Horizon 2020

ROMSOC

Reduced Order Modelling, Simulation and Optimization of Coupled systems

Benchmark cases

Deliverable number: D5.3

Version 0.1

Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 765374

Project Acronym:	ROMSOC
Project Full Title:	Reduced Order Modelling, Simulation and Optimization of Coupled systems
Call:	H2020-MSCA-ITN-2017
Topic:	Innovative Training Network
Type of Action:	European Industrial Doctorates
Grant Number:	765374

Editor:	Andrés Prieto, ITMATI
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	01/07/2021
Actual Delivery Date	01/07/2021
Number of pages:	11
Keywords:	Benchmarks, Model hierarchies
Authors:	Onkar Jadhav, TUB XXX - YYY, Institution XXX - YYY, Institution XXX - YYY, Institution
Peer review:	ZZZ - Institution

Abstract

Based on the multitude of industrial applications, benchmarks for model hierarchies will be created that will form a basis for the interdisciplinary research and for the training programme. These will be equipped with publically available data and will be used for training in modelling, model testing, reduced order modelling, error estimation, efficiency optimization in algorithmic approaches, and testing of the generated MSO/MOR software. The present document includes a detailed description of the computer implementation of these benchmarks involving not only the required publically available data but also the used software packages, libraries and any other relevant information, which guarantee a fully reproducibility of the reported numerical results.

Contents

1 Introduction **1**

2 Parametric Model Order Reduction Approach **1**

2.1 Adaptive Greedy Sampling Technique 2

3 Benchmark Case 1 **2**

3.1 Description of Input Data 2

3.2 Step-by-Step Procedure 2

4 Benchmark Case 2 **6**

4.1 Description of Input Data 6

4.2 Step-by-Step Procedure 6

5 Summary **10**

List of Acronyms

TUB	Technische Universität Berlin
ITMATI	Technological Institute of Industrial Mathematics

1 Introduction

It is essential to be aware of the financial risk associated with the invested product. Underlying risk factors such as interest rates have a direct influence on the invested asset. Two prominent risk analysis examples are packaged retail investment and insurance products (PRIIPs) and Monte Carlo or historical Value at Risk (VaR). The financial instruments are evaluated via the dynamics of short-rate models, based on the convection-diffusion-reaction partial differential equations (PDEs). The choice of the short-rate model depends on the underlying financial instrument. This benchmark report shows an example of a floater with caps and floors under the one-factor Hull-White model. These models are calibrated based on several thousand simulated yield curves that generate a high dimensional parameter space $\mathcal{P}$. The first benchmark case address this problem and presents a detail methodology for yield curve simulation. The source code for the yield curve simulation is given the file `YieldCurveSimulation.m`. The parameter calibration based on these yield curves is given in file `1FHWcalibration.nb`. In short, to understand the risks associated with a financial product, one has to perform several thousand computationally demanding simulations of the model with a high dimensional parameter space, which require efficient algorithms. The work provides a method to perform such a computationally costly task as fast as possible but with a reliable outcome based on a model order reduction approach.

2 Parametric Model Order Reduction Approach

We employ the projection based MOR technique to solve the full order model (FOM)

$$A(\rho(t))V^{n+1} = B(\rho(t))V^n, \qquad V(0) = V_0, \tag{1}$$

where the matrices $A(\rho) \in \mathbb{R}^{M\times M}$, and $B(\rho) \in \mathbb{R}^{M\times M}$ are parameter dependent matrices. $V \in \mathbb{R}^M$ is a high dimensional state vector. t is the time variable $t = [0, T]$. ρ is a group of model parameters. We need to solve the system (1) for at least 10,000 parameter groups ρ generating a parameter space $\mathcal{P}$ of $10000 \times m$, where m is total number of tenor points of the yield curve. The idea is to project a high dimensional space onto a low dimensional subspace, Q as

$$\bar{V}^n = QV_d^n, \tag{2}$$

where $Q \in \mathbb{R}^{M\times d}$ is a reduced order basis with $d \ll M$, V_d is a vector of reduced coordinates, and $\bar{V} \in \mathbb{R}^M$ is the solution obtained using the reduced order model. We get the reduced model as

$$\begin{aligned} Q^T A(\rho)QV_d^{n+1} &= Q^T B(\rho)QV_d^n, \\ A_d(\rho)V_d^{n+1} &= B_d(\rho)V_d^n, \end{aligned} \tag{3}$$

where the matrices $A_d(\rho) \in \mathbb{R}^{d\times d}$ and $B(\rho) \in \mathbb{R}^{d\times d}$ are the parameter dependent reduced matrices. We now solve this reduced model for the entire parameter space $\mathcal{P}$.

We obtain the parametric reduced order model (3) based on a proper orthogonal decomposition method (POD) [1, 2]. POD generates an optimal order orthonormal basis Q which serves as a low dimensional subspace in the least square sense for a given set of computational data. The selection of a data set plays an important role, and most prominently obtained by the *method of snapshots* [3]. In this method, the optimal basis is computed based on a set of state solutions. These state solutions are known as snapshots and are calculated by solving the HDM for some parameter values. The quality of the parametric reduced model mainly depends on the selection of training parameters $\rho_1, \rho_2, \ldots, \rho_k$ for which the snapshots are computed. Thus, it necessitates defining an efficient sampling technique for the high dimensional parameter space. We present a classical as well as adaptive greedy sampling approach to select these training parameters. The greedy sampling method introduced in [4] is proven to be an efficient method for sampling a high dimensional parameter space in the framework of MOR.

2.1 Adaptive Greedy Sampling Technique

The greedy sampling technique selects the parameter vectors at which the error between the reduced order model and the full model is maximum. Further, we compute the snapshots using these parameter vectors so that we can obtain the best suitable reduced order basis Q. However, the computation of a relative error $\|\epsilon_{\mathrm{RM}}(.,\rho)\| = \|V(.,\rho) - \bar{V}(.,\rho)\|/\|V(.,\rho)\|$ between the full model and the reduced model is expensive. Thus, usually, the error is replaced by the error bounds or the relative residual for the approximate solution $\bar{V}$.

For a high dimensional parameter space, it is not feasible to compute the error estimate for each parameter vector. Thus, we run the greedy sampling algorithm for a pre-defined set of parameter vectors $\hat{\mathcal{P}} \subseteq \mathcal{P}$. The selection of this subset could be random. However, the random selection of a parameter set may not contain the parameter vector corresponding to the most significant error. Therefore, instead of selecting $\hat{\mathcal{P}}$ randomly, we propose to select it adaptively. We construct a surrogate model $\bar{\varepsilon}$ to approximate the error estimator ε over the entire parameter space. Further, we use the surrogate model to locate the parameter vectors $\hat{\mathcal{P}}$. The detailed developed adaptive greedy approach is presented in [5].

3 Benchmark Case 1

The first proposed benchmark case is to validate the numerical methods implemented for the simulation of yield curves and parameter calibration. The implemented numerical methods are following the guidelines provided by the PRIIP regulations.
We perform a principal component analysis on the collected historical data to ensure that the simulation results in a consistent curve. Further, using the principal components corresponding to their maximum energies, we calculate the consistent interest rates and composed them into a matrix called as the matrix of returns. Finally, we obtain the simulated yield curve by applying the bootstrapping procedure on the matrix of returns. To fulfill the regulations demand, we perform the bootstrapping process for at least 10,000 times. The detailed procedure can be found in the PRIIPs regulations. In this work, we implement the parameter calibration as described in [6]. We use the inbuilt UnRisk functions for the parameter calibration. UnRisk PRICING ENGINE integrates the pricing and calibration engines into Mathematica.

3.1 Description of Input Data

To perform yield curve simulation, we collect historical interest rate data. We construct a data matrix $A \in \mathbb{R}^{n\times m}$ of the collected historical interest rates data, where each row of the matrix forms a yield curve, and the column represents the m tenor points, which are the different contract lengths of an underlying instrument. For example, we have collected the daily interest rate data at $m \approx 20$ tenor points in time over the past five years, and since a year has approximately 260 working days, we obtain $n \approx 1306$ observation periods. This data matrix is read as A is the MATLAB file `YieldCurveSimulation.m`.

3.2 Step-by-Step Procedure

3.2.1 Yield Curve Simulation

The yield curve simulation procedure is well described in [5]. The regulations demand to take the natural logarithm of the ratio between the interest rate at each observation period and the interest rate at the preceding period. To ensure that we can form the natural logarithm, we need that all elements of the data matrix A are positive which is achieved by adding a correction term a as shown in the `.m` file `corr_A1 = A+a`. Then we calculate the log returns over each period and store them into a new matrix $\hat{A} = \hat{A}_{ij} \in \mathbb{R}^{n\times m}$ as

$$\hat{A}_{ij} = \frac{\ln(\bar{A}_{ij})}{\ln(\bar{A}_{i-1,j})}.$$

We calculate the arithmetic mean μ_j of each column of the matrix $\hat{A}$,

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n}\hat{A}_{ij},$$

subtract μ_j from each element of the corresponding jth column of $\hat{A}$ and store the obtained results in a matrix $\bar{\bar{A}}$ with entries $\bar{\bar{A}}_{ij} = \hat{A}_{ij} - \mu_j$. This corrected returns are stored in the variable `BarBarA` in `.m` file. We then compute the singular value decomposition of the matrix $\bar{\bar{A}}$ to generate the matrix of returns stored in the variable `Matreturns`. The selection of singular vectors is based on their energy levels. To do so, we plot the singular values of the data matrix A. MATLAB `Figure(1)` and `Figure(2)` show these plots of monotonously decreasing singular values. We then select the first p right singular vectors corresponding to the p largest singular values. The regulations suggest selecting the first three singular vectors.

We then perform *bootstrapping*, where large numbers of small samples of the same size are drawn repeatedly from the original data set. According to the PRIIP regulations, for the yield curve simulation we have to perform a bootstrapping procedure for at least 10 000 times. The standardized KID also has to include the recommended *holding period*, i.e., the period between the acquisition of an asset and its sale. The time step in the simulation of yield curves is typically one observation period. If H is the recommended holding period in days, e.g., $H \approx 2600$ days, then there are H observation periods in the recommended holding period. The source code for the bootstrapping simulation is addressed in the subsection `%% Simulations` in the `YieldCurveSimulation.m` file. For each such observation period, we select a random row from the matrix of returns M_R, i.e., altogether H random rows, and construct a matrix $[\chi_{ij}] \in \mathbb{R}^{H\times m}$ from these selected rows. In our benchmark example $H = 10$ years $\approx$ 2600 days. Then we sum over the selected rows of the columns corresponding to the tenor point j, i.e.,

$$\bar{\chi}_j = \sum_{i=1}^{h}\chi_{ij},\ j = 1,\cdots,m.$$

In this way, we obtain a row vector $\bar{\chi} = [\bar{\chi}_1\ \bar{\chi}_2\ \cdots\ \bar{\chi}_m] \in \mathbb{R}^{1\times m}$. The final simulated yield rate y_j at tenor point j is then the rate $\bar{d}_{nj}$ of the last observation period at the corresponding tenor point j, multiplied by the exponential of $\bar{\chi}_j$, adjusted for any shift γ used to ensure positive values for all tenor points, and adjusted for the forward rate so that the expected mean matches current expectations.

The forward rate between time points t_k and t_ℓ starting from a time point t_0 is given as

$$r_{k,\ell} = \frac{R(t_0,t_\ell)(t_\ell - t_0) - R(t_0,t_k)(t_k - t_0)}{t_\ell - t_k},$$

where t_k and t_ℓ are measured in years and $R(t_0,t_k)$ and $R(t_0,t_\ell)$ are the interest rates available from the data matrix for the time periods (t_0,t_k) and (t_0,t_ℓ), respectively. The forward rate calculation is presented in the section `%% Forward Rates` in the `.m` file. Thus, the final simulated yield curve between time points t_k and t_ℓ is given by

$$y(t_\ell) = \bar{d}_{k,\ell}\exp(\bar{\chi}_\ell) - \gamma + r_{k,l},\ \ell = 1,\cdots,m, \tag{4}$$

and the simulated yield curve from the calculated simulated returns is given by

$$y = [y_1\ y_2\ \cdots\ y_m].$$

We then perform the bootstrapping procedure for at least $s = 10\,000$ times and construct a simulated yield curve matrix

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \vdots & \vdots \\ y_{s1} & \cdots & y_{sm} \end{bmatrix} \in \mathbb{R}^{s\times m}. \tag{5}$$

The simulated yield curves are stored in the variable `Sim_return` and in percentage form `Sim_returnPerc`. These simulated yield curves then can be plotted using the `plot` function of the MATLAB.

```
%% Plot output for output of simulated yield curves.
tic
figure(3)
X = [1 5 10 15 20 25 30 40 50]; % xticks
T1 = TenorPoints;               % tenor points of yield curves
plot(T1,Sim_returnPerc')
set(gca,'FontSize',14)
set(gcf, 'Position',  [100, 100, 800, 500])
ax.FontSize = 14;
xticks(X);
xlabel('Terms in years','fontsize',18,'interpreter','latex');
ylabel('Simulated yield curves','fontsize',18,'interpreter','latex');
grid on
toc
```

3.2.2 Calibration of the Parameter $a(t)$

For a zero coupon bond $B(t,T)$ maturing at time T, based on the Hull-White model, one obtains a closed-form solution, see [6], as

$$B(t,T) = \exp\{-r(t)\Gamma(t,T) - \Lambda(t,T)\}, \tag{6}$$

where $\kappa(t) = \int_0^t b(s)ds = bt$, since b is assumed constant,

$$\Gamma(t,T) = \int_t^T e^{-\kappa(t)}dt,$$
$$\Lambda(t,T) = \int_t^T \left[e^{\kappa(v)}a(v)\left(\int_v^T e^{-\kappa(z)}dz\right) - \frac{1}{2}e^{2\kappa(v)}\sigma^2\left(\int_v^T e^{-\kappa(z)}dz\right)^2\right]dv.$$

Here we have again considered that σ is constant.

To perform the calibration, we use as input data i) the initial value of $a(0)$ at $t = 0$, ii) the zero-coupon bond prices, iii) the constant value of the volatility σ of the short-rate $r(t)$, and iv) the constant value b each for all maturities T_m, $0 \le T_m \le T$, where T_m is the maturity at the mth tenor point. Then we compute $\kappa(t)$ from $\frac{\partial}{\partial T}\kappa(T) = \frac{\partial}{\partial T}\int_0^T b(s)ds = b$ and use

$$\frac{\partial}{\partial T}\Gamma(0,T) = e^{-\kappa(T)}$$

to compute $\Gamma(t)$.

Then, for $0 \le T_m \le T$, we get

$$
\begin{aligned}
\frac{\partial}{\partial T}\Lambda(0,T) &= \int_0^T \Bigg[e^{\kappa(v)} a(v) e^{-\kappa(T)} \\
&\qquad - e^{2\kappa(v)}\sigma^2 e^{-\kappa(T)} \left(\int_v^T e^{-\kappa(z)} dz \right) \Bigg] dv, \\
e^{\kappa(T)} \frac{\partial}{\partial T}\Lambda(0,T) &= \int_0^T \left[e^{\kappa(v)} a(v) - e^{2\kappa(v)}\sigma^2 \left(\int_v^T e^{-\kappa(z)} dz \right) \right] dv, \\
\frac{\partial}{\partial T}\left[e^{\kappa(T)} \frac{\partial}{\partial T}\Lambda(0,T) \right] &= e^{\kappa(T)} a(T) - \int_0^T e^{2\kappa(v)}\sigma^2 e^{-\kappa(T)} dv, \\
e^{\kappa(T)}\left[e^{\kappa(T)} \frac{\partial}{\partial T}\Lambda(0,T) \right] &= e^{2\kappa(T)} a(T) - \int_0^T e^{2\kappa(v)}\sigma^2 dv, \\
\frac{\partial}{\partial T}\left[e^{\kappa(T)}\left[e^{\kappa(T)} \frac{\partial}{\partial T}\Lambda(0,T) \right]\right] &= \frac{\partial a(T)}{\partial T} e^{2\kappa(T)} + 2a(T) e^{2\kappa(T)} \frac{\partial}{\partial T}\kappa(T) - e^{2\kappa(T)}\sigma^2, \\
\frac{\partial}{\partial T}\left[e^{\kappa(T)}\left[e^{\kappa(T)} \frac{\partial}{\partial T}\Lambda(0,T) \right]\right] &= \frac{\partial a(T)}{\partial T} e^{2\kappa(T)} + 2a(T) e^{2\kappa(T)} b(T) - e^{2\kappa(T)}\sigma^2.
\end{aligned}
$$

The simulated yield $y(T)$ at the tenor point T is given by [7]

$$
y(T) = -\ln B(0,T), \tag{7}
$$

and from (7) and (6) we obtain $\Lambda(0,T) = [y(T) - r(0)\Gamma(t)]$. In this way, for $a(t)$ we get the ordinary differential equation (ODE)

$$
\frac{\partial}{\partial t} a(t) e^{2\kappa(t)} + 2a(t)\cdot b \cdot e^{2\kappa(t)} - e^{2\kappa(t)}\sigma^2 = \frac{\partial}{\partial t}\left[e^{\kappa(t)} \left[e^{\kappa(t)} \frac{\partial}{\partial t}(y(t) - r(0)\Gamma(0,t)) \right]\right],
$$

which we solve numerically with the given initial conditions. If we approximate $a(t)$ by a piecewise constant function with values $a(i)$ which change at the tenor point i, then we obtain a linear system

$$
L\alpha = F,
$$

for the vector $\alpha = [a(i)]$, where L is lower triangular matrix with non-zero diagonal elements. In [8] it is noted that the integral equation Λ is of the first kind with L2 kernel and a small perturbation (noise) in the market data that are used to obtain the yield curves leads to large changes in the model parameter $a(t)$. This means that the problem to compute $a(t)$ from the data is an ill-posed problem and for this reason we determine the vector α via Tikhonov regularization as

$$
\alpha_\mu^\delta = \text{argmin}\|L\alpha - F^\delta\|^2 + \mu\|\alpha\|^2, \tag{8}
$$

where α_μ^δ is an approximation to α, μ is the regularization parameter, $\delta = \|F - F^\delta\|$ is the noise level, and $\mu\|\alpha\|^2$ is a regularization term. We then solve the optimization problem (8) to obtain an approximation of the parameter $a(t)$ via the commercial software *UnRisk PRICING ENGINE* [9] in Mathematica. The source code for the calibration of the model parameter $a(t)$ is in the file `1FHWcalibration.nb`. This program takes the simulated yield curves stored in the file `SimulatedYieldCurves.xlsx` as an input. The parameters $b = 0.015$ and $\sigma = 0.006$ are kept constant. Note that this file demands a special license to the commercial software UnRisk Pricing Engine, which is initiated by `Needs["UnRisk UnRiskFrontEnd"]`. The program returns the 10 000 deterministic drift rates $a(t)$ for 10 000 simulated yield curves. For the floater example, we need parameter values only until the 10Y tenor point (maturity of the floater). Henceforth, we consider the simulated yield curves with only the first 12 tenor points `t = `$\{$`0, 65, 260, 520, 780, 1040, 1300, 1560, 1820, 2080, 2340, 2600`$\}$. The parameter $a(t)$ is stored in the variable `DDF1FL` after calibrating the Hull-White model using the command `MakeGeneralHullWhiteModel`. Then we export these piecewise constant parameters into a file using the command `Export["DD_1FHWModel_10000.xlsx", DDF1FL]`. By providing the simulated yield curve, the UnRisk pricing function returns the calibrated parameter $a(t)$ for that yield curve. Based on $s = 10\,000$ different simulated yield curves, we obtain s different piecewise constant parameters $a_\ell(t)$, which change their values $\alpha_{\ell,i}$ only at the m tenor points. We incorporate these in a matrix

$$\mathcal{A} = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1m} \\ \vdots & \vdots & \vdots \\ \alpha_{s1} & \cdots & \alpha_{sm} \end{bmatrix}. \tag{9}$$

4 Benchmark Case 2

The main task of this research is to evaluate the financial instrument based on short-rate models. Second benchmark case is to verify the implemented MOR algorithm. We use a finite difference method for simulating the convection-diffusion-reaction PDE. The projection-based MOR approach has been implemented, and the reduced-order basis is obtained using the proper orthogonal decomposition approach with the classical and adaptive greedy sampling methods.

The benchmark case addresses both the algorithms and presents MATLAB code for the same.

4.1 Description of Input Data

The classical greedy sampling and the adaptive greedy sampling have been used to locate the training parameters required to generate the reduced basis. The classical greedy sampling algorithm takes the parameter space $\mathcal{P}$, maximum number of iterations I_{max}, maximum number of parameter groups c, and tolerance ε_{tol} as inputs. The variable `a` defines the matrix composed of parameter vectors $a(t)$ in the section `%% Model Parameters` in `.m` file. `a` takes the calibrated parameters stored in the excel file `DD_1FHWModel_10000.xlsx` obtained in the benchmark case 1. The user can define the number of maximum parameter groups required to initiate the algorithm, the maximum number of iterations, and the tolerance using the variables `c`, `Imax`, `max_tol`, respectively. For example, in our sample code, the values are `c = 20`, `Imax = 10`, `max_tol = `10^{-5}.

The adaptive greedy sampling algorithm also takes the parameter space $\mathcal{P}$, maximum number of iterations I_{max}, maximum number of parameter groups c, and tolerance e_{tol}^{max} as inputs. Along with other inputs, the user needs to specify the number of adaptive candidates to complete the surrogate model loop. The user can define these inputs using the variables `c`, `Imax`, `max_tol`, and `ck` respectively. For example, in our sample code, the values are `c = 20`, `Imax = 10`, `ck = 10`, and `max_tol = `10^{-8}.

4.2 Step-by-Step Procedure

4.2.1 Classical Greedy Sampling

The algorithm is initiated by selecting a parameter group ρ_1 from the parameter set $\mathcal{P}$ and computing a reduced basis Q_1. The first group of parameter is shown using variables `a(1:1,:), b, sigma`. It is necessary to note that the choice of a parameter group to obtain the initial reduced basis Q_1 does not affect the final result. One can initiate the greedy sampling algorithm with any parameter group. Nonetheless, for the simplicity of computations, we select the first parameter group ρ_1 from the parameter space $\mathcal{P}$ to obtain Q_1.

The greedy iteration are indexed with `Niter` and runs for `Imax` iterations. Furthermore, a pre-defined parameter set $\hat{\mathcal{P}}$ of cardinality c has been chosen randomly from the set $\mathcal{P}$ and shown with variable `NpSel` in

MATLAB file. At each point of $\hat{\mathcal{P}}$, the algorithm determines a reduced model with reduced basis Q_1 and then computes error estimator values, $\varepsilon(\rho_j)_{j=1}^{c}$. The parameter group in $\hat{\mathcal{P}}$ at which the error estimator is maximal is then selected as the optimal parameter group ρ_I.

`max` function is used to locate this parameter group. The error estimator values obtained by solving the reduced models are stored in `Error`. The following command locates the parameter group that maximizes the error estimator

```
[max_Error(1,(Niter-1)),max_idErr(1,(Niter-1))] = max(Error(:))
```

It also gives the location of the parameter group with the pre-defined parameter set `NpSel`. Then the full model is simulated for this parameter group and the snapshot matrix $\hat{V}$ is updated. Finally, a new reduced basis is obtained by computing a truncated singular value decomposition of the updated snapshot matrix. The truncated SVD is based on the randomized algorithm, which is given by the `function RandSVD`

```
[U1,S1,V1] = RandSVD(SnapShots,RankS(1,Niter-1));
```

These steps are then repeated for I_{max} iterations or until the maximal value of the error estimator is lower than the specified tolerance ε_{tol}.

```
if max_Error(1,(Niter-1)) < max_tol
    Q = Q_Niter;
    break
end
```

The truncated SVD computes only the first k columns of the matrix Φ. The optimal projection subspace Q then consists of d left singular vectors ϕ_i known as *POD modes*. The dimension d of the subspace Q is chosen such that we get a good approximation of the snapshot matrix. According to [10], large singular values correspond to the main characteristics of the system, while small singular values give only small perturbations of the overall dynamics. The relative importance of the ith POD mode of the matrix $\hat{V}$ is determined by the *relative energy* Ξ_i of that mode

$$\Xi_i = \frac{\Sigma_i}{\sum_{i=1}^{k} \Sigma_i} \tag{10}$$

If the sum of the energies of the generated modes is 1, then these modes can be used to reconstruct a snapshot matrix completely [11]. In general, the number of modes required to generate the complete data set is significantly less than the total number of POD modes [12]. Thus, a matrix $\hat{V}$ can be accurately approximated by using POD modes whose corresponding energies sum to almost all of the total energy. Thus, we choose only d out of k POD modes to construct $Q = [\phi_1 \cdots \phi_d]$ which is a parameter independent projection space. `%% Selection of the reduced dimension` describes the procedure to select the first d left singular vectors based on their relative energies that generate the reduced basis `Q_d`.

```
Eg = diag(S1)./sum(diag(S1));
%
for ii = 1:length(Eg)
    SumEg = sum(Eg(1:ii,1))*100;
    if SumEg > 99.99
        d = ii;
        break
    end
end
%
Q_d = Q_Niter(:,1:d);
```

The function `CGPlots` is used to plot the results obtained using the classical greedy sampling approach. The progression of the maximal and average residuals with each iteration of the greedy algorithm is presented in `Figure(1)`. The projection error associated with the reduced basis is calculated using (14) is plotted in `Figure(3)`.

$$\epsilon_{\mathrm{POD}}^{\mathrm{AG,CG}} = \frac{1}{iT}\sum_{i=1}^{iT} \|V_i(\rho_I) - \sum_{iT=1}^{\ell} (V_i(\rho_I)\phi_k)\phi_k\|_2^2 = \sum_{\ell=d+1}^{iT} \Sigma_\ell^2. \tag{11}$$

4.2.2 Adaptive Greedy Sampling

To overcome drawbacks of the classical greedy sampling approach, we implement the adaptive sampling approach which selects the parameter groups adaptively at each iteration of the greedy procedure, using an optimized search based on surrogate modeling. The surrogate model constructs an approximate model for the desired simulation output, i.e., in our case, the error estimator. Then the surrogate model is trained on some error estimator values. This training data is obtained by solving the reduced model for some random parameter groups. Subsequently, we can deploy this trained surrogate model to perform simulations instead of the original model. Since the single evaluation of the surrogate model is faster than the original model, performing thousand of evaluations for the given high dimensional parameter space is no longer a problem. In short, the surrogate modeling methods make those expensive computations economical.

There are different choices to build a surrogate model: regression models [13], decision trees [14], machine learning and artificial neural networks [13, 15], and kriging models [16]. In this work, we present two options based on the principal component regression model (PCR) and the K-nearest neighbor (KNN) algorithm. The PCR algorithm is presented in the function `PCRSM`, while the KNN algorithm is in the function `KNN`.

The adaptive greedy sampling algorithm utilizes the designed surrogate model to locate optimal parameter groups adaptively at each greedy iteration $i = 1, ..., I_{max}$. The greedy iteration are indexed with `i` and runs for `Imax` iterations. The first few steps of the algorithm resemble the classical greedy sampling approach. First the parameter group ρ_1 is selected from the parameter space $\mathcal{P}$ and the reduced basis Q_1 is constructed. Furthermore, the algorithm randomly selects c_0 parameter groups (defined using `c0` in `.m` file) and constructs a temporary parameter set $\hat{\mathcal{P}}_0 = \{\rho_1, ..., \rho_{c_0}\}$ and shown with variable `NpSel` in MATLAB file. For each parameter group in the parameter set $\hat{\mathcal{P}}_0$, the algorithm determines a reduced order model and computes an array of corresponding residual errors $\hat{\varepsilon}_0 = \{\varepsilon(\rho_1), ..., \varepsilon(\rho_{c_0})\}$. Then a surrogate model for the error estimator $\bar{\varepsilon}$ is constructed based on the estimator values $\{\varepsilon(\rho_j)\}_{j=1}^{c_0}$ using either `PCRSM` or `KNN`. The user can choose the surrogate model as desired. The obtained surrogate model is then simulated for the entire parameter space $\mathcal{P}$. Furthermore, we locate c_k parameter groups corresponding to the first c_k maximal values of the surrogate model. This selection is made using the maxk function as follows

```
[MAX_SM(jjj,:),MAX_SMid(jjj,:)] = maxk(abs(SM),ck)
```

which determines the first `ck` values of the surrogate model `SM` and the corresponding parameter groups `a(MAX_SMid',:)`. We then construct a new parameter set $\hat{\mathcal{P}}_k = \{\rho_1, ..., \rho_k\}$ composed of these c_k parameter groups as

```
NpSel2 = cat(1,NpSel,a(MAX_SMid',:)).
```

The algorithm determines a reduced model for each parameter group within the parameter set $\hat{\mathcal{P}}_k$ and obtains an array of error estimator values $\hat{\varepsilon}_k = \{\varepsilon(\rho_1), ..., \varepsilon(\rho_{c_k})\}$. Furthermore, we concatenate the set $\hat{\mathcal{P}}_k$ and the set $\hat{\mathcal{P}}_0$ to form a new parameter set $\hat{\mathcal{P}} = \hat{\mathcal{P}}_k \cup \hat{\mathcal{P}}_0$. Let $e_{\mathrm{sg}} = \{\hat{\varepsilon}_0 \cup \cdots \cup \hat{\varepsilon}_k\}$ be the set composed of all the error estimator values available at the kth iteration. The algorithm then uses this error estimator set e_{sg} to build a new surrogate model. The quality of the surrogate model increases with each iteration as we get more error estimator values. This process is repeated until the cardinality of the set $\hat{\mathcal{P}}$ reaches c, giving

$$\hat{\mathcal{P}} = \hat{\mathcal{P}}_0 \cup \hat{\mathcal{P}}_1 \cup \hat{\mathcal{P}}_2 \cup \cdots \cup \hat{\mathcal{P}}_K.$$

Finally, the optimal parameter group ρ_I which maximizes the error estimator is extracted from the parameter set $\hat{\mathcal{P}}$. The following command locates the parameter group that maximizes the error estimator

```
[max_Error(1,(Niter-1)),max_idErr(1,(Niter-1))] = max(Error2(:))
```

It also gives the location of the parameter group with the pre-defined parameter set `NpSel2`.

4.2.2.1 Convergence of the Adaptive Greedy Sampling Algorithm In the classical greedy sampling approach, we used the residual error to observe the convergence of the algorithm, which estimates the relative error between the full model and the reduced model. However, in the adaptive greedy POD algorithm, we use an approximate model $\hat{\epsilon}_{\mathrm{RM}}$ (`RE_appx`) for the relative error ϵ_{RM} (`RE`) as a function of the residual error ε to monitor the convergence. This is more accurate than using only the residual error as a convergence criterion.

At each greedy iteration, the algorithm solves one full model for the optimal parameter group ρ_I to update the snapshot matrix and construct a new reduced basis. We can utilize this information for the construction of an approximate relative error model. We solve two reduced models for the optimal parameter group, one before and another after updating the reduced basis (Q_{bef}, Q_{aft}), and obtain respective error estimator values $\varepsilon^{bef}(\rho_I)$ and $\varepsilon^{aft}(\rho_I)$ (`resd_b, resd_a`). Then we calculate the relative errors $\epsilon^{bef}_{\mathrm{RM}}$, $\epsilon^{aft}_{\mathrm{RM}}$ (`RE_b, RE_a`) between the full model and the two reduced models constructed before and after updating the reduced basis. Here superscript bef and aft denote the before and after updating the reduced basis. In this way, we get a set of error values E_p at each greedy iteration that we can use to construct an approximate model for the relative error based on the error estimator.

$$E_p = \{(\epsilon^{bef}_{\mathrm{RM},1}, \varepsilon^{bef}_1) \cup (\epsilon^{aft}_{\mathrm{RM},1}, \varepsilon^{aft}_1), \dots, (\epsilon^{bef}_{\mathrm{RM},i}, \varepsilon^{bef}_i) \cup (\epsilon^{aft}_{\mathrm{RM},i}, \varepsilon^{aft}_i)\} \tag{12}$$

The error set is generated as follows in the code

```
        RE_U = cat(2,RE_b,RE_a);
        resd_U = cat(2,resd_b,resd_a);
        Ep = cat(RE_u, resd_U);
```

We then construct an approximate model for the relative error using the `polyfit` command based on the error estimator as

$$\log(\hat{\epsilon}_{\mathrm{RM},i}) = \gamma_i \log(\varepsilon) + \log\tau. \tag{13}$$

Setting $\mathcal{Y} = \log(\hat{\epsilon}_{\mathrm{RM}}), \mathcal{X} = \log(\varepsilon)$ and $\hat{\tau} = \log(\tau)$ we get

$$\mathcal{Y} = \gamma_e \mathcal{X} + \hat{\tau},$$

where γ_e is the slope of the linear model and $\hat{\tau}$ is the intercept with the logarithmic axis $\log(y)$.

Similar to the classical greedy sampling program, `%% Selection of the reduced dimension` describes the procedure to select the first d left singular vectors based on their relative energies that generate the reduced basis `Q_d`.

```
        Eg = diag(S1)./sum(diag(S1));
        %
        for ii = 1:length(Eg)
            SumEg = sum(Eg(1:ii,1))*100;
            if SumEg > 99.99
                d = ii;
                break
            end
        end
        %
        Q_d = Q_Niter(:,1:d);
```

The function `AGPlots` is used to plot the results obtained using the classical greedy sampling approach. The progression of the maximal and average residuals with each iteration of the greedy algorithm is presented in `Figure(1)`. The projection error associated with the reduced basis is calculated using (14) is plotted in `Figure(3)`.

$$\epsilon_{\mathrm{POD}}^{\mathrm{AG,CG}} = \frac{1}{iT}\sum_{i=1}^{iT} \|V_i(\rho_I) - \sum_{iT=1}^{\ell}(V_i(\rho_I)\phi_k)\phi_k\|_2^2 = \sum_{\ell=d+1}^{iT}\Sigma_\ell^2. \tag{14}$$

After each greedy iteration, we get more data points in the error set E_p, which increases the accuracy of the error model, provided that the linear model assumption is validated. `Figure(4)` illustrate with the obtained results that this linear model assumption is sufficient to achieve an acceptable approximate relative error model. We then use this error model in the adaptive greedy sampling to monitor the convergence of the algorithm.

5 Summary

The benchmark cases present the procedure for yield curve simulation along with the developed model order reduction framework for a numerical example of a floater instrument with caps and floors. We solve this instrument using the robust one-factor Hull-White model.

All computations are carried out on a PC with 4 cores and 8 logical processors at 2.90 GHz (Intel i7 7th generation). We used MATLAB R2018a for the yield curve simulations, FDM, and the model reduction. The numerical method for the yield curve simulations is tested with real market based historical data. Market data are available from market data providers like Thomson Reuters, Bloomberg, and several others. We obtained this data from MathConsult within their UnRisk Omega datasets [9]. The daily interest rate data are collected at 21 tenor points in time over the past 5 years, where each year has 260 working days, so there are 1300 observation periods. We have used the inbuilt UnRisk tool for the parameter calibration, which is well integrated with Mathematica (version used: Mathematica 11.3). Further, we used calibrated parameters for the construction of Hull-White models.

References

[1] A. Chatterjee, "An introduction to the proper orthogonal decomposition," *Curr. Sci.*, vol. 78, pp. 808–817, 2000.

[2] G. Berkooz, P. Holmes, and J. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annu. Rev. Fluid Mech.*, vol. 25, no. 1, pp. 539–575, 1993.

[3] L. Sirovich, "Turbulence and the dynamics of coherent structures. Part I: coherent structures," *Quart. Appl. Math.*, vol. 45, no. 3, pp. 561–571, 1987.

[4] C. Prud'homme, D. Rovas, K. Veroy, L. Machiels, Y. Maday, A. Patera, and G. Turinici, "Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods," *J. Fluids Eng.*, vol. 124, no. 1, pp. 70–80, 2001.

[5] A. Binder, O. Jadhav, and V. Mehrmann, “Model order reduction for the simulation of parametric interest rate models in financial risk analysis,” *J. Math. Industry*, vol. 11, pp. 1–34, 2021.

[6] S. Shreve, *Stochastic Calculus and Finance*, 1st ed. New York, US: Springer-Verlag, 2004.

[7] M. Aichinger and A. Binder, *A Workout in Computational Finance*, 1st ed. West Sussex, UK: John Wiley and Sons Inc., 2013.

[8] H. Engl, “Calibration problems–an inverse problems view,” *Wilmott*, pp. 16–20, 2007.

[9] MathConsult, “Calibration of interest rate models,” MathConsult GmbH, Linz, Austria, Report, 2009.

[10] M. Rathinam and L. Petzold, “A new look at proper orthogonal decomposition,” *SIAM J. Numer. Anal.*, vol. 41, no. 5, pp. 1893–1925, 2003.

[11] M. Williams, P. Schmid, and J. Kutz, “Hybrid reduced-order integration with proper orthogonal decomposition and dynamic mode decomposition,” *SIAM J. Multiscale Model. Simul.*, vol. 11, no. 2, pp. 522–544, 2013.

[12] R. Pinnau, “Model reduction via proper orthogonal decomposition,” in *Model Order Reduction: Theory, Research Aspects and Applications*, W. Schilders, H. van der Vorst, and J. Rommes, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 95–109.

[13] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 1st ed. New York, NY: Springer-Verlag, 2013.

[14] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.

[15] A. Forrester, A. Sóbester, and A. Keane, *Engineering design via surrogate modelling: a practical guide*, 1st ed. West Sussex, UK: John Wiley and Sons Inc., 2008.

[16] D. Jones, “A taxonomy of global optimization methods based on response surfaces,” *J. Global Optim.*, vol. 21, no. 4, pp. 345–383, 2001.

ROMSOC

The ROMSOC project

June 25, 2021

ROMSOC-D5.3-0.1

Horizon 2020