Numerical methods for studying transition probabilities in stochastic ocean-climate models

Baars, Sven

CHAPTER 4

LYAPUNOV EQUATIONS

In this chapter, we focus on stochastic partial differential equations (SPDEs) describing fluid flows for which certain processes have been represented stochastically or for which the forcing of the flow has stochastic properties. In our case this is the stochastic representation of the freshwater forcing as described in Section 2.5.2. The direct way to investigate these flows is to use ensemble simulation techniques for many initial conditions to estimate the probability density function (PDF) for several observables of the flows (Slingo and Palmer, 2011). Other methods which have been suggested use some form of model order reduction. For example, a stochastic Galerkin technique forms the basis of the Dynamical Orthogonal Field method (DO) (Sapsis and Lermusiaux, 2009). Non-Markovian reduced models can also be obtained by projecting on a basis of eigenvectors of the underlying deterministic system (Chekroun et al., 2015).

In a deterministic-stochastic continuation method recently suggested by Kuehn (2012), the results from fixed point computation in a deterministic model are used to obtain information on the stationary PDF of the stochastically forced system. A restriction is that linearized dynamics near the fixed point adequately describe the behavior of the stochastic system. In this case, one only needs the leading-order linear approximation and determines the covariance matrix from the solution of a Lyapunov equation. This provides all the information to determine the probability of sample paths. The nice aspect of this method is that one can combine it easily with deterministic pseudo-arclength continuation methods. However, in order to apply the approach to systems of PDEs with algebraic constraints, generalized Lyapunov equations have to be solved.

Direct methods to solve a generalized Lyapunov equation such as the

Bartels–Stewart algorithm (Bartels and Stewart, 1972) are based on dense matrix solvers and hence inapplicable for large systems. Other existing methods which use low-rank approximations such as Extended and Rational Krylov subspace methods (Simoncini, 2007; Druskin and Simoncini, 2011; Stykel and Simoncini, 2012; Druskin et al., 2014) and alternating directions implicit (ADI) based iterative methods (Kleinman, 1968; Penzl, 1999) might also become expensive for high-dimensional problems, particularly when trying to use previous initial guesses along a continuation branch.

The aim of this chapter is to present new methodology to efficiently trace PDFs of SPDEs with algebraic constraints in parameter space. In Section 4.1, an extension of the approach suggested in Kuehn (2012) and the novel procedure to efficiently solve a generalized Lyapunov equation numerically are presented. In fact, this solves an open conjecture in Kuehn (2015), which states that it should be possible to reuse previous solutions of a continuation in a specialized Lyapunov solver. We describe the results on the model of the Atlantic Ocean circulation in Section 4.2. The numerical aspects and capabilities of the novel method for this application are shown in Section 4.3. In Section 4.4, we provide a summary and discuss the results.

4.1 Methods

Any ocean-climate model consists of a set of conservation laws (momentum, mass, heat and salt), which are formulated as a set of coupled partial differential equations, that can be written in general form as (Griffies, 2004)

$$\mathcal{M}(\boldsymbol{p})\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{L}(\boldsymbol{p})\boldsymbol{u} + \mathcal{N}(\boldsymbol{u},\boldsymbol{p}) + \mathcal{F}(\boldsymbol{u},\boldsymbol{p}), \tag{4.1}$$

where $\mathcal{L}$, $\mathcal{M}$ are linear operators, $\mathcal{N}$ is a nonlinear operator, $\boldsymbol{u}$ is the state vector, $\mathcal{F}$ contains the forcing of the system and $\boldsymbol{p} \in \mathbb{R}^m$ indicates a vector of parameters. Appropriate boundary and initial conditions have to be added to this set of equations for a well-posed problem.

4.1.1 Formulation of the problem

When (4.1) is discretized, eventually a set of ordinary differential equations with algebraic constraints arises, which can be written as

$$M(\boldsymbol{p})\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = L(\boldsymbol{p})\boldsymbol{x} + N(\boldsymbol{x},\boldsymbol{p}) + F(\boldsymbol{x},\boldsymbol{p}), \tag{4.2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector, $M(\boldsymbol{p}) \in \mathbb{R}^{n\times n}$ is a generally singular matrix of which every zero row is associated with an algebraic constraint, $L(\boldsymbol{p}) \in \mathbb{R}^{n\times n}$ is the discretized version of $\mathcal{L}$, and $F : \mathbb{R}^m \to \mathbb{R}^n$ and $N : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ are the finite-dimensional versions of the forcing and the

nonlinearity respectively. When noise is added to the forcing, the evolution of the flow can generally be described by a stochastic differential-algebraic equation (SDAE) of the form

$$M(\boldsymbol{p})\,\mathrm{d}\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t;\boldsymbol{p})\,\mathrm{d}t + \boldsymbol{g}(\boldsymbol{X}_t;\boldsymbol{p})\,\mathrm{d}\boldsymbol{W}_t, \tag{4.3}$$

where $\boldsymbol{f}(\boldsymbol{X}_t;\boldsymbol{p}) = L(\boldsymbol{p})\boldsymbol{X}_t + N(\boldsymbol{X}_t,\boldsymbol{p}) + F(\boldsymbol{X}_t,\boldsymbol{p})$ is the right-hand side of (4.2), $\boldsymbol{W}_t \in \mathbb{R}^{n_w}$ is a vector of n_w-independent standard Brownian motions (Gardiner, 1985), and $\boldsymbol{g}(\boldsymbol{X}_t;\boldsymbol{p}) \in \mathbb{R}^{n_w \times n}$.

Suppose that the deterministic part of (4.3) has a stable fixed point $\bar{\boldsymbol{x}} = \bar{\boldsymbol{x}}(\boldsymbol{p})$ for a given range of parameter values. Then linearization around the deterministic steady state yields (Kuehn, 2012)

$$M(\boldsymbol{p})\,\mathrm{d}\boldsymbol{X}_t = A(\bar{\boldsymbol{x}};\boldsymbol{p})\boldsymbol{X}_t\,\mathrm{d}t + B(\bar{\boldsymbol{x}};\boldsymbol{p})\,\mathrm{d}\boldsymbol{W}_t, \tag{4.4}$$

where $A(\boldsymbol{x};\boldsymbol{p}) \equiv (D_{\boldsymbol{x}}\boldsymbol{f})(\boldsymbol{x};\boldsymbol{p})$ is the Jacobian matrix and $B(\boldsymbol{x};\boldsymbol{p}) = \boldsymbol{g}(\bar{\boldsymbol{x}};\boldsymbol{p})$. From now on, we drop the arguments of the matrices A, B and M.

In the special case that M is a nonsingular matrix, the equation (4.4) can be rewritten as

$$\mathrm{d}\boldsymbol{X}_t = M^{-1}A\boldsymbol{X}_t\,\mathrm{d}t + M^{-1}B\,\mathrm{d}\boldsymbol{W}_t, \tag{4.5}$$

which represents an n-dimensional Ornstein–Uhlenbeck (OU) process. The corresponding stationary covariance matrix C is determined from the following *Lyapunov equation* (Gardiner, 1985)

$$M^{-1}AC + CA^TM^{-T} + M^{-1}BB^TM^{-T} = 0.$$

This equation can be rewritten as a *generalized Lyapunov equation*

$$ACM^T + MCA^T + BB^T = 0. \tag{4.6}$$

If M is a singular diagonal matrix then (4.5) does not apply. However, if the stochastic part is non-zero only on the part where M is nonsingular (which occurs often when the noise is in the forcing of the flow), then (4.4) can be written as

$$\begin{pmatrix} 0 & 0 \\ 0 & M_{22} \end{pmatrix} \begin{pmatrix} \mathrm{d}\boldsymbol{X}_{t,1} \\ \mathrm{d}\boldsymbol{X}_{t,2} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{X}_{t,1} \\ \boldsymbol{X}_{t,2} \end{pmatrix} \mathrm{d}t + \begin{pmatrix} 0 \\ B_2 \end{pmatrix} \mathrm{d}\boldsymbol{W}_t, \tag{4.7}$$

where M_{22} represents the nonsingular part of M. Consequently, for nonsingular A_{11} we can separate (4.7) into an algebraic part and an explicitly time-dependent part,

$$A_{11}\boldsymbol{X}_{t,1} + A_{12}\boldsymbol{X}_{t,2} = 0, \tag{4.8a}$$

$$M_{22}\,\mathrm{d}\boldsymbol{X}_{t,2} = S\boldsymbol{X}_{t,2}\,\mathrm{d}t + B_2\,\mathrm{d}\boldsymbol{W}_t, \tag{4.8b}$$

where $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement of A, which is well-defined since A_{11} is nonsingular as the Jacobian A has eigenvalues strictly

in the left-half complex plane by the assumption on deterministic stability of $\bar{\boldsymbol{x}}$. Since M_{22} is nonsingular by construction we can find the stationary covariance matrix C_{22} by solving the corresponding generalized Lyapunov equation

$$SC_{22}M_{22}^T + M_{22}C_{22}S^T + B_2B_2^T = 0. \tag{4.9}$$

We remark that (4.9) could alternatively be derived using an epsilon-embedding approach for the differential algebraic equations. In order to find the full covariance matrix we use $C_{ij} = \mathbb{E}[\boldsymbol{X}_{t,i}\boldsymbol{X}_{t,j}^T]$ together with (4.8a) which gives

$$\begin{aligned}
C_{12} &= -A_{11}^{-1}A_{12}\mathbb{E}[\boldsymbol{X}_{t,2}\boldsymbol{X}_{t,2}^T] \\
&= -A_{11}^{-1}A_{12}C_{22}, \\
C_{21} &= -\mathbb{E}[\boldsymbol{X}_{t,2}\boldsymbol{X}_{t,2}^T]A_{12}^TA_{11}^{-T} \\
&= -C_{22}A_{12}^TA_{11}^{-T} = C_{12}^T, \\
C_{11} &= A_{11}^{-1}A_{12}\mathbb{E}[\boldsymbol{X}_{t,2}\boldsymbol{X}_{t,2}^T]A_{12}^TA_{11}^{-T} \\
&= -A_{11}^{-1}A_{12}C_{21}.
\end{aligned}$$

In summary, we can obtain an estimate of the covariance matrix C of the stochastic dynamical system (4.3) by providing the matrices A, B and M and solving the corresponding generalized Lyapunov equation (4.6), or (4.9) in case M is singular. Once the covariance matrix C is computed, the stationary PDF of the approximating OU-process, indicated by $p(\boldsymbol{x})$ follows as (Gardiner, 1985; Kuehn, 2011)

$$p(\boldsymbol{x};\bar{\boldsymbol{x}}) = \frac{1}{(2\pi)^{\frac{n}{2}}} \mid C \mid^{-1/2} e^{-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^TC^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}})}. \tag{4.10}$$

The limitations of this approach are, firstly, that only a PDF estimate is provided valid on a subexponential time scale before large deviations occur and, secondly, that only the local behavior near the steady state and Gaussian stochastic behavior of the system are obtained.

4.1.2 A novel iterative generalized Lyapunov solver

The type of systems of the form (4.6) that we want to solve are typically sparse and have a dimension $n = \mathcal{O}(10^5)$ or larger. Solving systems of this size results in a C that is generally a dense matrix of the same size. This is computationally very expensive in terms of both time and memory. Consequently, one cannot aim to compute the full C but only a low-rank approximation of the form $C \approx VTV^T$. In existing iterative solution methods for low-rank approximations (Kleinman, 1968; Penzl, 1999; Saad, 1990; Simoncini, 2007; Stykel and Simoncini, 2012) the matrix V is usually computed using repetitive products

with B in every iteration, for instance in such a way that it spans the Krylov spaces $\mathcal{K}_m(A, B)$ or $\mathcal{K}_m(A^{-1}, B)$. In practice, however, the matrix B might have many columns, which means that in every iteration of such a method many matrix-vector products have to be performed or many linear systems have to be solved. These operations take up by far the largest amount of time in every iteration, which is why we would like a method that does not expand the search space with the same number of vectors as the number of columns in B.

The solution method we propose is based on a Galerkin projection, and is very similar to the method in Saad (1990). It works by solving projected systems of the form

$$V^T AVTV^T M^T V + V^T MVTV^T A^T V + V^T BB^T V = 0,$$

where C is approximated by a low-rank approximation $\tilde{C} = VTV^T$, and T is generally a dense symmetric matrix which should not be confused with the superscript T denoting transposition. Now if we take $\tilde{A} = V^T AV$, $\tilde{M} = V^T MV$, $\tilde{B} = V^T B$, we get the smaller projected generalized Lyapunov equation

$$\tilde{A}T\tilde{M}^T + \tilde{M}T\tilde{A}^T + \tilde{B}\tilde{B}^T = 0, \tag{4.11}$$

which can be solved by a dense solution routine (Bartels and Stewart, 1972).

A problem that arises when solving generalized Lyapunov equations in an iterative manner is computing an estimate of the residual

$$R = A\tilde{C}M^T + M\tilde{C}A^T + BB^T, \tag{4.12}$$

for some approximate solution $\tilde{C}$. The (matrix) norm of this residual, which is generally a dense matrix, can be used in a stopping criterion. The 2-norm of the residual matrix is equal to its spectral radius which is defined by the absolute value of the largest eigenvalue. Since the residual is symmetric, approximations of the largest eigenpairs can be computed using only a few steps of the Lanczos method Lanczos (1950). Even though we cannot compute R explicitly, it is possible to apply the Lanczos method to determine the eigenpair because only matrix vector products $R\boldsymbol{x}$ are needed, which are evaluated as

$$R\boldsymbol{x} = A(V(T(V^T(M^T\boldsymbol{x})))) + M(V(T(V^T(A^T\boldsymbol{x})))) + B(B^T\boldsymbol{x}).$$

The goal of our method is to compute the matrix V, which we could also view as a search space by considering its columns as basis vectors for a linear subspace of $\mathbb{R}^n$. From now on we assume V to be orthonormalized. We suggest to expand V in every iteration by the eigenvectors associated with the largest eigenvalues of the residual, which we already obtained when computing the norm of the residual. The reasoning behind this will be explained below. Now in every iteration, we solve the projected system (4.11), but because

we expanded our search space (with the largest components of the residual), we hope that the new residual is smaller. The resulting algorithm for solving generalized Lyapunov equations is shown in Algorithm 3. Because of the peculiar choice of vectors to expand our space, we call this method the Residual Approximation-based Iterative Lyapunov Solver (RAILS).

input:	A, B, M	Matrices from (4.6).
	V_1	Initial space.
	m	Dimension increase of the space per iteration.
	l	Maximum number of iterations.
	ϵ	Convergence tolerance.
output:	V_k, T_k	Approximate solution, where $C \approx V_k T_k V_k^T$.

1: Orthonormalize V_1
2: Compute $\tilde{A}_1 = V_1^T A V_1$
3: Compute $\tilde{M}_1 = V_1^T M V_1$
4: Compute $\tilde{B}_1 = V_1^T B$
5: **for** $j = 1, \ldots, l$ **do**
6: Obtain $\tilde{A}_j = V_j^T A V_j$ by only computing new parts
7: Obtain $\tilde{M}_j = V_j^T M V_j$ by only computing new parts
8: Obtain $\tilde{B}_j = V_j^T B$ by only computing new parts
9: Solve $\tilde{A}_j T_j \tilde{M}_j^T + \tilde{M}_j T_j \tilde{A}_j^T + \tilde{B}_j \tilde{B}_j^T = 0$
10: Compute the approximate largest m eigenpairs (λ_p, $\boldsymbol{r}_p$) of the residual R_j using Lanczos
11: Stop if the approximated largest eigenvalue is smaller than ϵ
12: $V_{j+1} = [V_j, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_m]$
13: Re-orthonormalize V_{j+1}

Algorithm 3: RAILS algorithm for the projection based method for solving generalized Lyapunov equations.

4.1.3 Convergence analysis

We will now show why we choose the eigenvectors associated with the largest eigenvalues of the residual. Here we use $\mathrm{orth}(B)$ to denote the orthonormalization of B. We first show that in a special case, V_k as defined in Algorithm 3 spans the Krylov subspace

$$\mathcal{K}_k(A, B) = \{B, AB, \ldots, A^{k-1}B\}.$$

Proposition 4.1.1. *If $M = I$, $B \in \mathbb{R}^{n\times m}$, where m is also the number of vectors we use to expand the space V_k in every iteration and $V_1 = \mathrm{orth}(B)$, then* $\mathrm{Range}(V_k) \subseteq \mathcal{K}_k(A, B)$.

Proof. For $k = 1$ this is true by assumption. Now say that in step k, $(\lambda, \boldsymbol{q})$ is an eigenpair of the residual R_k, and assume that $\mathrm{Range}(V_k) \subseteq \mathcal{K}_k(A, B)$. Then we can write

$$R_k\boldsymbol{q} = \lambda\boldsymbol{q} = AV_k\boldsymbol{q}_1 + V_k\boldsymbol{q}_2 + B\boldsymbol{q}_3$$

where $\boldsymbol{q}_1 = T_kV_k^T\boldsymbol{q}$, $\boldsymbol{q}_2 = T_kV_k^TA^T\boldsymbol{q}$, $\boldsymbol{q}_3 = B^T\boldsymbol{q}$. From this it is easy to see that if we orthonormalize $\boldsymbol{q}$ with respect to V_k, it is only nonzero in the direction of AV_k. Now we take $V_{k+1} = [V_k, Q_k]$, where the columns of Q_k are the eigenvectors associated with the m largest eigenvalues of R_k orthonormalized with respect to V_k. Then

$$\begin{aligned}\mathrm{Range}(V_{k+1}) &\subseteq \mathrm{Range}(V_k) \cup \mathrm{Range}(AV_k)\\ &\subseteq \mathcal{K}_k(A, B) \cup A\mathcal{K}_k(A, B) = \mathcal{K}_{k+1}(A, B).\end{aligned}$$

□

Remark 4.1.1. *In case Q_i has full rank for every $i = 1, \ldots, k$ it is clear that actually the equality* $\mathrm{Range}(V_k) = \mathcal{K}_k(A, B)$ *holds in Proposition 4.1.1. This is also the behavior that we observed on a variety of different test problems.*

From Proposition 4.1.1 and Remark 4.1.1, we see that when we choose m equal to the number of columns of B, RAILS is equivalent to the method in Saad (1990) as long as Q_k has full rank in every iteration. What is important, is that we want to take m much smaller, in which case we assume that eigenvectors associated with the largest eigenvalues of the residual R_k point into the direction of the most important components of AV_k. A similar result holds when we want to look in the Krylov space $\mathcal{K}_k(A^{-1}, A^{-1}B)$.

Proposition 4.1.2. *If $M = I$, $B \in \mathbb{R}^{n\times m}$, where m is also the number of vectors we use to expand the space V_k in every iteration, $V_1 = \mathrm{orth}(A^{-1}B)$ and $V_{k+1} = [V_k, Q_k]$, where Q_k are the m eigenvectors associated with the largest eigenvalues of $A^{-1}R_k$ orthonormalized with respect to V_k, then* $\mathrm{Range}(V_k) \subseteq \mathcal{K}_k(A^{-1}, A^{-1}B)$.

Proof. This can be proved analogously to Proposition 4.1.1. □

We show this result since most other iterative Lyapunov solvers include an operation with A^{-1}. An example is the Extended Krylov method, which looks in the Krylov space $\mathcal{K}_{2k}(A, A^{-k}B)$. We remark that our method, when we start with $V_1 = \mathrm{orth}([B, A^{-1}B])$ and expand with $[Q_k, A^{-1}Q_k]$ is not equivalent to the Extended Krylov method.

We know that if V_k has n orthogonal columns, it spans the whole space, so the solution is in there. To show that our method has finite termination, we argue that the method has converged when the vectors we generate do not have a component perpendicular to V_k, which means that the size of the search space does not increase anymore.

Proposition 4.1.3. *Take $M = I$ and $B \in \mathbb{R}^{n\times m}$. After k steps of RAILS, the residual R_k has an eigenpair $(\lambda, \boldsymbol{q})$ with $\lambda = \|R_k\|_2$. If $\boldsymbol{q} \in \mathrm{Range}(V_k)$, then $V_k T_k V_k^T$ is the exact solution.*

Proof. We have

$$R_k\boldsymbol{q} = \lambda\boldsymbol{q} = AV_kT_kV_k^T\boldsymbol{q} + V_kT_kV_k^TA^T\boldsymbol{q} + BB^T\boldsymbol{q}.$$

Since $\boldsymbol{q} \in \mathrm{Range}(V_k)$ and V_k is orthonormalized, it holds that $\boldsymbol{q} = V_kV_k^T\boldsymbol{q}$. So then

$$\begin{aligned}\lambda\boldsymbol{q} = \lambda V_kV_k^T\boldsymbol{q} &= V_kV_k^TAV_kT_kV_k^T\boldsymbol{q} + V_kT_kV_k^TA^TV_kV_k^T\boldsymbol{q} + V_kV_k^TBB^TV_kV_k^T\boldsymbol{q}\\ &= V_k\tilde{A}_kT_kV_k^T\boldsymbol{q} + V_kT_k\tilde{A}_k^TV_k^T\boldsymbol{q} + V_k\tilde{B}_k\tilde{B}_k^TV_k^T\boldsymbol{q}\\ &= V_k(\tilde{A}_kT_k + T_k\tilde{A}_k^T + \tilde{B}_k\tilde{B}_k^T)V_k^T\boldsymbol{q}\\ &= 0\end{aligned}$$

since the part between brackets is the projected Lyapunov equation that we solved for. This shows us that the residual is zero, so $V_kT_kV_k^T$ is the exact solution. □

Corollary 4.1.1. *From Proposition 4.1.3 it follows that when $m = 1$, the equality $\mathrm{Range}(V_k) = \mathcal{K}_k(A, B)$ holds in Proposition 4.1.1.*

4.1.4 Restart strategy

A problem that occurs in the method described above is that the space V might get quite large. This means that it can take up a lot of memory, but also that the reduced system, for which we use a dense solver, can become large and take up most of the computation time. For this reason we implemented a restart strategy, where we reduce the size of V after a certain number of iterations. Usually, not all directions that are present in V are equally important, so we just want to keep the most important ones. We do this by computing the eigenvectors associated with the largest eigenvalues of VTV^T, which are then used as V in the next iteration of our method. Note that since V is orthonormalized, the nonzero eigenvalues of VTV^T are the same as the eigenvalues of T. The eigenvectors are given by VU, where U are the eigenvectors of T, which makes it quite easy to obtain them.

Besides limiting the size of the reduced problem we have to solve, another advantage is that we reduce the rank of the approximate solution, since we

only keep the most important components. This means that we need less memory to store the solution, but also that when we apply the solution, for instance when computing eigenvalues of the solution, we need fewer operations. To assure that we have a solution that has a minimal size, we also apply a restart when RAILS has converged, after which we let it converge once more. This usually leads to an approximation of lower rank.

A downside of restarting an iterative method is that we lose a lot of convergence properties, like for instance the finite termination property that was shown in Proposition 4.1.3. Since we keep reducing the size of the search space at the time of a restart, it might happen that stagnation occurs. This can be prevented by (automatically) increasing the tolerance of the vectors that we retain during a restart. If we keep doing this repetitively, eventually, the method should still converge.

To implement this restart method, we replaced Line 11 in Algorithm 3 by Algorithm 4.

input: k Iterations after which to restart.
τ Tolerance for the eigenvalues at a restart.

1: Set converged to true if the approximated largest eigenvalue is smaller than ϵ
2: **if** converged and convergence was already achieved earlier **then**
3: **stop**
4: **else if** converged or $j \bmod k = 0$ **then**
5: Compute the eigenpairs $(\lambda_p, \boldsymbol{q}_p)$ of T_j
6: $U = [\,]$
7: **for all** eigenvalues λ_p larger than τ **do**
8: $U = [U, \boldsymbol{q}_p]$
9: $V_{j+1} = V_j U$
10: $\tilde{A}_{j+1} = U^T \tilde{A}_j U$
11: $\tilde{M}_{j+1} = U^T \tilde{M}_j U$
12: $\tilde{B}_{j+1} = U^T \tilde{B}_j$

Algorithm 4: Restart method that replaces Line 11 in Algorithm 3.

4.1.5 Extended generalized Lyapunov equations

To explain the non-Gaussian behavior in sea surface temperature (Sardeshmukh and Sura, 2009) and sea surface height (Sura and Gille, 2010) an additional multiplicative noise term was introduced. The resulting correlated additive and multiplicative (CAM) noise gives rise to linearized stochastic

differential-algebraic equations of the form

$$M(\boldsymbol{p})\,\mathrm{d}\boldsymbol{X}_t = A(\bar{\boldsymbol{x}};\boldsymbol{p})\boldsymbol{X}_t\,\mathrm{d}t + (B(\bar{\boldsymbol{x}};\boldsymbol{p}) + [N_1(\bar{\boldsymbol{x}};\boldsymbol{p})\boldsymbol{X}_t,\ldots,N_m(\bar{\boldsymbol{x}};\boldsymbol{p})\boldsymbol{X}_t])\,\mathrm{d}\boldsymbol{W}_t,$$

where m is the dimension of the noise increment $\mathrm{d}\boldsymbol{W}_t$. The corresponding covariance matrix C can be determined from the *extended generalized Lyapunov equation*

$$ACM^T + MCA^T + BB^T + \sum_{j=1}^{m} N_j C N_j^T = 0, \tag{4.13}$$

where the matrices A and N_j are $n \times n$ matrices with A nonsingular, while B is $n \times m$. Note that these equations with $M = I$ are often referred to as generalized Lyapunov equations, but to avoid confusion with generalized Lyapunov equations of the form (4.6), we refer to these equations as extended Lyapunov equations.

Recently a method was proposed in Shank et al. (2015) for solving extended generalized Lyapunov equations. The method consists of applying a stationary iteration during which $N_j C N_j^T$ terms from the previous iteration are added in every iteration. For this method, we assume that the solution C is positive semidefinite, and in case that M is also positive definite, we require that A is negative semidefinite and that $\rho(\mathcal{M}^{-1}\mathcal{N}) < 1$, where $\mathcal{M}(X) = AXM^T + MXA^T$, $\mathcal{N}(X) = \sum_{j=1}^{m} N_j X N_j^T$ and $\rho(\mathcal{L})$ denotes the spectral radius of the operator $\mathcal{L}$ (Shank et al., 2015). The algorithm is shown in Algorithm 5.

input:	A, B, M, N_j	Matrices from (4.13).
	l	Maximum number of iterations.
	ϵ	Convergence tolerance.
output:	V_k	Approximate solution, where $C \approx V_k V_k^T$.

solve $ACM^T + MCA^T + BB^T = 0$ for $C_1 = V_1 V_1^T$
for $i = 2, 3, \ldots, l$ **do**
  $B_i = [N_1 V_{i-1}, \ldots, N_m V_{i-1}, B]$
  solve $ACM^T + MCA^T + B_i B_i^T = 0$ for $C_i = V_i V_i^T$
  if $\|AC_iM^T + MC_iA^T + \sum_{j=1}^{m} N_j C N_j^T\| < \epsilon$ **then**
    stop

Algorithm 5: Stationary iteration for solving extended generalized Lyapunov equations from Shank et al. (2015).

In Shank et al. (2015) the Extended Krylov method is used as method to solve the generalized Lyapunov equations that arise in every iteration. A downside of this algorithm is that B_i can become so large that the extended Krylov subspace that is computed by this method requires a huge amount of memory and that the orthonormalization and the solution of projected generalized Lyapunov equation (4.11) takes up the majority of the time. As a solution in this problem, B_i was split up into multiple single vectors $\boldsymbol{b}_1^{(i)}, \ldots, \boldsymbol{b}_{2m}^{(i)}$, after which $2m$ generalized Lyapunov equations had to be solved in every iteration.

The advantage of using RAILS instead of EKSM is twofold. Since we only expand with a few vectors in every iteration, the space does not become as large as when using EKSM, and therefore we do not have to split up the problem. Furthermore, RAILS can be restarted, meaning that we can also use V_{i-1} in the stationary iterations as initial guess for V_i. Moreover, since the matrices A and M are the same in every iteration, AV_{i-1}, $V_{i-1}AV_{i-1}^T$, MV_{i-1} and $V_{i-1}MV_{i-1}^T$, do not have to be recomputed. By doing this, we only require a few iterations of RAILS in every stationary iteration to obtain the approximate solution. The restart method from Algorithm 4 can be called after the first solution of the projected generalized Lyapunov equation (4.11) in every stationary iteration to reduced the size of the initial space.

4.2 Problem setting

In this section we discuss the problem setting of the ocean model as described in Section 2.5.2. In Section 4.2.1, we first determine the bifurcation diagram of the deterministic model using pseudo-arclength continuation methods. In the next sections, the case with stochastic freshwater forcing is considered, focusing on validation of the new methods (Section 4.3.1), comparison with other methods (Section 4.3.2), numerical aspects (Section 4.3.3 and Section 4.3.4) and application in a continuation (Section 4.3.5). In Section 4.3.6, we discuss the performance on an example of an extended Lyapunov equation.

4.2.1 Bifurcation diagram

For the deterministic case, we fix the equatorially symmetric surface forcing as described in (2.10).

The equations are discretized on a latitude-depth equidistant $n_x \times n_y \times n_z$ grid using a second-order conservative central difference scheme. An integral condition expressing the overall conservation of salt is also imposed, as the salinity equation is only determined up to an additive constant. The total number of degrees of freedom is $n = 6n_xn_yn_z$, as there are six unknowns per point. The standard spatial resolution used is $n_x = 4, n_y = 32, n_z = 16$ and the solutions are uniform in the zonal direction, with the zonal velocity $u = 0$.

The bifurcation diagram of the deterministic model for parameters as in Table 2.1 is shown in Figure 4.1a. On the y-axis, the sum of the maximum (Ψ^+) and minimum (Ψ^-) values of the meridional streamfunction Ψ is plotted, where Ψ is defined through

$$\frac{\partial \Psi}{\partial z} = v \cos\theta, \; -\frac{1}{r_0 \cos\theta}\frac{\partial \Psi}{\partial \theta} = w. \tag{4.14}$$

For the calculation of the transports, the basin is assumed to have a zonal width of $64°$. The value of $\Psi^+ + \Psi^-$ is zero when the MOC is symmetric with respect to the equator.

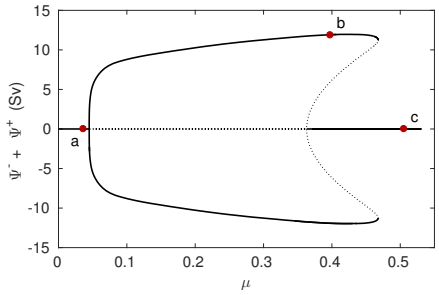
For small values of μ, a unique equatorially anti-symmetric steady MOC exists of which a pattern at location a is shown in Figure 4.1b. This pattern destabilizes at a supercritical pitchfork bifurcation and two asymmetric pole-to-pole solutions appear. An example of the MOC at location b in Figure 4.1b shows a stronger asymmetric overturning with sinking in the northern part of the basin. The pole-to-pole solutions cease to exist beyond a saddle-node bifurcation near $\mu = 0.47$ and both branches connect again with the anti-symmetric solution at a second supercritical pitchfork bifurcation. At this bifurcation, the anti-symmetric solution with equatorial sinking (see MOC at location c in Figure 4.1b) appears which is stable for larger values of μ. The value of μ at the point b, $\mu_b = 0.40$, will be our reference freshwater forcing.

4.2.2 Stochastic freshwater forcing

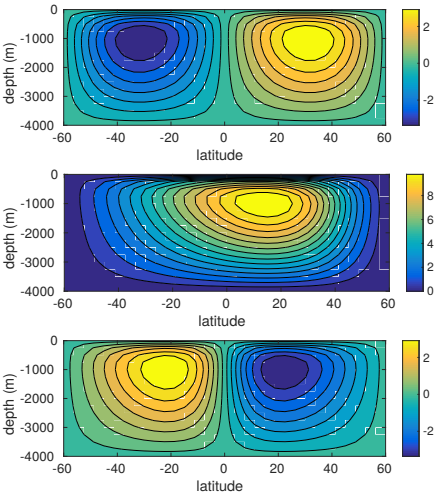
The freshwater forcing is chosen as described in Section 2.5.3. In this case, the noise matrix B in (4.4) simply represents additive noise which is (i) only active in the freshwater component, (ii) only present at the surface, (iii) meridionally uncorrelated (unless stated otherwise), and (iv) has magnitude σ of 10% of the deterministic freshwater forcing amplitude at each latitude θ (see (2.11)).

4.3 Results

Using the available Jacobian A of the deterministic continuation, the mass matrix M, which is a diagonal matrix with non-zero elements in the T and S rows, and the forcing B as described above, we can determine the local probability distribution of a steady state using the generalized Lyapunov equation (4.9). We use grid sizes of $4 \times n_y \times 16$, where n_y is a varying number of grid points in the meridional direction, 16 is the number of grid points in the vertical direction and there are 4 grid points in the zonal direction. Since our model is two-dimensional, both the forcing and the solution will be constant in this direction. For the forcing, this means that B contains n_y vectors with the forcing as described in (2.11).



(a) Bifurcation diagram where a solid line is stable and dashed line is unstable.



(b) The streamfunction at μ_a, μ_b and μ_c respectively.

Figure 4.1: (a) Bifurcation diagram of the deterministic equatorially symmetric 2D MOC model, with the forcing as in (2.10). (b) Streamfunction pattern at μ_a, μ_b and μ_c respectively.

For RAILS, we use the algorithm as described in Section 4.1 and always expand with $m = 3$ vectors per iteration unless stated otherwise. When comparing to other Lyapunov solvers, which were mostly written in Matlab, we use a Matlab implementation of RAILS (Section 4.3.2), but when we solve larger systems, we prefer to use a C++ implementation (all other sections). Computations are performed on one node of Peregrine, the HPC cluster of the University of Groningen. We use this machine to be able to make fair comparisons with other methods, which use large amounts of memory. Peregrine has nodes with 2 Intel Xeon E5 2680v3 CPUs (24 cores at 2.5GHz) and each node has 128 GB of memory. Only one core is used in the results below to be able to make fair comparisons.

The results in this section can be reproduced with the C++ and Matlab code at `https://github.com/Sbte/RAILS`.

4.3.1 Comparison with stochastically forced time forward simulation

In climate sciences it is common use empirical orthogonal functions (EOFs) to investigate the variability of a system Dijkstra (2013); Navarra and Simoncini (2010). The EOFs are the eigenvectors of the covariance matrix, and in the context of principal component analysis are also called the principal components.

The EOFs that belong to the largest eigenvalues of the covariance matrix are the ones that can be used to explain a large part of the variance, which is why these are of interest.

A first check of the correctness of the approximate solution of the generalized Lyapunov equations is obtained by comparing the EOFs and weighted eigenvalues of the covariance matrix that we get from both the Lyapunov solver and a stochastically forced time forward simulation at $\mu = \mu_b$, similar to those performed in Van der Mheen et al. (2013). The EOFs, which are the eigenvectors of the covariance matrix, are used to get an idea of

This time series (for $n_y = 32$) is plotted in Figure 4.2a and shows that Ψ^+ fluctuates around the mean MOC value at μ_b. The patterns of the MOC, the temperature field and the salinity field of both EOF1 and EOF2 are shown in Figure 4.2b and Figure 4.2c.

The eigensolutions of the generalized Lyapunov equation, also for $n_y = 32$, are shown in Figure 4.3. Comparing Figure 4.2 with Figure 4.3, we see that the results from the transient flow computation and the approximate solution of the generalized Lyapunov equation look very similar. Also, the eigenvalues we find with both methods are very similar, as can be seen in Table 4.1. The fact that they are not exactly the same is most likely due the fact that the time series takes a long time to converge to a statistical steady state. Since the eigenvalues are really close nevertheless, we can indeed use RAILS to compute an estimate of the local probability distribution of steady states of the MOC.

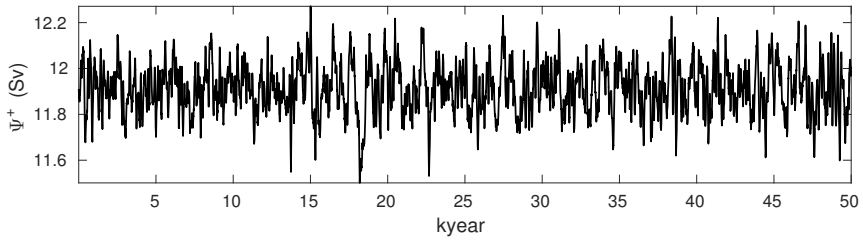
As another check, we use the Bartels–Stewart algorithm (Bartels and Stewart, 1972), which is a dense solver of which the solution time increases with $\mathcal{O}(n^3)$ and the required memory with $\mathcal{O}(n^2)$. The implementation we use is `sb03md` from the SLICOT library (Benner et al., 1999). Results from this method (Table 4.1, also for $n_y = 32$) confirm that our solution method provides correct solutions.

Method	λ_1	λ_2	λ_3	λ_4
RAILS	0.677	0.176	0.078	0.033
Dense Lyapunov	0.677	0.176	0.079	0.033
Time series	0.679	0.170	0.082	0.033

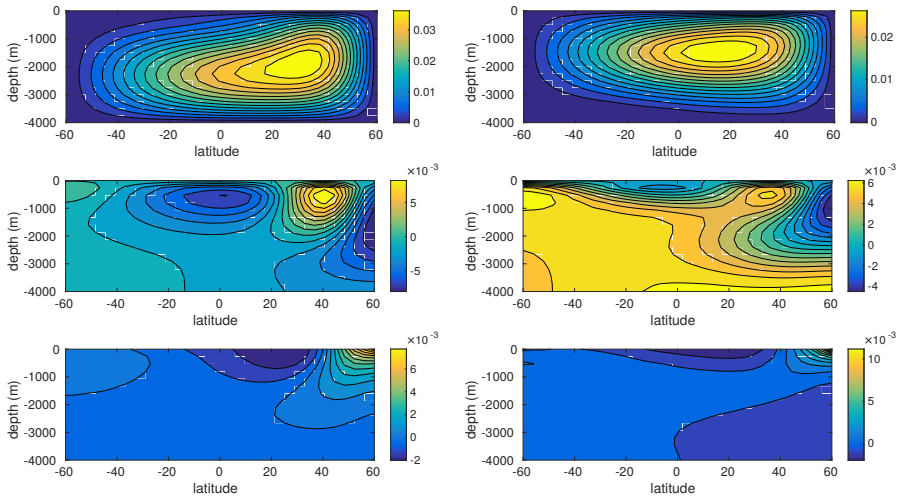
Table 4.1: First four weighted eigenvalues of the covariance matrix. For the RAILS solver $\|R\|_2/\|BB^T\|_2 < 10^{-2}$ was used as stopping criterion.

4.3.2 Comparison with other Lyapunov solvers

In this section, we compare the results of RAILS to those obtained with standard implementations of the Extended Krylov (EKSM) (Simoncini, 2007), Projected Extended Krylov (PEKSM) (Stykel and Simoncini, 2012), Rational Krylov (RKSM) (Druskin and Simoncini, 2011), Tangential Rational Krylov



(a) Time series of the MOC maximum Ψ^+.



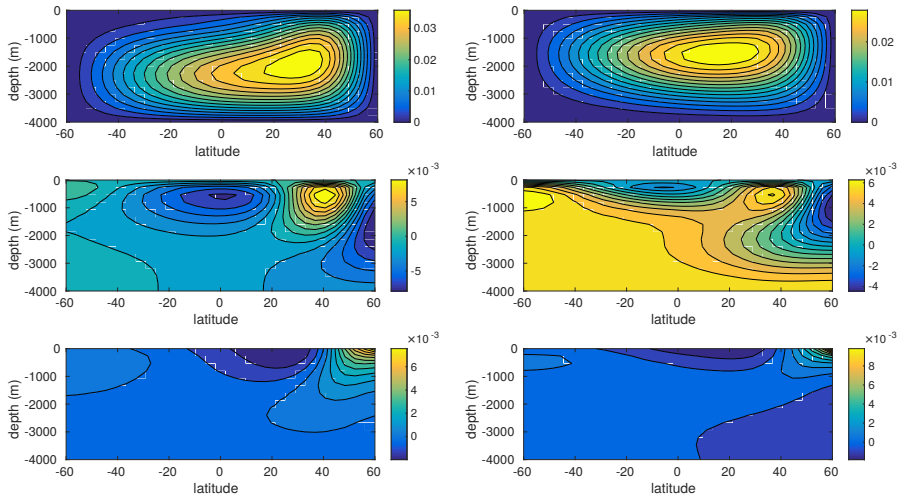
(b) The first EOF.

(c) The second EOF.

Figure 4.2: (a) Time series of the maximum MOC strength Ψ^+ for a 50,000 years simulation of the model for $\mu = \mu_b$ under the freshwater forcing as in (2.11). The variability has a slight negative skewness of -0.05 and a kurtosis of 3.04. (b) Patterns of Ψ (top), isothermals (middle) and isohalines (bottom) for the first EOF of this simulation. (c) Similar to (b) but for the second EOF.

(TKRSM) (Druskin et al., 2014) and Low-rank ADI (LR-ADI) (Penzl, 1999) methods. For the LR-ADI method, results with the heuristic shifts from Penzl (1999) and self-generating shifts based on a Galerkin projection from Benner et al. (2014) are reported. Implementations of the Extended Krylov and Rational Krylov methods were obtained from the website of Simoncini (Simoncini, 2016), whereas the LR-ADI method from M.E.S.S. was used (Saak et al., 2016).

What we want to show is that RAILS can be competitive without requiring linear system solves in every iteration, which all the methods we compare to require. However, convergence properties of the other methods might be better since they use these linear system solves. For this reason, we also add



(a) The first EOF.

(b) The second EOF.

Figure 4.3: (a) Patterns of Ψ (top), isothermals (middle) and isohalines (bottom) respectively for the first EOF obtained by solving the generalized Lyapunov equation. (b) Similar to (a) but for the second EOF.

experiments with our method, where we expand the search space by $S^{-1}\boldsymbol{r}_i$, where $\boldsymbol{r}_i$ are the eigenvectors of the residual associated with the largest eigenvalues. From now on we will refer to this method as Inverse RAILS. To be able to better compare to Extended Krylov, we could also expand our search space by $[\boldsymbol{r}_i, S^{-1}\boldsymbol{r}_i]$, but some preliminary experiments showed that Inverse RAILS performs slightly better.

For Inverse RAILS, Extended/Rational Krylov and ADI based methods, the linear system solves of the form $S\boldsymbol{y} = \boldsymbol{x}$ are implemented by first computing an LU factorization of A using UMFPACK whenever possible (available from `lu` in Matlab), and then solving the system

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{y}} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{x} \end{pmatrix}.$$

This is similar to what has been used in LR-ADI methods in Freitas et al. (2008). Alternatives would be first computing S and then making a factorization of S, which is not feasible since S tends to be quite dense, or using an iterative method where we need repeated applications of S, which includes solving a system with A_{11}. Both alternatives tend to be slower than the method described above if we add up factorization and solution times.

For the (Tangential) Rational Krylov methods we precompute the initial values $s_0^{(1)}$ and $s_0^{(2)}$ as the eigenvalues of S with the smallest and largest real

part. The time it required to compute the eigenvalues is not included in the results.

For the Projected Extended Krylov method, we followed März (1996) for obtaining the spectral projectors that are required and implemented the method according to Stykel and Simoncini (2012). The advantage of this method is that it does not require the Schur complement as we described above. The disadvantage is that instead spectral projectors are required. For this method, we have to compute the projected matrix $V^T AV$ explicitly, and not implicitly as suggested in Stykel and Simoncini (2012) to obtain sufficient accuracy. Without this, the Lyapunov solver that was used for the small projected Lyapunov equation would fail to find a solution. For comparison, we implemented the same projection method that was used in Stykel and Simoncini (2012) also in RAILS, where we chose to expand the basis with $A^{-1}\boldsymbol{r}_i$.

For all methods we use the same stopping criterion, namely that we require the relative residual $\rho = \|R\|_2/\|BB^T\|_2 < \epsilon$, where $\epsilon = 10^{-2}$, which is sufficient for our application. The resulting absolute residual norm will be around 10^{-5} for the MOC problem with $n_y = 32, 64, 128$. We also show the final relative residual in our results. For RAILS we use a random V_1 as initial guess, since this seemed to give slightly better results than taking B as initial guess. Using a random initial guess also shows that even if storing B in a dense way requires too much memory, we are still able to start our method where the other methods cannot. During a continuation run, we can of course do much better than a random initial guess by using the approximate solution space from the previous continuation step. For Inverse RAILS, we use $S^{-1}B_2$ as initial guess as suggested by Proposition 4.1.2. For the same reason, we use $A^{-1}B$ as initial guess for Projected RAILS. Results for the MOC problem with $n_y = 32$ are shown in Table 4.2. Note that B always has n_y columns.

Let us discuss all columns of Table 4.2 separately. The first column contains the rank of the approximate solution, which is more-or-less the same for every method, because we did some post-processing on the non-RAILS methods to only keep eigenvectors belonging to eigenvalues that are larger than a certain tolerance. In this case we took $4 \cdot 10^{-6}$. For RAILS, we do not have to do this since the restart method already takes care of this. The projected variants have a larger rank, because they iterate on the full space instead of only the space belonging to the Schur complement.

Dim shows the maximum dimension of the search space during the iteration. Note that this is much smaller for all variants of RAILS than for almost all of the other methods. For the standard RAILS method this is due to the restart that we use, but we restart after 50 iterations, so both other variants of RAILS do not restart, except in the last step when the method already converged and the rank is being minimized. The only other method that has a small maximum space dimension is the Tangential Rational Krylov method, which also expands the space by only a few vectors in each iteration. This is also the reason why we compare to this method.

Method	Rank	Dim	Its	MVPs	IMVPs	t_s (s)	ρ
RAILS	59	204	217	634	0	8	$9.2 \cdot 10^{-3}$
Inverse RAILS	60	127	34	128	128	7	$9.5 \cdot 10^{-3}$
Projected RAILS	80	133	36	134	134	9	$9.9 \cdot 10^{-3}$
EKSM	60	448	7	224	256	11	$6.2 \cdot 10^{-3}$
PEKSM	81	704	11	704	384	24	$6.3 \cdot 10^{-3}$
RKSM	60	448	13	448	416	64	$9.2 \cdot 10^{-3}$
TRKSM	60	187	16	219	155	46	$9.7 \cdot 10^{-3}$
LR-ADI (heuristic)	60	800	25	0	480	129	$6.1 \cdot 10^{-3}$
LR-ADI (projection)	60	1312	41	0	800	212	$6.4 \cdot 10^{-3}$

Table 4.2: Comparison of different Lyapunov solvers. Rank is the rank of the final approximate solution, Dim is the maximum dimension of the approximation space during the iteration, Its is the number of iterations, MVPs are the number of matrix-vector products, IMVPs are the number of inverse matrix-vector products and t_s is the time required for solution of the Lyapunov equation, which includes the computation of the LU factorization when necessary. For all methods the stopping criterion is a relative residual of 10^{-2}. RAILS was restarted after 50 iterations with a tolerance of $4 \cdot 10^{-6}$ for the eigenpairs that were retained. This is the same tolerance that was used for the other methods to minimize the rank of the approximate solution.

The next column contains the number of iterations. We show this just for reference purposes. Every method has a different notion of what an iteration is, so we can not really compare these values. For instance in this case RAILS expands by 3 vectors per iteration, Extended Krylov by 64, and LR-ADI by 32.

Then we show the MVPs, which are the number of matrix-vector products. For RAILS this is quite high compared to the other methods, but then the advantage is that no linear solves are required, which can be seen if we look at the IMVPs. If we include these in RAILS, which we did in the Inverse RAILS method, we see that it needs far fewer linear solves than the other methods.

The most important part of this table is the solution time t_s, from which we can see that all variants of RAILS are faster than all other methods. The fastest method is Inverse RAILS. However, it is only a bit faster than standard RAILS, at the cost of having to solve linear systems in each iteration. For the solution of the linear systems, an LU factorization was computed beforehand that was utilized by the two variants of RAILS that require a solve, and by the (Projected) Extended Krylov method. The time this takes is included in the solution time. For the LR-ADI and Rational Krylov methods precomputing the LU factorization is not possible, because in each iteration a different shifted linear system has to be solved. Mainly for this reason RAILS is much faster than Tangential Rational Krylov, even though it uses a larger space.

Lastly, we added a column with the explicitly computed final residual norms, which are all between $6 \cdot 10^{-3}$ and 10^{-2}.

We use quite a loose tolerance for the results in Table 4.2, which is sufficient for our purposes. We will continue with this loose tolerance in later experi-

ments. However, with this tolerance, it is quite hard to see the convergence properties of the different methods. In Figure 4.4, we show a convergence plot with a much smaller tolerance: a relative residual of 10^{-8}. We choose to put CPU time on the x-axis, since there is not really any other quantity that represents a similar amount of work for each method.

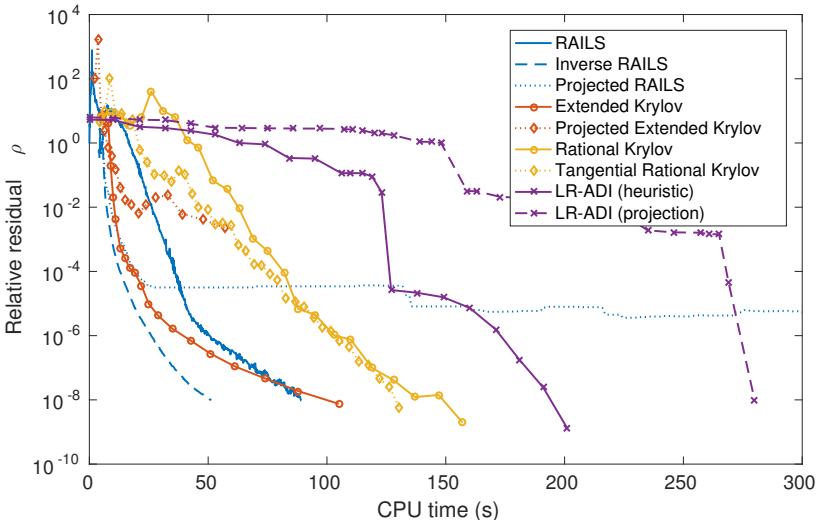


Figure 4.4: Convergence history of the relative residual ρ of all the different methods against CPU time. RAILS was restarted after 15 iterations with a tolerance of 10^{-12} for the eigenpairs that were retained and expanded with 10 vectors per iteration.

What we see in Figure 4.4 is that two methods perform really badly: the Projected Extended Krylov method and Projected RAILS. Projected Extended Krylov actually breaks down due to the small projected Lyapunov equation having eigenvalues with opposite signs and Projected RAILS stagnates. This might be due to round-off errors during the projection, so it seems that here the projected variants are not very well suited for solving our problem. The other methods all converge, but for the Krylov type methods, we clearly see that the cost per iteration increases the further they converge. This is because it becomes increasingly expensive to solve the small projected Lyapunov equation. On the other hand, the LR-ADI methods performed increasingly well for smaller tolerances. They are still really expensive however, due to the shifted solves in every iteration. The fastest method, again, is the Inverse RAILS. It also shows almost monotone convergence, except in the first few steps, and converges very rapidly. Standard RAILS performs very well, but convergence is more erratic. It is, however, promising that a method that only uses matrix-vector products can converge faster than the other existing methods that we tried.

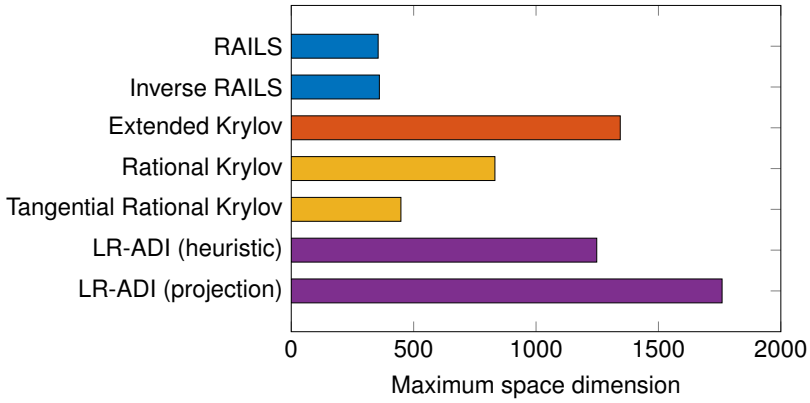


Figure 4.5: Memory usage of the different methods as described in Figure 4.4 in terms of maximum space dimension. Both projected methods were left out of this figure since they did not converge.

In Figure 4.5 we show the memory usage of the methods in Figure 4.4. We left out the methods that did not converge. Here it is clear that RAILS uses the least amount of memory due to the small number of vectors that is used to expand the space and due to the restart strategy.

4.3.3 Numerical scalability

Now we want to show how RAILS behaves when we solve larger systems. We do this by solving the same problem with different grid sizes, namely $4 \times 32 \times 16$, $4 \times 64 \times 16$ and $4 \times 128 \times 16$. For the solution of the generalized Lyapunov equation, when increasing the dimension of our model by a factor 2, the size of the solution increases by a factor 4, since the solution is a square matrix. However, we try to compute a low-rank approximation of the solution, so if the rank of the approximate solution stays the same when we increase the dimension of our model by a factor 2, the size required to store our approximate solution is also increased by a factor 2.

For the MOC problem, if we increase n_y by a factor 2, we know that the number of columns in B also increases by a factor 2, since those two are equal. From this, we would also expect the rank of the approximate solution to increase. Therefore, we first look at a simplified problem where we use $\hat{B} = B \cdot \mathbf{1}$, with $\mathbf{1}$ being a vector of ones, as right-hand side, so $\hat{B}$ is a single vector containing the row sums of the original B. For our test problem, this can be seen as a fully space correlated stochastic forcing. We expect that the rank of the approximate solution stays the same.

From Table 4.3 we see that indeed the rank of the approximate solution does not change when we increase the dimension of the model. However,

the number of iterations to find the approximate solution is dependent on the spectral properties of A. And since we are refining, new high-frequency parts in the approximate solution will be amplified strongly in the residual evaluation and appear also in the search space. This will slow down the convergence process as can also be seen in Table 4.3.

Size	Rank	Dim	Its	MVPs	t_s (s)	t_m (S)
32	12	41	231	223	3	1.5
64	13	42	350	338	13	10
128	13	42	536	518	58	52

Table 4.3: Performance of RAILS for different grid sizes with $\hat{B} = B \cdot \mathbf{1}$. The grids are of size $4 \times n_y \times 16$ where n_y is the size in the first column. Rank is the rank of the final approximate solution, Dim is the maximum dimension of the approximation space during the iteration, Its is the number of iterations, MVPs are the number of matrix-vector products, t_s is the time required for solution of the Lyapunov equation and t_m is the cost of the matrix-vector product. The stopping criterion is a relative residual of 10^{-2}. RAILS was restarted after 30 iterations with a tolerance of 10^{-5} for the eigenpairs that were retained. We expanded the space with 1 vector in each iteration.

We are of course also interested in the increase of the solution time. There are five factors that influence this: the length of the vectors that span our basis, the number of vectors that span our basis, the number of iterations, the cost of the matrix-vector product, and the cost of solving the projected Lyapunov equation. First of all, the length of the vectors that span our basis increases by a factor 2 when the dimension of the problem is increased by a factor 2, so we can expect a factor 2 in time increase from this. Secondly, the rank of the approximate solution and the dimension of the search space does not increase. This means that also the cost of solving the projected Lyapunov equation does not increase, so we do not expect a solution time increase from this. Then we have the number of iterations, which does seem to increase by a factor 1.5. And finally we have the cost of the matrix-vector product, which we listed in an extra column of Table 4.3. Note that this is actually a matrix-vector product with the Schur complement S which requires a linear solve with the matrix block A_{11}, which is relatively expensive. If we subtract the cost of the matrix-vector product, we would expect a factor 3 in time cost increase from the increased vector length and the increased number of iterations. In Table 4.3 we observe roughly a factor 2. This being less than 3 might be due to higher efficiency of vector operations for larger vectors.

Going back to the original problem, where we take B as the original stochastic forcing, we expect the solution time to increase by a larger factor, since the projected Lyapunov equation solve and the vector operations become more costly, because the maximum search space dimension and the rank of the approximate solution now do increase. In Table 4.4 we see that the number of iterations from $n_y = 32$ to $n_y = 64$ increases by a factor 1.5

again, so we would expect the solution time without matrix-vector products to increase by a factor 3, which is true. The number of iterations from $n_y = 64$ to $n_y = 128$ increases by a factor 2, so we would expect the time cost to increase by a factor 4, and this is also true. This is because the solution of the projected Lyapunov equation is still relatively cheap for the problems we have here, since those projected equations have at most size 247×247. For problems with a larger maximum search space dimension, the cost of solving this projected Lyapunov equation would become dominant.

Size	Rank	Dim	Its	MVPs	t_s (s)	t_m (s)
32	59	198	233	682	7	2
64	86	223	340	997	31	17
128	147	247	652	1915	178	115

Table 4.4: Performance of RAILS for different grid sizes with B as the original stochastic forcing. The grids are of size $4 \times n_y \times 16$ where n_y is the size in the first column. Rank is the rank of the final approximate solution, Dim is the maximum dimension of the approximation space during the iteration, Its is the number of iterations, MVPs are the number of matrix-vector products, t_s is the time required for solution of the Lyapunov equation and t_m is the cost of the matrix-vector product. The stopping criterion is a relative residual of 10^{-2}. RAILS was restarted after 50 iterations with a tolerance of 10^{-5} for the eigenpairs that were retained.

4.3.4 Towards a 3D model

Ultimately, we want to do computations on a full 3D model. To illustrate what will happen for a full 3D model, we include some results where we take the forcing in such a way that it is not zonally averaged, but it is taken such that there is no correlation in the zonal direction. The new forcing can be seen as a diagonal matrix, where all salinity nodes at the surface have a nonzero on the diagonal, and the rest of the matrix is zero. We can write our new forcing, using Matlab notation, as $\hat{B} = \mathrm{diag}(B \cdot \mathbf{1})$ where $\hat{B}$ is the new forcing and B is the original forcing. This means that there are $4 \cdot n_y - 1$ nonzero columns in $\hat{B}$.

We choose to compare RAILS to Extended Krylov, which was the only method that came close to RAILS in the earlier experiments in terms of time, and to the Tangential Rational Krylov method, since this was the only method that came close in terms of maximum space dimension. Since Extended Krylov does not perform well anymore when the projected system becomes really large, which would be the case for this problem, we split $\hat{B}$ into multiple parts. We can do this because for our problem, the right-hand side $\hat{B}\hat{B}^T$

can be written as

$$\hat{B}\hat{B}^T = \sum_{i=1}^{4\cdot n_y - 1} \hat{B}_i \hat{B}_i^T,$$

where $\hat{B}_i$ is the ith column of $\hat{B}$. Since the rank of the approximate solution is highly dependent on the number of vectors in $\hat{B}$, we expect the maximum space dimension that is needed, and therefore also the size of the projected system, to decrease. After splitting $\hat{B}$, we separately solve the Lyapunov equations with these new right-hand sides, of which we reduce the rank of the approximate solution separately to save memory. Afterwards, we merge the low-rank solutions back into one low-rank solution, which is the approximate solution for the system with $\hat{B}$. We report results with the number of parts that resulted in the least amount of solution time. The optimal number of parts that we found was 4, so that is three of size n_y and one of size $n_y - 1$. The maximum space dimension that we report is the maximum space size of solving one part plus the number of vectors that are required to store the reduced approximate solution of the previous solves. The results with the 3D forcing are shown in Table 4.5.

Method	Size	Rank	Dim	Its	MVPs	IMVPs	t_f (s)	t_s (s)
EKSM	32	172	763	6	1114	1241	3	43
	64	309	1353	5	1979	2234	15	278
TRKSM	32	177	681	16	808	554	0	109
	64	314	1232	17	1487	977	0	645
RAILS	32	166	312	252	739	0	0	16
	64	276	419	406	1189	0	0	82
	128	563	699	651	1912	0	0	584

Table 4.5: Comparison of different Lyapunov solvers for different grid sizes with $\hat{B} = \mathrm{diag}(B \cdot \mathbf{1})$. The grids are of size $4 \times n_y \times 16$ where n_y is the size in the second column. Rank is the rank of the final approximate solution, Dim is the maximum dimension of the approximation space during the iteration, Its is the number of iterations, MVPs are the number of matrix-vector products, IMVPs are the number of inverse matrix-vector products t_f is the time required for computing the LU factorization and t_s is the time required for solution of the Lyapunov equation. For all methods the stopping criterion is a relative residual of 10^{-2}. RAILS was restarted after 50 iterations with a tolerance of 10^{-6} for the eigenpairs that were retained. The tolerance that was used in Extended Krylov and Tangential Rational Krylov to minimize the rank of the approximate solution in such a way that the residual was still below the tolerance was set to 10^{-6} and $4 \cdot 10^{-7}$ for the 32 and 64 problems respectively.

We see that in this case RAILS can solve systems much faster and with much less memory than the Extended Krylov and Tangential Rational Krylov methods, even when applying the additional trick that we described above to

reduce the memory usage of the Extended Krylov method. It is also clear that both computing and applying the inverse becomes a real problem for these kinds of systems. RAILS, however, does not need an inverse, and employs a restart strategy to reduce the space size, which is why it performs so much better. The reason why we do not show any results with the Extended Krylov and Tangential Rational Krylov methods for the $4 \times 128 \times 16$ problem is that we did not have enough memory to do these computations, since we chose to not employ an iterative solver.

4.3.5 Continuation

Now that we have shown the performance of RAILS compared to other methods, we show how it can be useful in the context of continuation. The main idea is that since RAILS can be restarted and since we can choose the number of vectors that we use to expand the space, we can easily start from the low-rank solution of another generalized Lyapunov equation, in our case the solution that was determined in the previous continuation step. We refer to this method as Recycling RAILS. We use the settings from Section 4.3.2, starting at point b from the bifurcation diagram in Figure 4.1, and moving towards the bifurcation with constant step size $\mathrm{d}s = 0.05$. We do this for 20 continuation steps on the problem of size $4 \times 32 \times 16$.

		RAILS		Recycling RAILS	
Step	Par	Its	$t_s(s)$	Its	$t_s(s)$
1	0.403	210	7.3	210	7.0
2	0.406	242	8.4	49	2.0
3	0.410	245	8.6	35	1.3
4	0.413	248	8.8	39	1.5
5	0.415	250	9.1	39	1.5
10	0.428	254	9.1	36	1.3
15	0.439	285	10.1	50	2.2
20	0.448	304	11.1	54	2.3

Table 4.6: Performance of RAILS during 20 steps of the continuation process with fixed step size $\mathrm{d}s = 0.05$. Here Recycling RAILS is RAILS restarted from the solution of the generalized Lyapunov equation at the previous continuation step. Par is the actual parameter value, Its is the number of iterations, t_s is the time required for solution of the Lyapunov equation. For all methods the stopping criterion is a relative residual of 10^{-2}. RAILS was restarted after 50 iterations with a tolerance of 10^{-6} for the eigenpairs that were retained.

The results are shown in Table 4.6. It is clear that reusing the solution from the previous continuation step, which we do in Recycling RAILS, is of great benefit. Both the number of iterations and the computational time are reduced by roughly a factor of 6. The local variations in the number of iterations can

mostly be explained by the random initial guess that is used for computation of the eigenvectors of the residual. We also see a global pattern of the generalized Lyapunov equations becoming harder to solve when getting closer to the saddle-node bifurcation, which is expected. This does not seem to affect the efficiency of the recycling method.

4.3.6 Extended Lyapunov equations

Since usage of CAM noise for the MOC problem that is discussed in this section has not yet been investigated, we will not look at the performance of RAILS on a MOC related problem. Instead, we will look at Example 2 from Shank et al. (2015), which is described in more detail in Damm (2008). The example consists of a central finite difference discretization of the 2D convection-diffusion operator $L(\boldsymbol{x}) = \Delta\boldsymbol{x} - \boldsymbol{x}_y$ on $\Omega = (0,1)^2$ with Robin boundary conditions $\boldsymbol{n} \cdot \nabla\boldsymbol{x} = \frac{1}{2}u_j(\boldsymbol{x} - 1)$ on an increasing number of sides of the domain, and Dirichlet boundary conditions $\boldsymbol{x} = 0$ on the rest of the boundary. Here the control variable u_j represents the heat transfer coefficient on boundary j. The number of Robin boundary conditions that are imposed determines the number of extra terms of the extended Lyapunov equation. The matrices in (4.13) are given by

$$
\begin{aligned}
A &= I \otimes D + D \otimes I - I \otimes G + \frac{2}{h}\left(\sum_{j=1}^{m} N_j\right), \\
N_1 &= \frac{1}{2h}(E_1 \otimes I), \qquad N_2 = \frac{1}{2h}(E_n \otimes I), \\
B_1 &= -\frac{1}{2h}(E_1 \otimes \mathbf{1}), \quad B_2 = -\frac{1}{2h}(E_n \otimes \mathbf{1}), \\
B &= [B_1, \ldots, B_m], \qquad M = I,
\end{aligned}
$$

where $D \in \mathbb{R}^{n\times n}$ is the central finite difference discretization of the 1D Laplace operator, $G \in \mathbb{R}^{n\times n}$ is the central finite difference discretization of the derivative, $E_j = \boldsymbol{e}_j\boldsymbol{e}_j^T$ with canonical unit vector $\boldsymbol{e}_j \in \mathbb{R}^n$, $\mathbf{1} \in \mathbb{R}^n$ is a vector of all ones, $n = 70$, and $h = 1/(n+1)$. The solution C may be interpreted as a controllability Gramian of the system Damm (2008). The implementation of this example which was used in Shank et al. (2015), along with the implementation of the stationary iterations, can be found at Simoncini (2016). We found that the example was not implemented correctly, however, and therefore the results we find here may differ from the results reported in Shank et al. (2015).

In Table 4.7 and Table 4.8 we show the performance of RAILS when expanded with $\boldsymbol{r}_i$ (RAILS), $A^{-1}\boldsymbol{r}_i$ (Inverse RAILS) and $[\boldsymbol{r}_i, A^{-1}\boldsymbol{r}_i]$ (Extended RAILS), EKSM, and EKSM where B_i is split into single vectors. The first table contains results from the case with one Robin boundary condition, the second table contains results with two Robin boundary conditions. In Table 4.7, we

see that Extended RAILS and Inverse RAILS perform very well in terms of matrix-vector products, the number of linear system solves and the required CPU time. The rank of the final solution of all methods is comparable. The reason why the number of matrix-vector products and the number of linear system solves is so much smaller than those of EKSM is that we restart from the solution of the previous stationary iteration.

Method	Rank	MVPs	IMVPs	t_s (s)	ρ
RAILS	66	1084	0	58.0	$8.1 \cdot 10^{-9}$
Inverse RAILS	68	264	264	5.3	$6.4 \cdot 10^{-9}$
Extended RAILS	68	274	137	3.8	$6.4 \cdot 10^{-9}$
EKSM	69	2461	2656	67.6	$6.1 \cdot 10^{-9}$
EKSM + splitting	69	1992	2186	8.7	$7.7 \cdot 10^{-9}$

Table 4.7: Comparison of RAILS and EKSM with and without splitting of B_i on the example described in this section with one Robin boundary condition. Rank is the rank of the final approximate solution, MVPs are the number of matrix-vector products, IMVPs are the number of inverse matrix-vector products and t_s is the time required for solution of the extended Lyapunov equation. For all methods the stopping criterion is a relative residual of 10^{-8}. The number of eigenvectors of the residual in RAILS was set to be the same as the number of columns of the matrix B_i. RAILS was restarted after 10 iterations.

In Table 4.8, we see again that Extended RAILS performs well in all categories. Because we use two Robin boundary conditions in this case, an extra extended term is added, and therefore also the number of columns of B_i of the stationary iteration is larger. As we also noticed for the MOC problem, this is something for which RAILS performs increasingly well compared to other methods. We notice that even though the number of linear system solves that is required for Extended RAILS is more than a factor 20 lower than that of EKSM with splitting, it is only a factor 3 faster. This is because the problem that is solved here is relatively easy, meaning that solving a linear system is relatively cheap. Therefore all instances of RAILS are actually dominated by the cost of computing the eigenvectors. We expect that RAILS performs even better for more complex systems like an ocean-climate model with CAM noise.

4.4 Summary and Discussion

We have presented a new method for numerically determining covariance matrices, and hence we can compute probability density functions (PDFs) near steady states of deterministic PDE systems which are perturbed by noise. This method enables the application of the approach suggested by Kuehn (2011, 2015) to larger systems of SPDEs with algebraic constraints and exploits the structure of the generalized Lyapunov equation in the computa-

Method	Rank	MVPs	IMVPs	t_s (s)	ρ
RAILS	123	1793	0	246.3	$9.6 \cdot 10^{-9}$
Inverse RAILS	123	416	416	9.2	$9.7 \cdot 10^{-9}$
Extended RAILS	123	464	232	8.0	$9.6 \cdot 10^{-9}$
EKSM	123	4429	4865	197.6	$9.6 \cdot 10^{-9}$
EKSM + splitting	134	4652	5090	23.5	$8.7 \cdot 10^{-9}$

Table 4.8: Comparison of RAILS and EKSM with and without splitting of B_i on the example described in this section with two Robin boundary conditions. Rank is the rank of the final approximate solution, MVPs are the number of matrix-vector products, IMVPs are the number of inverse matrix-vector products and t_s is the time required for solution of the extended Lyapunov equation. For all methods the stopping criterion is a relative residual of 10^{-8}. The number of eigenvectors of the residual in RAILS was set to be the same as the number of columns of the matrix B_i. RAILS was restarted after 10 iterations.

tions. This approach fits nicely within purely deterministic numerical bifurcation computations (Keller, 1977) and methods to tackle the SPDEs directly (Sapsis and Lermusiaux, 2009). It provides a Gaussian PDF which is valid under linearized dynamics near the deterministic steady state.

Our new algorithm to solve generalized Lyapunov equations is based on a projection method, where solutions are found iteratively by using a set of subspaces V_k. The key new aspects are (i) that at each iteration k, the subspace V_k is expanded with the eigenvectors corresponding to the largest eigenvalues of the residual matrix R, orthogonalized with respect to V_k and itself, and (ii) the restart strategy that is used to keep the space dimension low. We applied this method to a test problem consisting of a quasi two-dimensional model of the Atlantic Ocean circulation. Our method, RAILS, outperforms both Krylov and ADI based methods (Simoncini, 2007; Druskin and Simoncini, 2011; Stykel and Simoncini, 2012; Druskin et al., 2014; Kleinman, 1968; Penzl, 1999) for this test problem.

In practice, one has to provide the Jacobian matrix A of a deterministic fixed point and the matrix B representing the stochastic forcing in order to solve for the stationary covariance matrix C. In a matrix-based pseudo-arclength continuation method the Jacobian is available since a Newton–Raphson method is used (Dijkstra et al., 2014). The mass matrix M is readily available from the model equations. The method hence enables us to determine the continuation of local PDFs in parameter space. In particular, the projection approach provides subspaces, which may be re-used directly or by computing predictors along continuation branches.

The availability of the new method opens several new directions of analysis. In one set of applications, A is varied whereas the structure of B is fixed. This typically corresponds to varying a bifurcation parameter and analyzing the corresponding changes in PDF (i.e. covariance matrix C) and transition

probabilities (i.e. overlap of PDFs of two steady states) as the steady states (i.e. A) change and a bifurcation point is approached (Kuehn, 2011). For example, this could be used in the test problem considered in this study in order to investigate the scaling law in PDF near the saddle-node bifurcation on the branch of pole-to-pole solutions. In this way, the critical slowdown near a tipping point can be studied (Van der Mheen et al., 2013).

We have shown that for these types of problems, RAILS performs especially well since it can be restarted using the approximate solution of the previous continuation step, which results in rapid convergence. We also applied RAILS in a stationary iteration to solve extended generalized Lyapunov equations. This proved to be very efficient and may be used when CAM noise is applied instead of the additive noise that we used for the MOC.

Another interesting application is to fix A (or a set of As that exists for fixed parameter values) and vary B. In this case one steady state is fixed (or two in a regime of two steady states) and the impact of different B ("noise products") on the PDF/covariance matrix C (and possibly transition probabilities) is investigated. Different B can be constructed by changing (a) the dynamical component which is stochastically active (freshwater flux, wind, heat flux), (b) the magnitude, and (c) the pattern (i.e. the cross-correlation structure and the spatial weighting of the auto-covariances). Results from these computations will show the effect of the representation of small scale processes (the 'noise') on the probability density function (under the restriction of linear dynamics).

For the ocean circulation problem it would be interesting to compare the different impacts of freshwater flux versus wind stress noise e.g. on the PDF of the MOC or heat transports. Moreover, regarding the wind stress and (c) one could construct B based on EOFs of the atmospheric variability, as for example considered in Dijkstra et al. (2008). Note that for studying the effect of wind-stress noise, a full three-dimensional model, as developed in De Niet et al. (2007) is required. In order to construct B from more than one EOF one has to generalize the stochastic forcing to a sum of OU processes. In that case we can use the linearity of the generalized Lyapunov equation and solve for each term separately and combine later on.