Bayesian Strategies for Inverse Problems

Creation of a solver to determine differential equations of trajectories and comparison of the influence of schemes and models

Nathan Eckert

29th August 2016

Introduction

I decided to work on the computation of the heat flux in the walls of a combustion chamber with only the knowledge of the temperature at some points on the surface of the walls. Using data from sensors along the z-axis of the combustion chamber, the bayesian approach would be applied for the boundary conditions to determine if the profile along the other axis is constant and parabolic. The issue I had with implementing such a bayesian approach is that the direct problem needed approximately 3 minutes of computation to be solved, which is suitable for a deterministic approach for an inverse problem but not for a bayesian with a lot of samples. The computation of the direct problem was made by the solver Ansys Mechanical and I was wondering if I could do without and use instead a direct discretization with, for example, finite differences. The main determination was the choice of the scheme : how can I check if a scheme is better than an other one? To compare I needed to work with data that I could simulate. So I decided to use differential equations with known results, beginning with first-order equations and following up with more complex second-order equations. I stopped before adding equations that also need a spatial discretization and are a field equation (not just trajectories of a point).

My goal is to be able to determine the model (first order or second order for example), as well as which scheme is the best one to discretize this problem and at last, the coefficients of the equation. If you are interested, you can find the solver on Github on this repository.

1 Simulation of the data

We need as many ways of simulating the data as models we are choosing. That's why I chose solvable problems in order to generate exact solutions and add noise afterwards. I

was considering differentials equations of first or second order:

$y' + ay + b = 0$	First order
$y'' + ay' + by + c = 0$	Second order

To simulate the data, we have the coefficients, the inital position of y and the level of noise. Also we have the timestep between the two measurements. For the considered problem, we also add noise to this time, assuming the measurement of the time was not accurate. So the data resulting of our experiment is an array of points and the array of the time of the measurements of the position of each of these points, with an inaccurracy due to experimental noise on both. The experimental noise was assumed to be gaussian.

2 Models

The solver is able to compare two models. The models that can be compared are the following:

$y' + ay = 0$	First order homogenous
$y' + ay + b = 0$	First order
$y'' + ay' + by = 0$	Second order homogenous
$y'' + ay' + by + c = 0$	Second order

Why are we considering the homogenous models and the non-homogenous one, when we can say that the homogenous one is just the non-homogenous one with a constant equal to zero? I wanted to study the influence of the noise and be able to compare two models that should give the same outpout (with the constant being zero in the non-homogenous model). There were two possibilities to compare two different models. One possibility is to run the two models and at the end compare them by checking which model is more likely. The second possibility is to treat the model as a variable that should be randomly chosen and incorporate it into the Metropolis-Hasting algorithm. Both methods are feasable, though the second one is a bit more complicated to implement. Both would utilize the same number of sample. The advantage of the first method is that it's pretty straight forward but if one model is not likely at all, we are still computing it and hence are losing CPU time. This issue is not there in the second method. In order to compare the models, I chose the first method. Later I will discuss the second method which I chose in order to compare the discretization options.

3 Discretization

3.1 First order model

To discuss the different methods of discretization, let's just consider the general equation:

$$y' + ay' + b = 0 \tag{1}$$

At the timestep t_n,

$$y(t_n) = y_n$$

3.1.1 Discretization of the term of first order

The term that creates the most possibilities is the term of the first order

$$\begin{aligned} \text{Downwind discretization} \quad & y'(t_n) = \frac{y_{n+1} - y_n}{\Delta t} \\ \text{Upwind discretization} \quad & y'(t_n) = \frac{y_n - y_{n-1}}{\Delta t} \\ \text{Central discretization} \quad & y'(t_n) = \frac{y_{n+1} - y_{n-1}}{2\Delta t} \end{aligned}$$

3.1.2 Complete scheme

Using the discretization provided for all the terms in (1), we have now:

$$\begin{aligned} \text{Downwind discretization} \quad & \frac{y_{n+1} - y_n}{\Delta t} + ay_n + b = 0 && (2) \\ \text{Upwind discretization} \quad & \frac{y_n - y_{n-1}}{\Delta t} + ay_n + b = 0 && (3) \\ \text{Central discretization} \quad & \frac{y_{n+1} - y_{n-1}}{2\Delta t} + ay_n + b = 0 && (4) \end{aligned}$$

We can observe that we can find (4) by adding (2) and (3), so we will use that property to compute this scheme. Let's just focus on (2) and (3) :

$$\begin{aligned} \text{Downwind discretization} \quad & y_{n+1} = y_n(1 - a\Delta t) - b\Delta t && (5) \\ \text{Upwind discretization} \quad & y_{n+1}(1 + a\Delta t) = y_n - b\Delta t && (6) \end{aligned}$$

3.2 Second order model

We proceed for the second order model with the same method

$$y'' + ay' + by + c = 0 \tag{7}$$

At the timestep t_n,

$$y(t_n) = y_n$$

Also, the discretization of the term of the second order can be achieved easily through central differences thanks to the Taylor expansion:

$$y''(t_n) = \frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2}$$

3.2.1 Discretization of the term of first order

The term that creates the most possibilities is the term of the first order:

$$\begin{aligned}
\text{Downwind discretization} \qquad & y'(t_n) = \frac{y_{n+1} - y_n}{\Delta t} \\
\text{Upwind discretization} \qquad & y'(t_n) = \frac{y_n - y_{n-1}}{\Delta t} \\
\text{Central discretization} \qquad & y'(t_n) = \frac{y_{n+1} - y_{n-1}}{2\Delta t}
\end{aligned}$$

3.2.2 Complete scheme

Using the discretization provided for all the terms in (1), we have now:

$$\text{Downwind discretization} \qquad \frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2} + a\frac{y_{n+1} - y_n}{\Delta t} + by_n + c = 0 \tag{8}$$

$$\text{Upwind discretization} \qquad \frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2} + a\frac{y_n - y_{n-1}}{\Delta t} + by_n + c = 0 \tag{9}$$

$$\text{Central discretization} \qquad \frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2} + a\frac{y_{n+1} - y_{n-1}}{2\Delta t} + by_n + c = 0 \tag{10}$$

Same as before, we can omit the central discretization being just the sum of the two others.

$$\text{Downwind discretization} \qquad y_{n+1}(1 + a\Delta t) = y_n(2 + a\Delta t - b\Delta t^2) - y_{n-1} - c\Delta t^2 \tag{11}$$

$$\text{Upwind discretization} \qquad y_{n+1} = y_n(2 - a\Delta t - b\Delta t^2) + y_{n-1}(a\Delta t - 1) - c\Delta t^2 \tag{12}$$

These are the schemes used in the solver. It's important to notice that we have to compute the second term from the first one in order to have a proper initialization of our schemes. (These is done by using a standard upwind scheme on one timestep).

3.3 Remark on the discretization timestep

If you look carefully at the solver, you will notice that the timestep used for the computation of the scheme is the same as the one used in the experimental measurement. This assumption was made in order to simplify the computation and also because this assumption is very plausible: if you want to determine a trajectory and its differential equation, you should have as much data as possible and it doesn't make sense to increase too much the cost of computation of the model by using a very small timestep. However; in case of problem of convergence or if the measurements are done with a very large interval between them, we do need a more accurate time discretization for the schemes. Let's assume if we need a discretization twice better, we can simply use the scheme twice. The issue is that if we want to be able to change it from two but three times, we need to change the code another time. A better way is to compute the general formula for the schemes. This can be done by noticing that the first order scheme is an arithmetico-geometric sequence and the second order one is a linear second order recurrent sequence so explicit solutions can be easily found. This solution has also the advantage that if different time between the measurements are used, we can still compute the position at the measured times.

4 Comparison of the schemes

We have 3 schemes that we can use and we want to compare them. We spoke earlier of two methods for comparing the models. For the models, we were computing the results for two different models and comparing them afterwards. For the schemes we used a different method. The scheme was added as a parameter, randomly chosen at the beginning, which would be able to change at each iteration. Basically, at each iteration, for a given sample we pick a new scheme and pick a sample of this scheme among the current distribution and then evaluate the best sample between the new one and the old. We use the Metropolis-Hasting algorithm to exchange them. Afterwards, we proceed to the iteration of the parameters of the scheme. This method has an issue : at the beginning, the samples are very dependent of the prior guess of their distribution on the coefficients. So the first iteration exchange schemes between samples that not representatives of the real distribution : so after a certain time we arrive with the real distribution of each scheme and then compare them. But if there is one scheme with very few samples (due to exchange between parameters that didn't have yet the good distribution), the algorithm will not be able to pick samples from this scheme that are representative of the distribution and even if it is a good scheme, like its samples are "bad", it will disappear. The solution I have used is before performing the algorithm, run it on each scheme to have a more accurate distribution and then exchange them.

5 Algorithm and problem formulation

5.1 Bayesian formulation of the problem

We are investigating the measurements of the position $\hat{y}$ according to our parameters θ and with a gaussian noise ε of variance σ_{noise}. Our problem is the following one:

$$\hat{y} = g(\theta) + \varepsilon$$

Where $\hat{y}$ is the vector of all the n measurements. There is no limitation on the size of this matrix. It just needs to have enough data (3 measurements are not enough). We have several parameters θ : we can have the model, the scheme, and the parameters of differential equation. We could also easily add the initial conditions. Let's go backwards in order to study our parameters.

5.1.1 The coefficients of the differential equation

The idea of the solver is that if we know which model to compute and which scheme to use, we can find the coefficients of our differential equation. But which distribution should we use as a prior for them? My idea is that we could have an estimation of its value by looking at the experimental data and the problem and just chose a uniform distribution. For example, a physicist might imagine a value near 1 and decide to have the uniform prior on $[0; 2]$. The length of the interval is quite arbitrary but the smallest it is around the value, the better it will be. We cannot be sure of the value at the beginning; it must be chosen from experience. According to the model we are computing, there will be one, two or three coefficients parameters. In order to iterate our Metropolis-algorithm, we are using a gaussian distribution and modifying each parameter of the same sample with different values.

$$parameters = parameters_{prior} + N(0; \sigma^2_{algorithm} I)$$

In order to choose the next parameters, we perform a random walk which is useful to simplify the Metropolis-Hasting algorithm as it is a symmetrical probability distribution. We chose $\sigma^2_{algorithm}$ in order to have a good acceptance ratio in the algorithm (between 30 and 40 %).

Using Bayes' rule, the posterior for the parameters distribution will be:

$$p(\theta \mid \hat{y}, model, scheme) = \frac{p(\hat{y} \mid \theta, model, scheme) \cdot p(\theta)}{p(\hat{y} \mid model, scheme)}$$

Given our problem, the likelihood of our data according to the parameters is:

$$p(\hat{y} \mid \theta, model, scheme) \propto N(g(\theta), \sigma^2_{noise} I) = \frac{1}{(2\pi\sigma^2_{noise})^{\frac{n}{2}}} \times exp(\frac{-1}{2\sigma^2_{noise}} \|\hat{y} - g(\theta)\|^2)$$

And by using all of this expression in the previous Bayes' rule, we obtain:

$$p(\theta \mid \hat{y}, model, scheme) \propto \frac{1}{(2\pi\sigma^2_{noise})^{\frac{n}{2}}} \times exp(\frac{-1}{2\sigma^2_{noise}}\|\hat{y} - g(\theta)\|^2) \times prior_{uniform}$$

This is the posterior distribution of our sample and will be the prior for the next iteration.

5.1.2 The comparison of the schemes

Although the comparison of the scheme has already been discussed, it is important to point out that the probability of going to a new scheme does not depend on the number of samples in the scheme. The algorithm picks a new scheme and afterwards compares the new and old scheme before exchanging them. The non-dependance of the scheme of the number of sample of each scheme assure that it is as likely to try an upwind, as downwind or as a central scheme for the next iteration. The fact that one given scheme might be more successful than an another one is just due to the importance of the errors (measured by the posterior) of the new scheme compare to the prior which is the old scheme.

5.2 Algorithm

5.2.1 Metropolis-Hasting algorithm

As already said, we use a standard Metropolis-Hasting algorithm:
First, pick a new set of parameters θ_{post} from the old one θ_n (either with a random walk for the coefficients, or with a uniform probability to pick a new scheme)
Second, compute the value of $a(\theta_{post}, \theta_n) = min(1, \frac{\Pi(\theta_{post})}{\Pi(\theta_n)})$
Finally, set $\theta_{n+1} = \theta_{post}$ with the probability a and otherwise $\theta_{n+1} = \theta_n$

Here is an idea of the acceptance ratio according to the value of $\sigma_{algorithm}$:

Model	$\sigma^2_{algorithm}$	Acceptance ratio
First order	0.1	33 %
First order	0.15	22 %
Second order	0.1	48 %
Second order	0.15	28 %

According to the model we want, we can pick a good $\sigma_{algorithm}$, we just need to be careful when we compare two models to have a value that is in the range of $20 - 40\%$ for both models.

5.2.2 Remarks on the convergence and the ergodic mean

You can notice in the plot provided that that the ergodic mean never reaches convergence. This property is due to an instability of the equation. For example for the first order equation, we can see that in the histogram there is a very good convergence but some samples are going to minus infinity. The instability of this samples causes the ergodic mean to decrease and it seems that convergence is never reached. Finding a criteria of convergence is very hard, in this case. Two ideas to define it. First we leave the appreciation of the convergence to the user: we just need a big enough number of samples and iteration: you can find in the appendix the same model for:

- 2000 samples and 10000 iterations
- 1000 samples and 1000 iterations
- 100 samples and 100 iterations

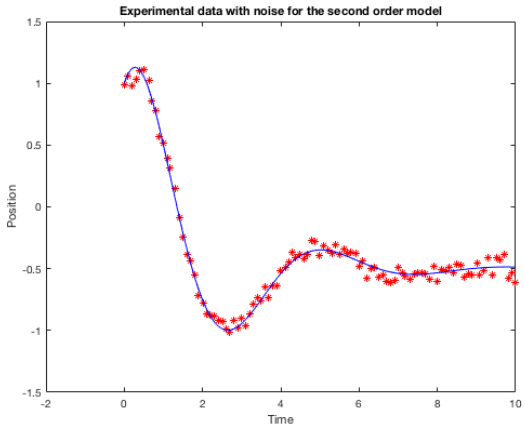
The other solution is to use a resampling method; if the value of a sample is out of given boundaries (for example the lower and upper boundaries of the prior), we can delete this sample and replace it with another one picked among the current distribution of the samples. This method is not implemented in the solver.

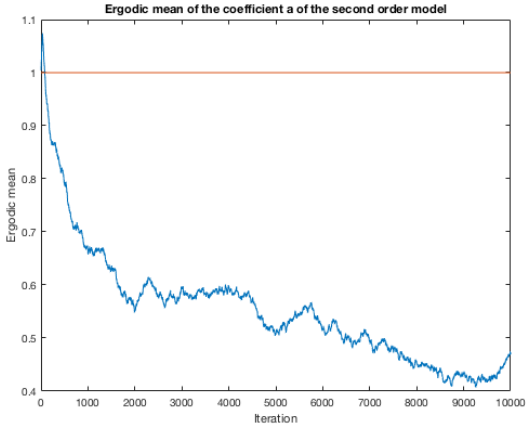
Conclusion

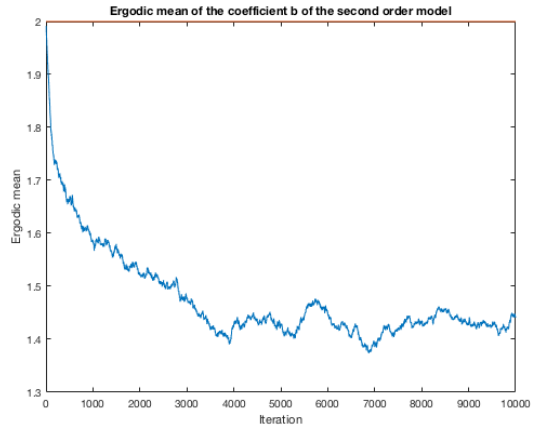
We succeed to implement a solver able to determine the coefficients of the differential equation of a trajectory. However, the convergence must be checked very carefully. This solver is able to compare the likelihood of two models and also the quality of the discretization schemes. The solver is currently quite limited but we can easily add new models, Navier-Stokes equations or advection equation for example, or new schemes. For those more complicated models, it would be great to see the influence of the diverse schemes, because it's more sensitive to the choice of discretization. We used an approach leading us to the knowledge of the coefficients of the equation, but for some problems, we know perfectly this coefficients (for example for water in Navier-Stokes), so it could be interesting to set a fix value for them and just have the scheme as parameters for example : that would help us to determine which scheme is the best one for a given problem.

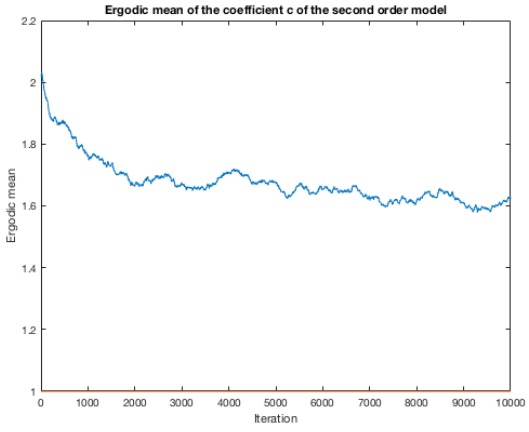
Appendix

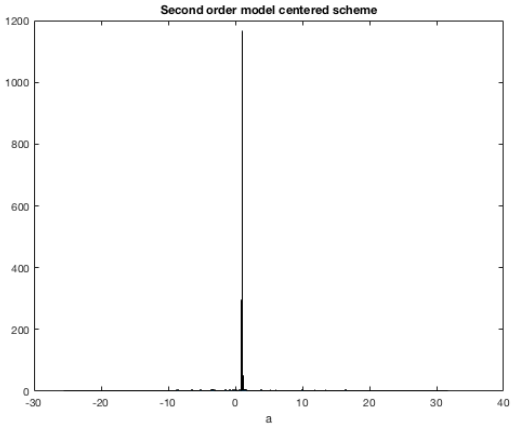
Central scheme for the second order with 2000 samples and 10000 iterations with noise. Expected values for the coefficients: a=1, b=2, c=1.

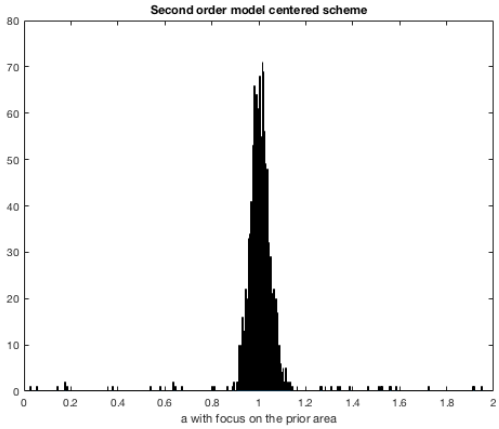


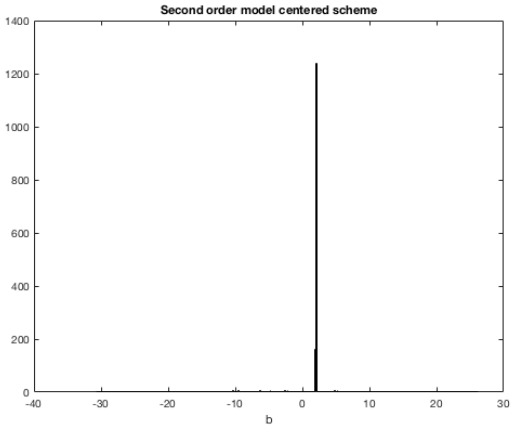


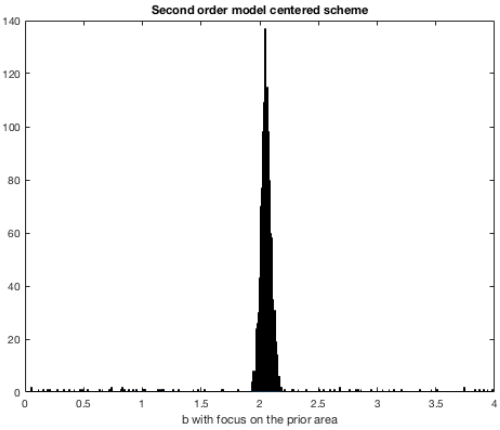


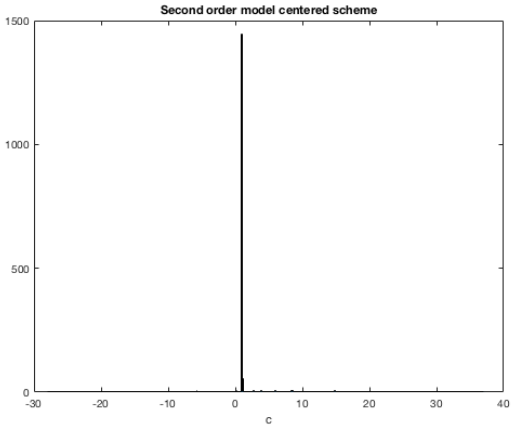


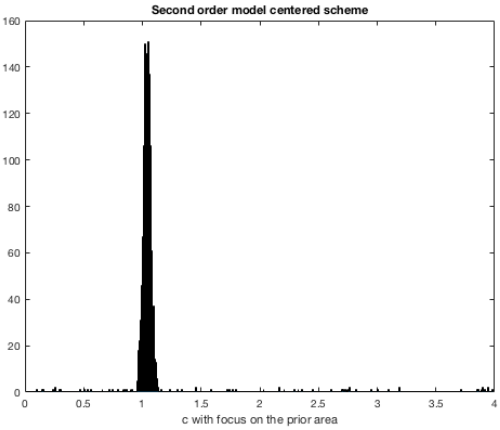

Second order model centered scheme
a


Second order model centered scheme
a with focus on the prior area

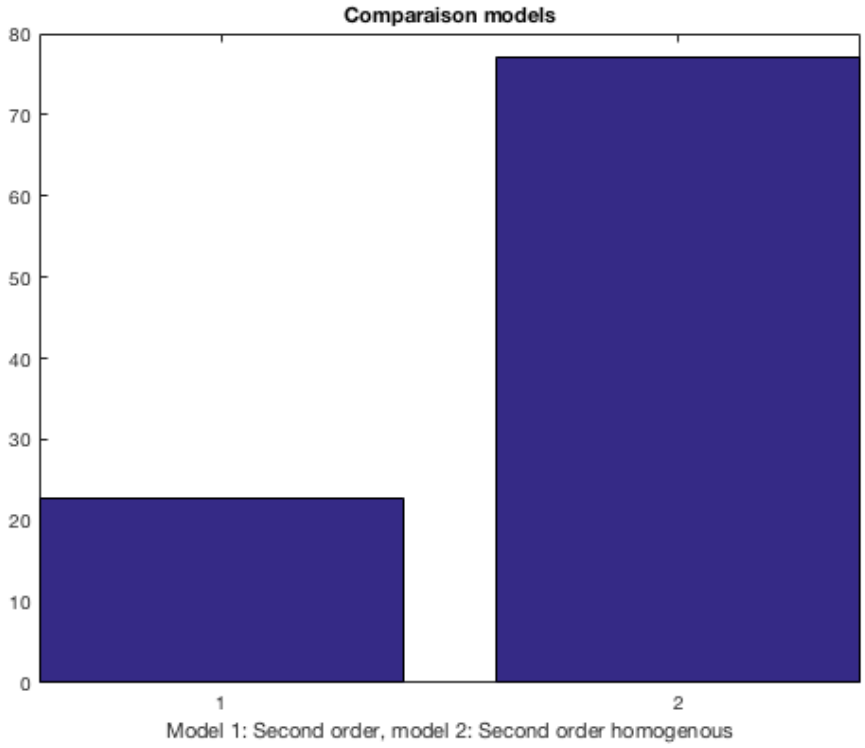

Second order model centered scheme
b


Second order model centered scheme
b with focus on the prior area


Second order model centered scheme
c

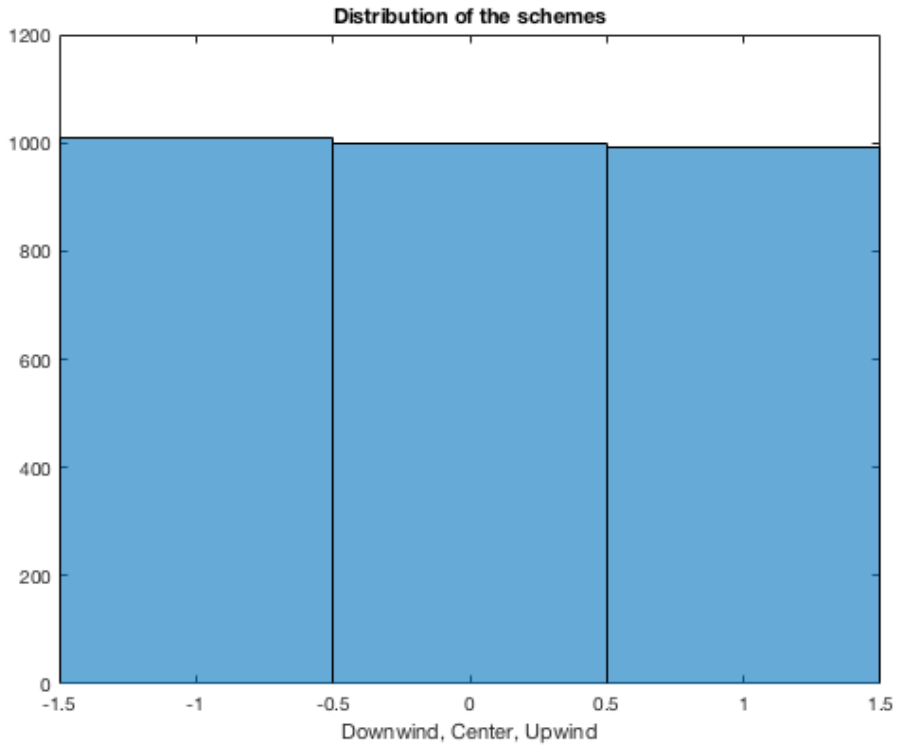

Second order model centered scheme
c with focus on the prior area

Comparison of the second order and the second order homogenous model for second order homogenous data with noise for 1000 samples and 1000 iterations



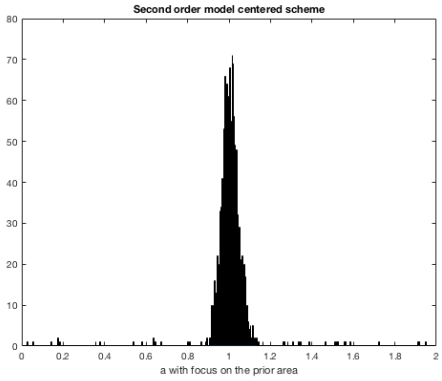
The solver is able to make the difference between two models very close (the second one is the same as the first one with a coefficient equal to zero). Also it's encouraging that the first model does not disappear completely: the solver is also able to fit the data if we assume a more complicated model.

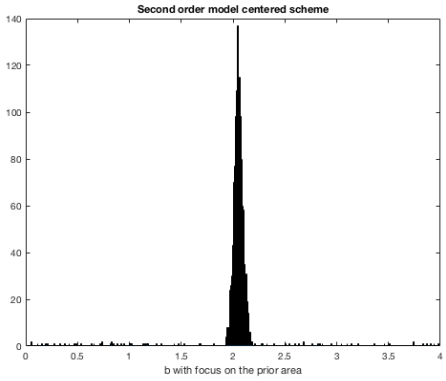
Comparison of the schemes on a second order problem. 1000 samples for each scheme at the beginning and 2000 iterations.



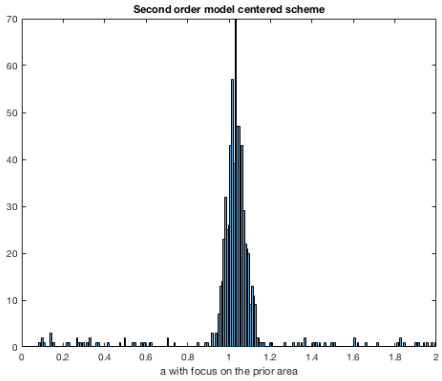
The uniform distribution of the schemes tells us that the discretization timestep is so small, that there is no influence of the choice of schemes.

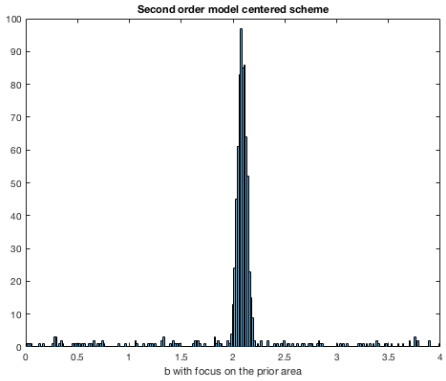
Comparison of the convergence with a different number of samples and iteration on the second order model.



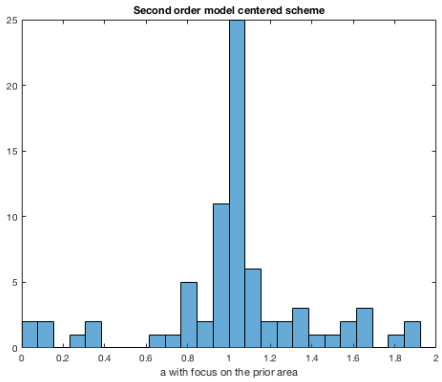


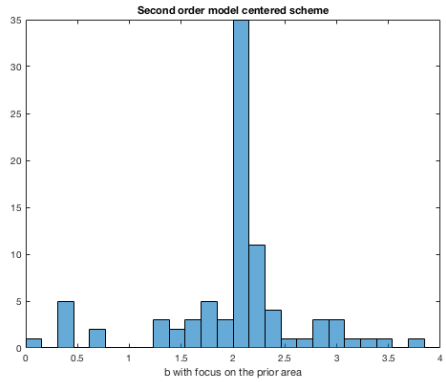
2000 samples and 10000 iterations



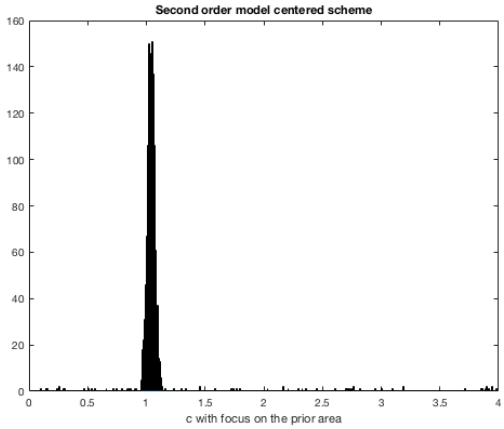


1000 samples and 1000 iterations

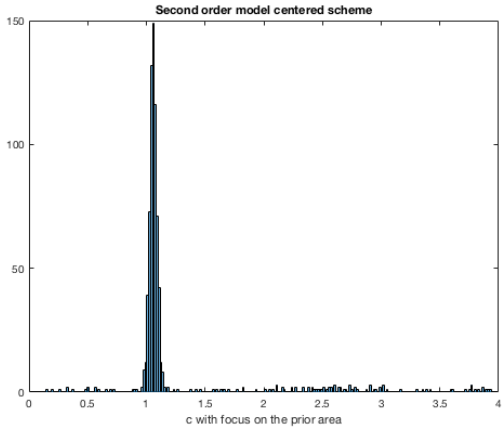




100 samples and 100 iterations

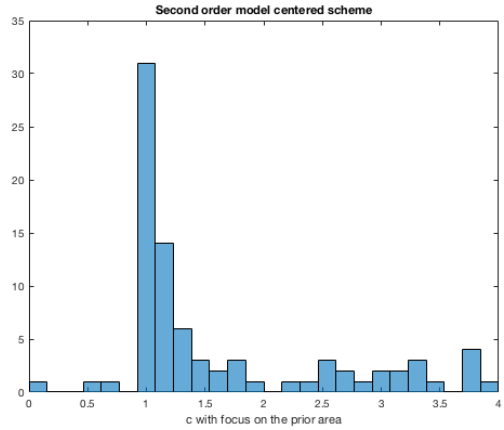


2000 samples and 10000 iterations



1000 samples and 1000 iterations

We observe that it is not very useful to have a lot of iterations. 1000 iterations are enough to have a good convergence with a reasonable number of samples.



100 samples and 100 iterations