578.15999pt

1 Introduction

This document highlights simple usages of the bayesian-framework package used for state estimation, parameter estimation and model selection.. All the functions for the examples are in the file ”functions.cpp”

2 Example 1

2.1 Description

We first look at a bivariate Gaussian distribution denoted by

$$p(\vec{x}) = \frac{1}{2\pi} \exp\left(-\frac{\vec{x}\vec{x}^T}{2}\right) \tag{1}$$

where the covariance is the identity matrix and the mean is zero. For that first example we compute the evidence using Chib-Jeliazkov and using a quadrature approach. We first need to create the likelihood function in the form of

$$p(\vec{x}) \propto -\frac{\vec{x}\vec{x}^T}{2} \tag{2}$$

```
extern ”C” double likelihood
```

The necessary files are in the Example-1 folder.

3 Example 2

3.1 Description

In the first example we perform model selection on a mass-spring-damper system where the mass, the damping and the forcing is known. The stiffness is unknown. The two proposed models are described by

$$\mathcal{M}_1: \quad m\ddot{u} + c\dot{u} + ku + k_c u^3 = A\cos(2\pi f t) \tag{3}$$

$$\mathcal{M}_2: \quad m\ddot{u} + c\dot{u} + ku + k_c u^3 = A\cos(2\pi f t) + \sigma W(t) \tag{4}$$

with measurements

$$d_j = u(t_j) + \varepsilon_j \tag{5}$$

where $\varepsilon_j \sim (0, \gamma^2)$. To solve the ODE described by Eq. (3) there is therefore no need to perform state estimation. The SODE described by Eq. (4) is stochastic and linear. State estimation will performed by the Kalman filter. In the code, the Kalman filter is not implemented. EKF will be use instead. For a linear system, EKF is the same as the KF.

3.2 Set-up

Now that we have the discretized formulation to simulate the mass-spring-damper system, we need to code it in functions.cpp. The first 3 lines indicates that we are using the *statespace* object and we will be using armadillo to represent the vectors.

The next step is to code Eq. (**??**)

You can notice that it is not exactly the same function as Eq. (**??**). The function implemented in the code is of a mass-spring-damper system that is also perturbed by white noise. However, if $\sigma_g = 0$, it becomes equivalent to the deterministic system. It will be use to both model Eqs. (**??**) and(**??**)

3.2.1 Generating the measurements

Since we do not have experimental data, we will create our own data. To that end, we will use Eq. (5). The code to generate the data is in the file *gen_data.cpp*. Depending on what state you are measuring, you will need to change the code accordingly. At the moment only the first state is measured. This may change in the future. The function *generateData* will create the measurements based on a specified noise to signal ratio (NSR).

The next step is to compile the code. To do so we first compile *functions.cpp*. In the terminal, type

```
make func
```

Notice the creation of the file *libf.so*. The next step is to compile the generating data file

```
make data
```

Notice the creation of *data.out*.

We have yet to specify the timestep, for how long the system should be simulated, how many data points, the parameters of the system, etc ... To do this we will write the configuration file *genmodels.cfg*. In that file we can specify as many models as we want. For example, the same model could be specified for various set of parameters. Each entry corresponds to one simulation of the system.

The next table shows the configuration flags and their meaning

Flag	Description
name	Name of the model
handle	name of the function to simulate the system
folder	where the data will be stored. Make sure that the folder exists
initialState	Initial state of the system
param	Parameters of the system
time	How long the system will be simulated
dt	timestep
dty	measurement timestep
NSR	Noise to signal ratio

In this case, we will have a data point each 0.1 second, the MSD is simulated for 10 seconds, $m = 1kg$, $c = 3.0Ns/m$, $k = 2000N/m$, $\sigma_g = 2.0$, $A = 400N$ and $f = 6Hz$. Furthermore, $u(0) = 0, \dot{u}(0) = 0$, the time step $\Delta t = 1e-4$ and the NSR is 5%.

To generate the data, type in the terminal

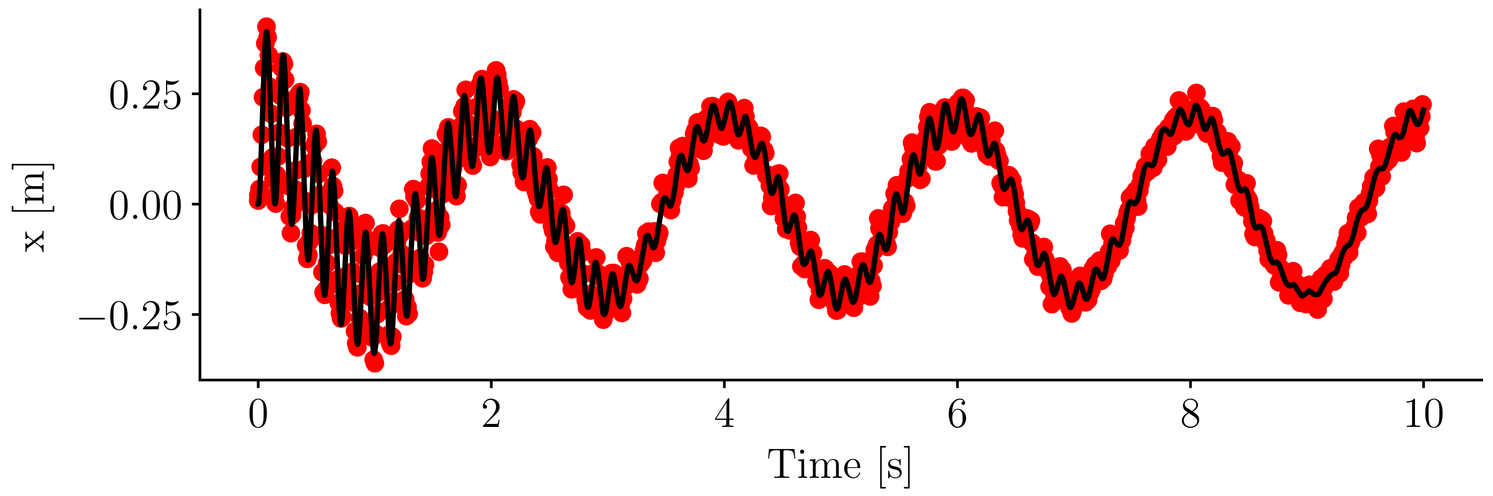
Note: If you have to make any change to this file, there is no need to recompile the whole package. Simply recompile that file.

```
mkdir Example1 (note: In case the folder did not exist)
./data.out genmodels.cfg
```

To plot the generated measurements

```
cd Example1
python ../../visualization/visualisation.py -d -n x
```

Running the previous command, created 3 files in folder *Example1*. The file *true.dat* contains the true signal from which the measurements are generated, *data.dat* contains 2 rows. The first row is the measurement and the second row contains the time at which they were taken. Finally, *variance.dat* is the variance of the measurements. One possible realisation of measurements is given in Fig **??**.



This concludes the first part of generating the data. The next part deals with parameter estimation.

3.2.2 Model selection and parameter estimation

In *functions.cpp*, we will write the proposed model. The proposed model can be different from the generating model but in our case, it will the same. We first propose a deterministic mass spring damper system that will be called *example1Model*. This model only has one unknown parameter, we must define the other parameters in the function itself. Each unknown parameter is part of the vector *parameter*. In this case the stiffness $k = parameter[0]$.

We also need to describe the measurement operator. In this case the noise operator doesn't include the additive Gaussian noise.

The final part is to describe the proposed state space for that model *example1StateSpace*.

The next step is to compile the code. To do so we first compile *functions.cpp*. In the terminal, type

```
make func
```

The next step is the configuration file for the proposed models *propmodels.cfg*. For this example, we only propose 1 model. For model selection, we would need to propose multiple models.

The next table shows the configuration flags and their meaning

Flag	Description	Default value
run	Flag to indicate to use this model	
state_estimator	What state estimator to use	
name	Name of the model	
handle	name of the state space to simulate the system	
folder	where the data will be stored. Make sure that the folder exists	
data	Name of the data file	
initialState	Initial state of the system	
param	Initial parameters (starting point the MCMC chain)	
method	MCMC method	
dt	timestep	
measurementCov	measurement covariance filename	
prior	parameter prior	

Before running the chains, you can do a simple optimization like Nelder-Mead to find a good starting point for the MCMC chain. Better optimization methods will be provided in future versions. To do so

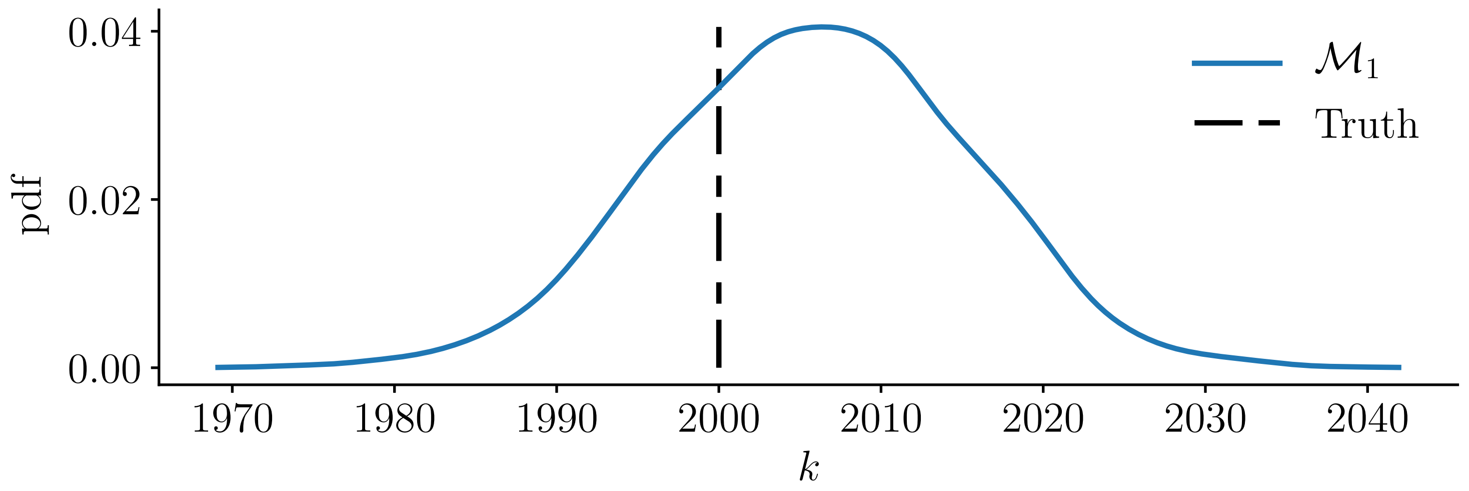
```
make opt
mpirun -np <NP> ./opt.out example1.cfg
```

To run the MCMC chains

```
make main
mpirun -np 2 ./run.out example1.cfg
```

3.3 Results

Before looking the results it is always a good idea to take a look at the MCMC chains. A visual inspection will confirm if the chain properly mixed. If it did not, tweak the MCMC parameters. You can also use the MAP as the starting point for better mixing. In some cases the chain might never mix for a ill-posed problem.



To visualize the results

```
cd Example1
python ../../visualization/visualisation.py -i MSD-DET.dat -n k -b 0.3
python ../../visualization/visualisation.py -i MSD-EKF.dat -n k -n \sigma -b 0.3 -b 0.3
```

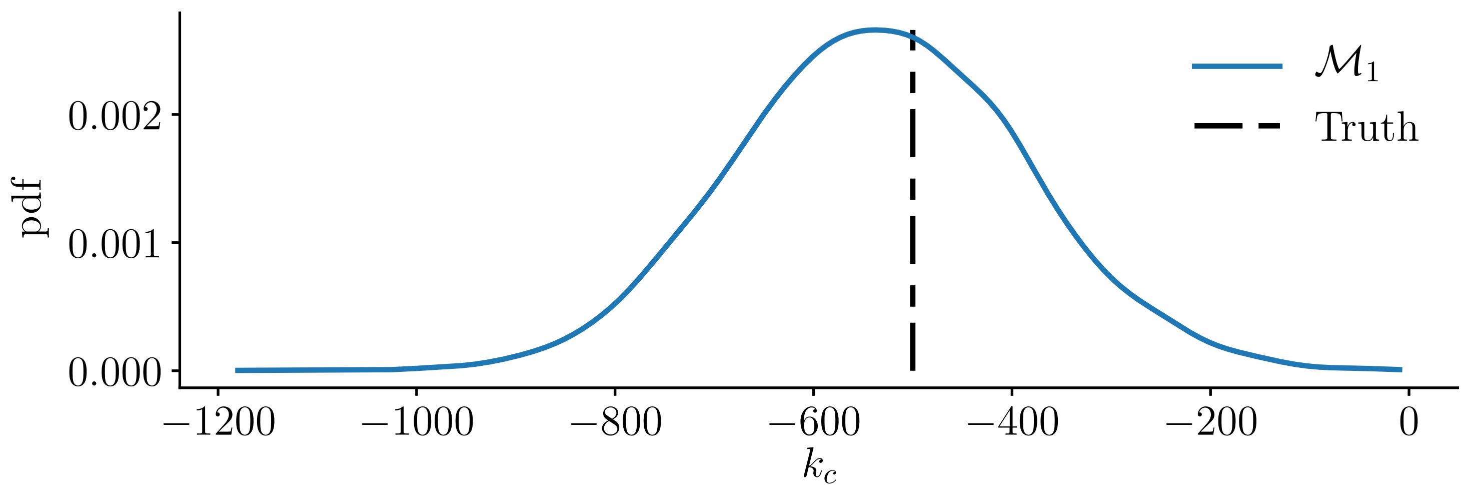
This will create multiple figures presented below

The log of the evidence is recorded in the files MSD-EKF-logevidence.dat and MSD-DET-logevidence.dat.

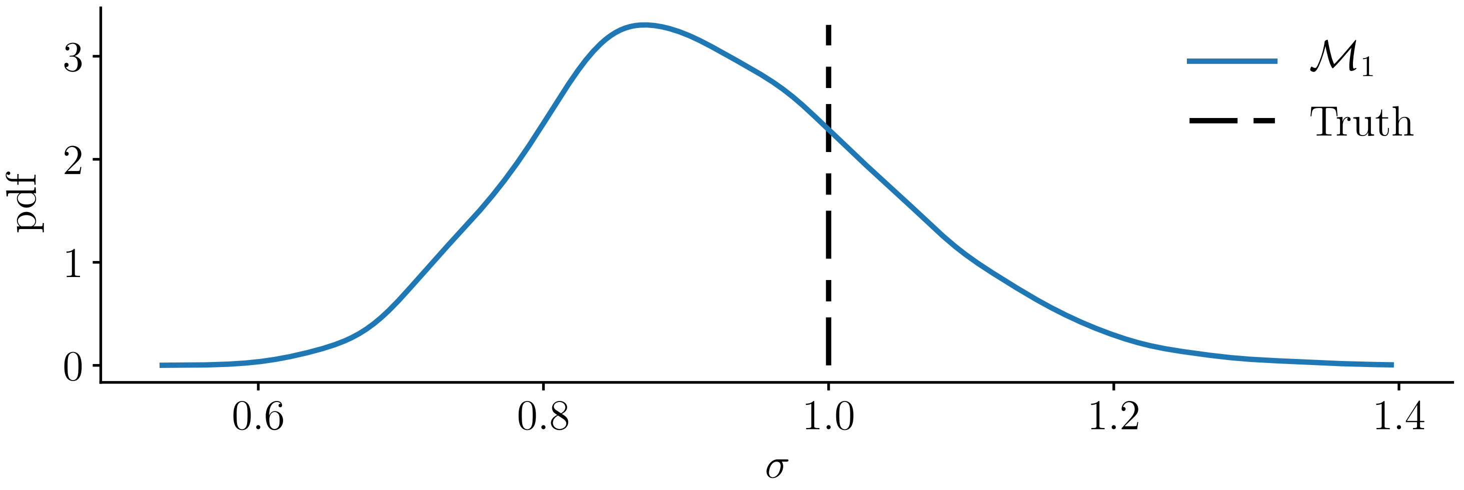
For this simple example, we can compute the posterior without using MCMC.

4 Example 3

In this example, we use the filters to perform state estimation. We compare EKF, EnKF and the bootstrap particle filter. We first generate the measurements.



$\mathcal{M}_1$
Truth
pdf
0.000
0.001
0.002
-1200
-1000
-800
-600
-400
-200
0
k_c



3
2
1
0
pdf
0.6
0.8
1.0
1.2
1.4
σ
$\mathcal{M}_1$
Truth

```
cd example-3
./data.out genmodels.cfg
```

The measurements are shown

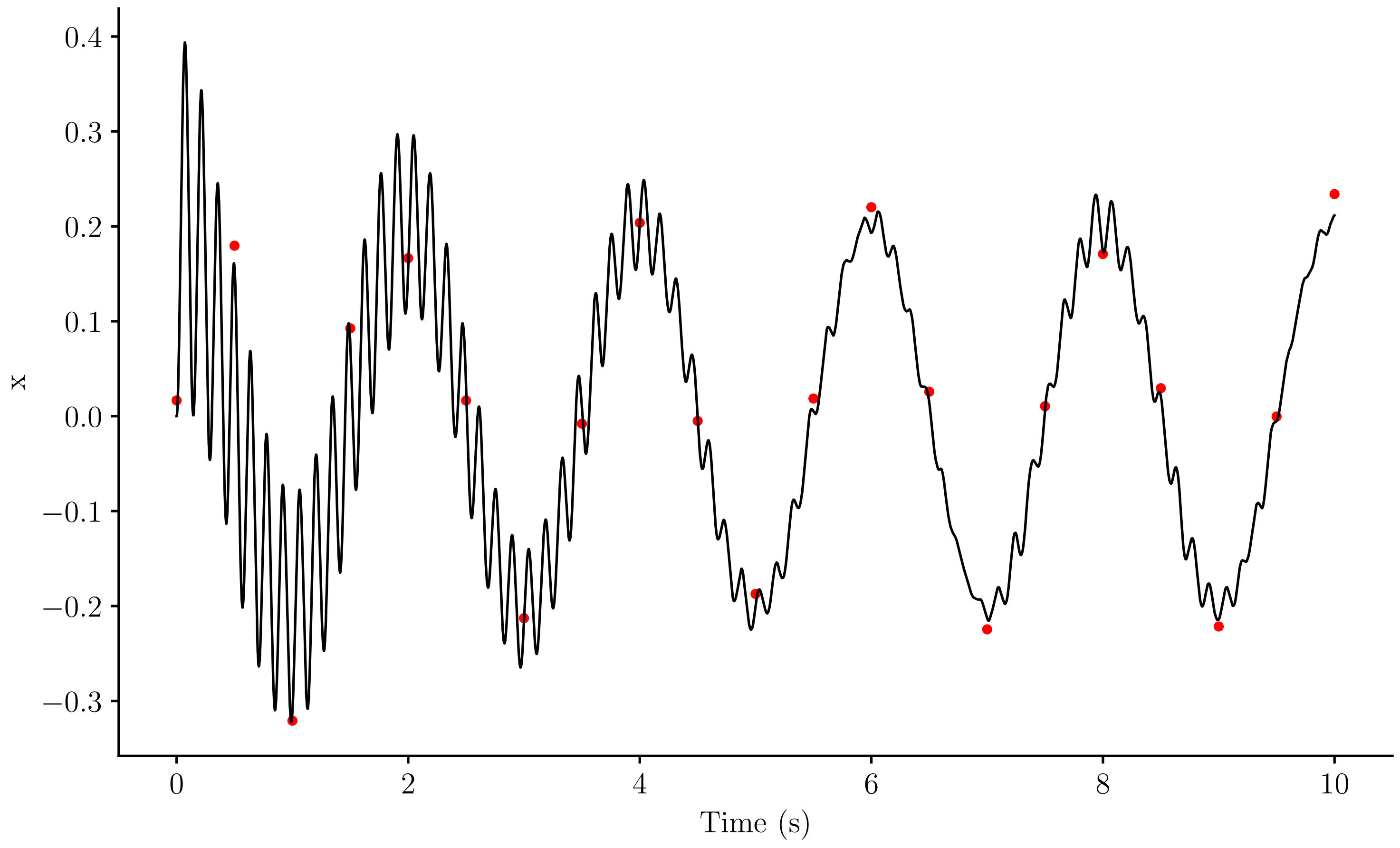


Figure 1: Synthetic measurements used for Example 3

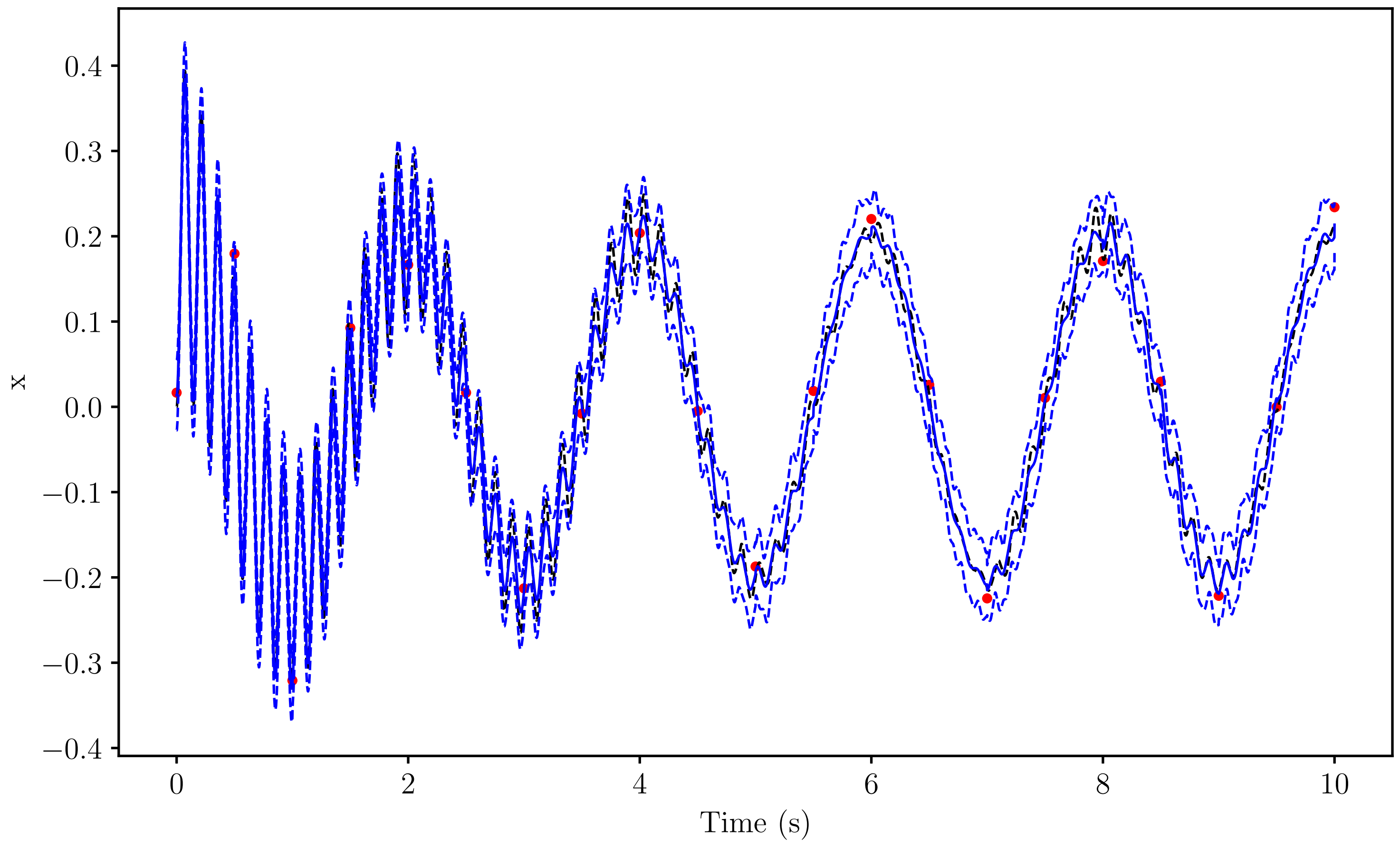


Figure 2: State Estimation using EKF

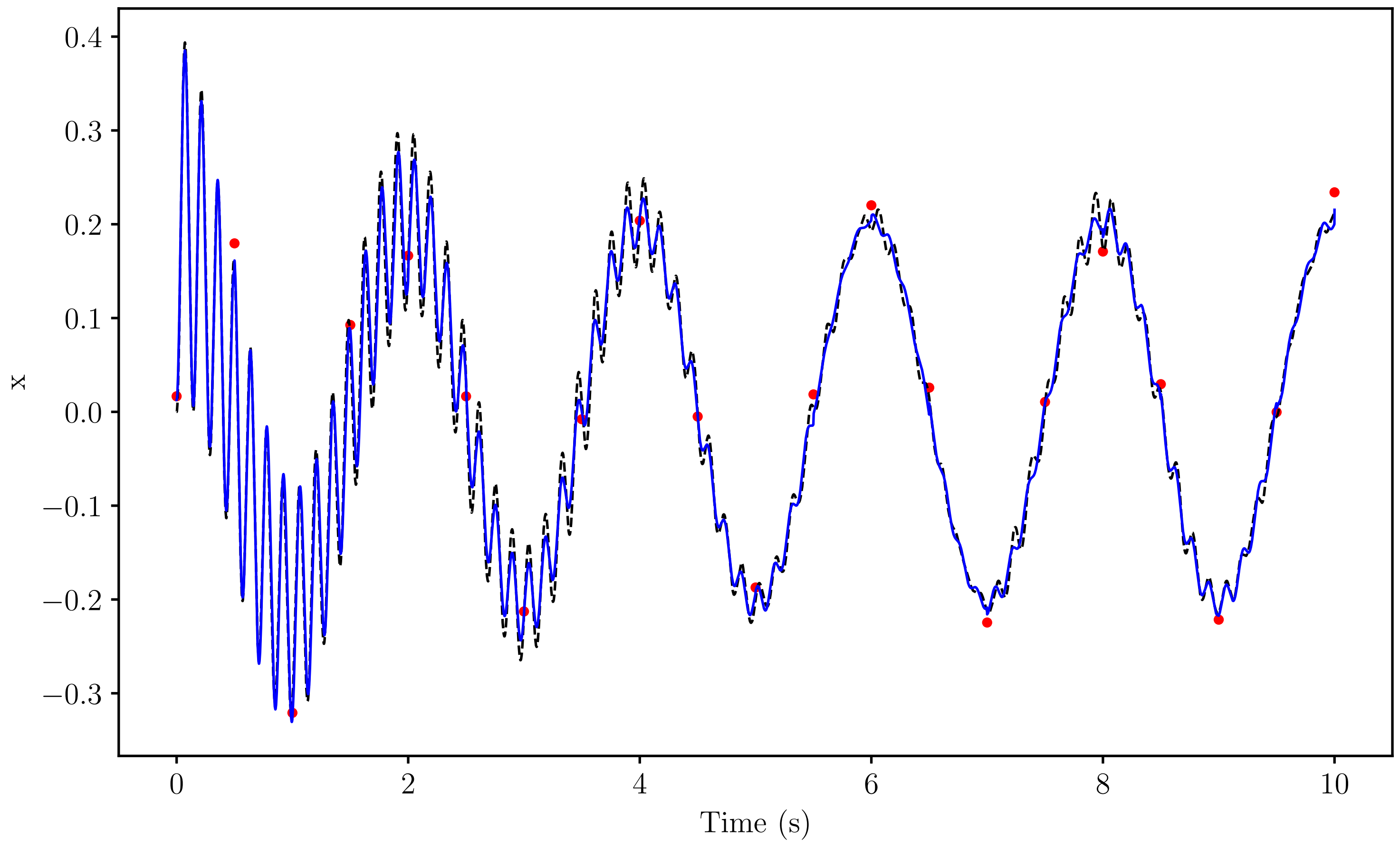


Figure 3: State Estimation using EnKF

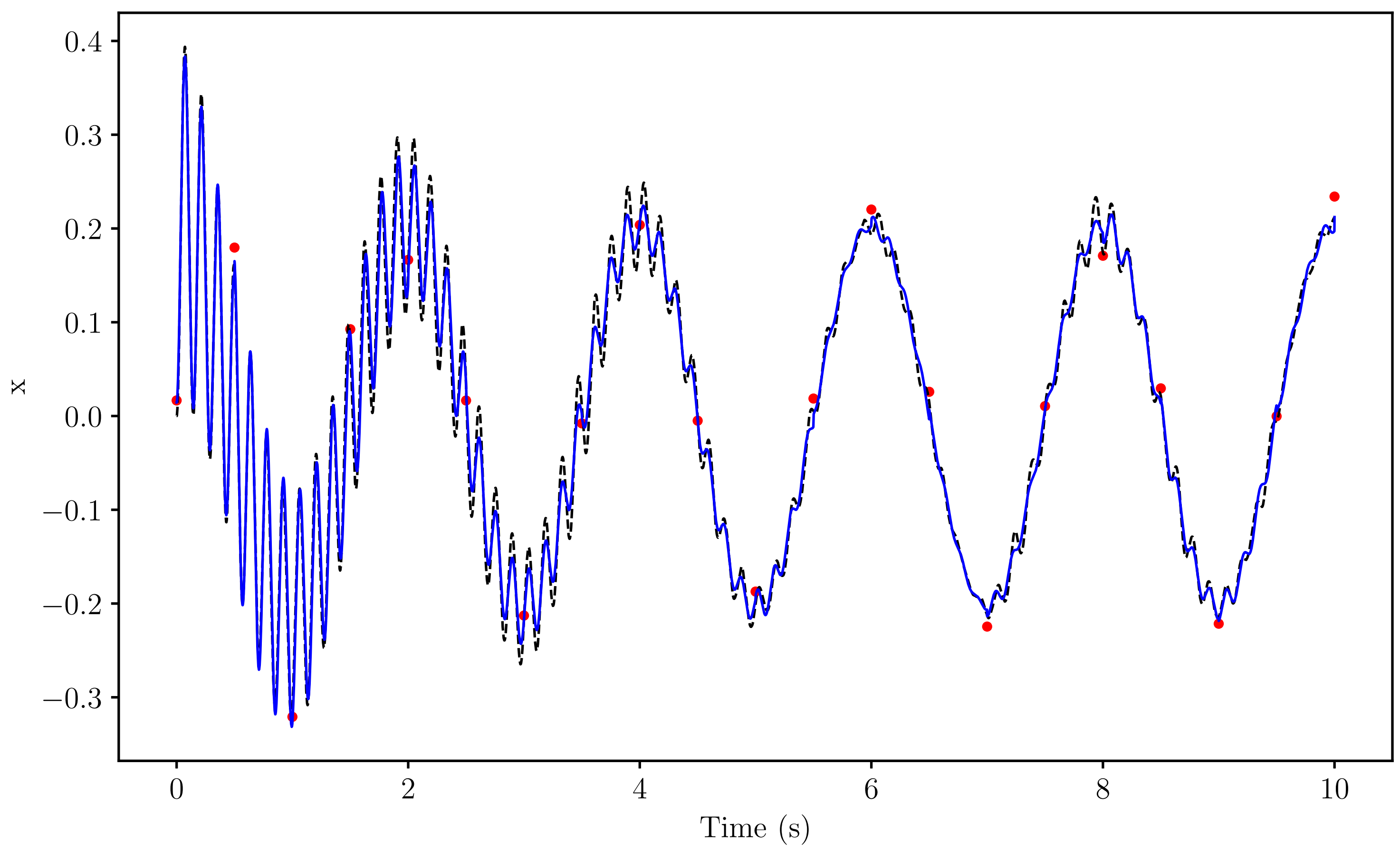


Figure 4: State Estimation using PF-SERIAL

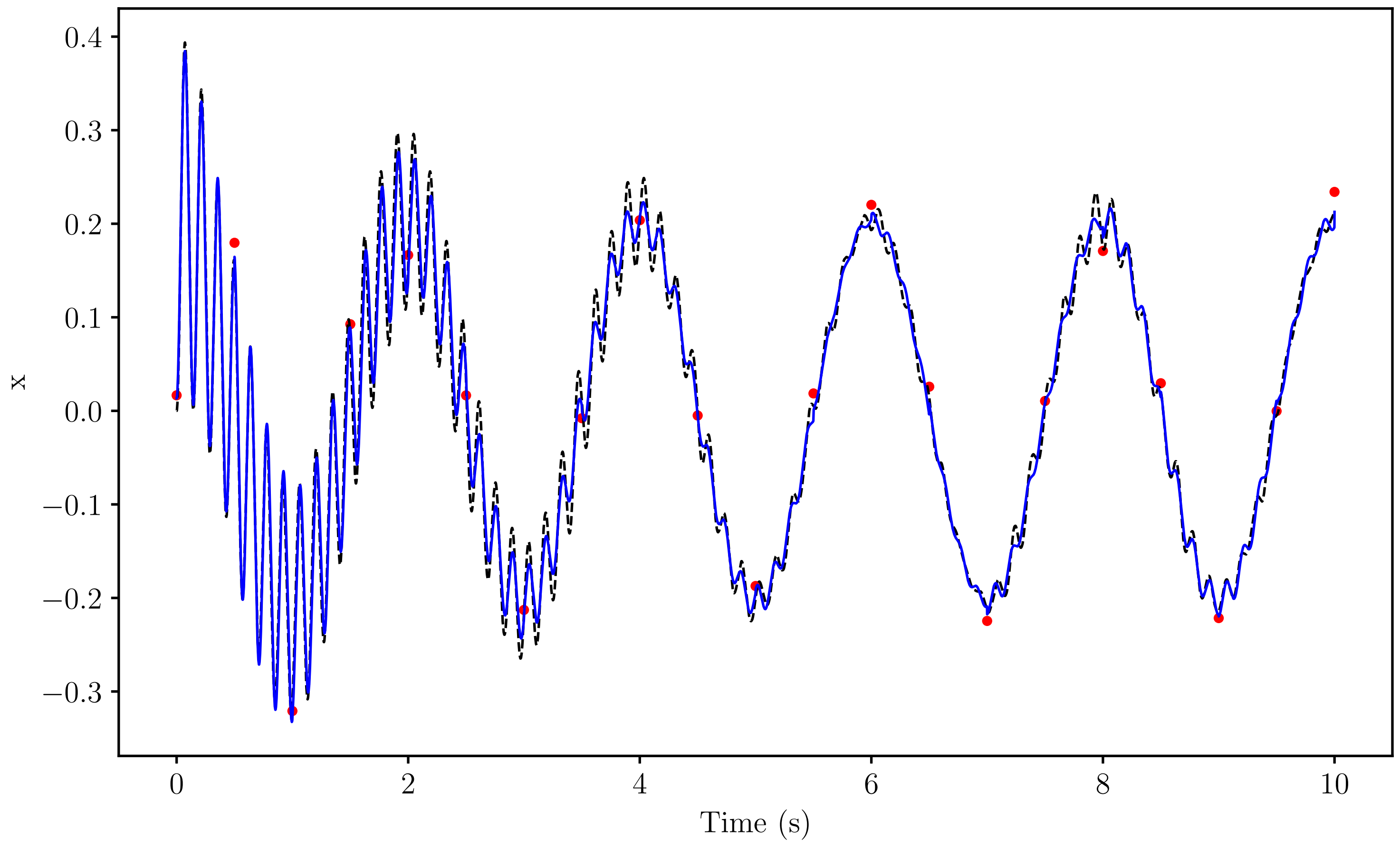


Figure 5: State Estimation using PF-MPI

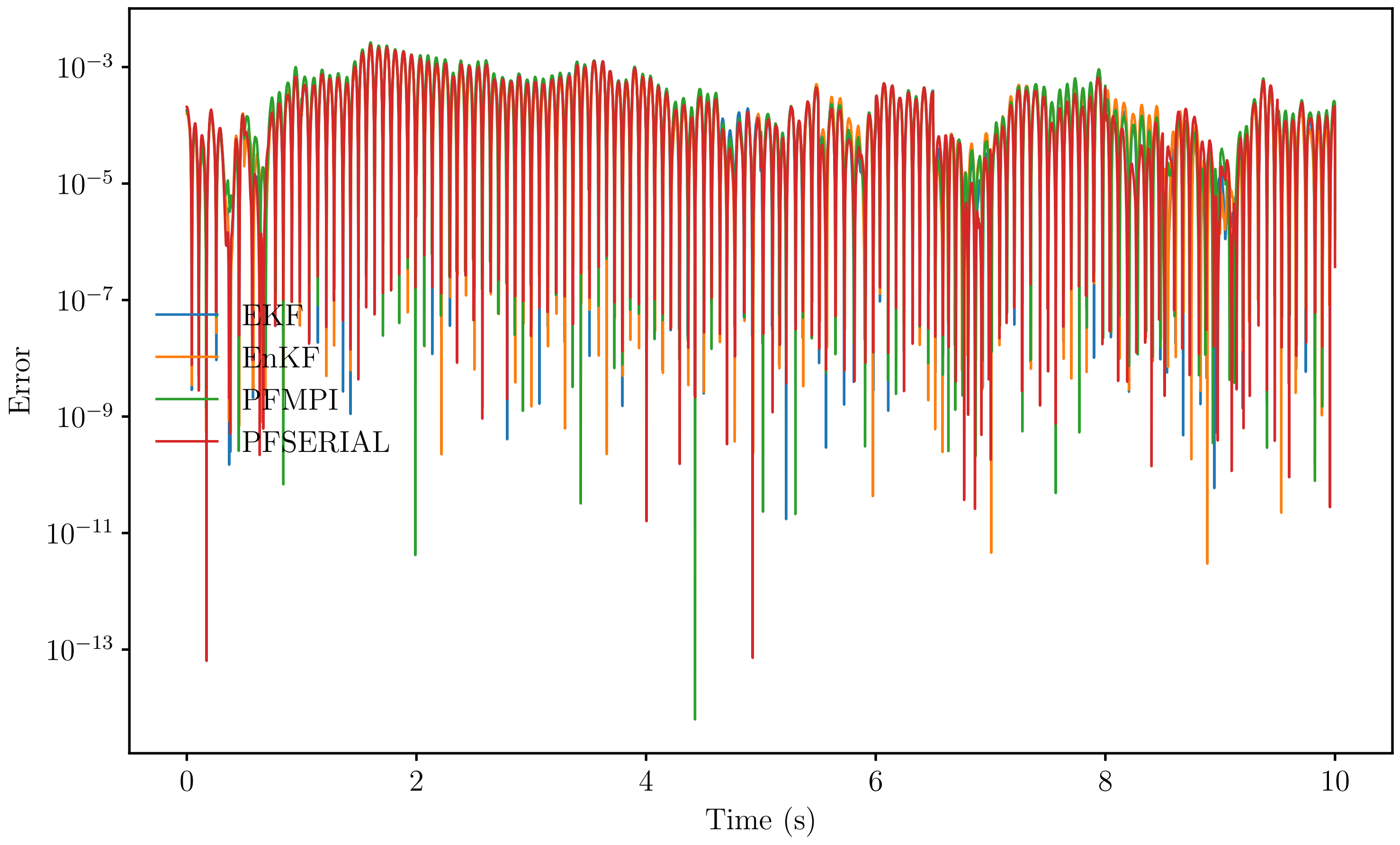


Figure 6: Error