastic forcing present in this equation. In the models we studied an improvement might be to separate the stochastic forcing from the rest of the equation. This is possible since, in our models, we do not have coefficients in the stochastic term depending on the solution. If the stochastic forcing can be separated from the Newton algorithm, this might increase its accuracy. This term should then be added to the solution when the root approximation has finished. Note that this method is not applicable in the general situation where the stochastic term of the differential equation does depend on the solution. An example of such a general form is the test equation in section 3.1.1.

In spite of the instabilities, the implicit scheme has proven to be fairly accurate in section 6.4, where we compared the structure and height of the solutions from the implicit and explicit schemes. Here the solution from the explicit scheme served as a reference to which we could compare the implicit solutions. This might seem a bit odd since the explicit discretisations are, by themselves, not exact solutions of the problem. However, by taking an adequately small time-step it is possible to have a solution that is stable and does not depend on any (Newton) approximations, except for the inevitable spatial and time-discretisations. This should be enough reason to take the explicit solution as the "true" one.

In comparing the implicit with the explicit scheme we had the issue of integrating an equal stochastic process. This should be adequately solved by the method discussed in section 5.2, but there are some other issues to consider. For example, if it is necessary to use an implicit scheme with a very large time-step, then the integration of the stochastic process will automatically be too coarse, resulting in unwanted behaviour. Hence, while for stability the time-step is not restricted, to obtain useful results one has to restrict it to the largest possible step in a discretised stochastic process, thereby ensuring a proper stochastic integration. This is quite a restriction on the methods developed in this project. Especially on the non-linear models, since in the linear models it is possible to separate the stochastic from the deterministic part and integrate the deterministic part over an arbitrary large time-step, while maintaining a small step for the stochastic part.

In the analysis of the steady state in section 6.5 and the stability regions in section 6.5.1, we find that it is the stability of the initial conditions that determine which part of the solution takes the largest amount of time to enter a steady state. But we can say that in general the stochastic part is responsible for extending the stabilisation time. It should therefore be possible to determine whether a stochastic diffusion system has found its steady state by only observing the time-evolution of the probability distributions. However, due to computational limits we have not been able to observe the behaviour of the models at very small rates of diffusion and very unstable initial configurations, hence we cannot be certain of this.

Let us now consider the results from section 6.1 to 6.3. Here we observe the effect of diffusion on linear and non-linear SPDE's. It is clear that a high diffusion coefficient leads to large correlations between the random variables, which can be seen from the covariance matrices. This effect can also be witnessed indirectly in the results from the principal component analysis, where some of the uncorrelated random variables lose their influence and others gain some. Combining this with the transformed orthogonal matrices governing the random variables, we observe the influence of a subset of the initial random variables. Via the PCA we are thus able to recognise the stochastic variables responsible for a large part of the solution. In models with a large diffusion coefficient we might even be able to isolate the major components and get rid of the less important ones. This means that we create a new step in the algorithm computing the principal components, but at the same time reduce the amount of random variables. It should be interesting to see whether this can be an improvement to the method.

References

[1] Desmond J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 43(3):pp. 525–546, 2001.

[2] Desmond J. Higham. Mean-square and asymptotic stability of the stochastic theta method. *SIAM Journal on Numerical Analysis*, 38(3):pp. 753–769, 2001.

[3] P.E. Kloeden and E. Platen. *Numerical solution of stochastic differential equations.* Springer-Verlag, Berlin, 1992.

[4] D.C. Lay. *Linear algebra and its applications*, chapter 7.5. Pearson/Addison-Wesley, Boston, third edition, 2006.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[6] Yoshihiro Saito and Taketomo Mitsui. Stability analysis of numerical schemes for stochastic differential equations. *SIAM Journal on Numerical Analysis*, 33(6):pp. 2254–2267, 1996.

[7] Themistoklis P. Sapsis and Pierre F.J. Lermusiaux. Dynamical criteria for the evolution of the stochastic dimensionality in flows with uncertainty. *Physica D: Nonlinear Phenomena*, 241(1):60 – 76, 2012.

[8] Jonathon Shlens. A tutorial on principal component analysis. `http://www.snl.salk.edu/~shlens/`, 2005.

[9] F.W. Wubs. *Computational Methods of Science.* Lecture Notes in Applied Mathematics. University of Groningen, 2011.

A Code

A.1 Principal Component Analysis

The following is a MATLAB function performing a principal component analysis of a given data set by the method discussed in section 5.1.

```
function [signals Var PC CovMat] = PCA(G)
%% PC-Analysis. [signals Var PC] = PCA(G).
% Accepts an {m}x{n} matrix
% containing {m} variables, measured {n} times.
% It returns the principal components {PC} (combined they
% may perform a change of basis), the transformed data {signals}
% and the variances {Var} in decreasing order.
% It furthermore returns a covariance matrix.

[~, experiments] = size(G);
mn     = mean(G,2);
GPCA   = G - repmat(mn,1,experiments);  % mean deviation form
CovMat = (GPCA * GPCA')/ (experiments - 1);

Y      = GPCA'/sqrt(experiments - 1);
[~, S, PC] = svd(Y);
S = diag(S);
Var = S.*S;
signals = PC' * GPCA;
end
```

A.2 Jacobian Approximation

A MATLAB function computing an approximation of the Jacobian of a given function handle, focusing on the multidimensional Newton-Raphson method.

```
function [J] = jcbNewt(F,Init)
%JACB Expects a function handle and an
% initial guess for the Newton method.
% Returns the Jacobian matrix of the
% function F at the vector Init.
% tic
n = length(Init);
J = zeros(n,n);
d = 1e-6;
for i = 1:n
    xa = Init;
    xb = xa;
    xb(i) = xa(i) + d;
    J(:,i) = (F(xb) - F(xa))/d;
end
% jcbnewt_time = toc
```

A.3 Newton-Raphson

The implementation of the Newton-Raphson method:

```
function [sol,nrm,iter] = NR_method(fhandle,xinit,tol,maxiter)
%NR_METHOD NR_method(fhandle,xinit,tol,maxiter) approximates the root of
% the function specified by fhandle, using a Newton-Raphson algorithm with
% initial guess xinit. A tolerance tol and maximum number of iterations
% maxiter need to be specified. The algorithm makes use of backtracking via
% the variable lambda.
% tic
J       = @(x) jcbNewt(fhandle,x); % computes the jacobian of the function
                                   % in fhandle at x
nrm     = 1;
iter    = 0;
lambda  = 1;
while nrm > tol && iter < maxiter
    sol   = xinit - lambda*(J(xinit) \ fhandle(xinit));
    if max(abs(fhandle(sol)) > abs(fhandle(xinit))) ~= 0 && lambda > 0.1
        lambda = lambda / 2;
    else
        lambda = 1;
        nrm    = norm(fhandle(sol));
        xinit  = sol;
        iter   = iter + 1;
    end
end
% nr_method_time = toc
end
```

A.4 Nearest Orthogonal Matrix

An algorithm finding a nearest orthogonal matrix:

```
function [Aort] = nearestOrth(A)
%NEARESTORTH finds and returns the nearest orthogonal matrix to a given
% matrix A. The method is based on the singular value decomposition, where
% the matrix containing the singular values is replaced by a matrix with
% ones on the diagonal, thereby creating the nearest orthogonal matrix.
[U,S,V] = svd(A);
[m n] = size(S);
S = speye(m,n);
Aort = U*S*V;
end
```

A.5 Simulations

The rest of the code, including the simulations, can be found at the following website under Bachelor Projects: `http://www.math.rug.nl/cmnm/Main/Teaching`