handout

July 25, 2017

1 Quantifying Uncertainties in Physical Models

> Ignorance is preferable to error and he is less remote from the truth who believes nothing than he who believes what is wrong. Thomas Jefferson (1781)

1.1 How do I run the code in this handout?

1. Download the zipped version of this file.
2. Uncompress it anywhere in your computer (just remember where).
3. If you have Python (preferably the Anaconda version) installed in your computer and you know how to open a jupyter notebook, then just do it. Otherwise, keep reading.
4. Login to nanoHUB.org.
5. On the top right corner click at `Search` and search for the `"jupyter notebook tool"`.
6. Launch the tool.
7. Click at `File->Open`.
8. Click at the little home icon (top left corner).
9. Click on the `Upload` button at (top right corner).
10. Upload `"handout.ipynb"` (this notebook) from the folder you uncompressed in setp 2.
11. Repeat steps 9 and 10 to upload two more files:
    - `"catalysis.csv"` (some data that we are going to use),
    - and `"scheme.png"` (an image representing the physical model we will play with).

1.2 Objectives

- To tell the difference between **aleatory** and **epistemic** uncertainties.
- To define **predictive modeling**.
- To use **probability theory** to represent both aleatory and epistemic uncertainties.
- To **propagate uncertainty** through a physical model using Monte Carlo.

1.3 Readings

- Oden, Moser, Ghattas, Computer Predictions with Quantified Uncertainty, Part I
- Oden, Moser, Ghattas, Computer Predictions with Quantified Uncertainty, Part II

1.4 Definitions

We are not going to make a big effort to be consistent about the use of the following terms, since their precise meaning is still under debate.

1.4.1 Uncertainty

In general, we are uncertain about a logical proposition if we do not know whether it is true or false. In particular, we can be uncertain about: + the value of a model parameter + the mathematical form of a model + the initial conditions of a ordinary differntial equations + the boundary conditions of a partial differential equation + the value of an experimental measurment we are about to perform + etc.

Uncertainty may be *aleatory* or *epistemic*. Aleatory uncertainty is associated with inherent system randomness. Epistemic uncertainty is associated with lack of knowledge. If you think too hard, the distinction between the two becomes philosophical. We are not going to push this too hard. Fortunately, our approach (the Bayesian approach) treats both uncertainty on an equal footing.

1.4.2 Predictive Modeling

Predictive modeling is the process of assigning error bars to the predictions of computational models which rigorously quantify the effect of all (ideally) associated uncertainties. This quantified uncertainty can be used to assess the risk of making decisions based on these model predictions.

1.5 Example: Catalytic Conversion of Nitrate to Nitrogen

This is Example 3.1 of (Tsilifis, 2014).

Consider the catalytic conversion of nitrate (NO_3^-) to nitrogen (N_2) and other by-products by electrochemical means. The mechanism that is followed is complex and not well understood. The experiment of (Katsounaros, 2012) confirmed the production of nitrogen (N_2), ammonia (NH_3), and nitrous oxide (N_2O) as final products of the reaction, as well as the intermediate production of nitrite (NO_2^-). The data are reproduced in Comma-separated values (CSV) and stored in `data/catalysis.csv`. The time is measured in minutes and the conentrations are measured in mmol $\cdot$ L^{-1}. Let's load the data into this notebook using the Pandas Python module:

```
In [1]: # If this fails, you haven't uploaded "catalysis.csv".
        # Repeat 11 of the instructions.
        import pandas as pd
        catalysis_data = pd.read_csv('catalysis.csv', index_col=0)
        catalysis_data
```

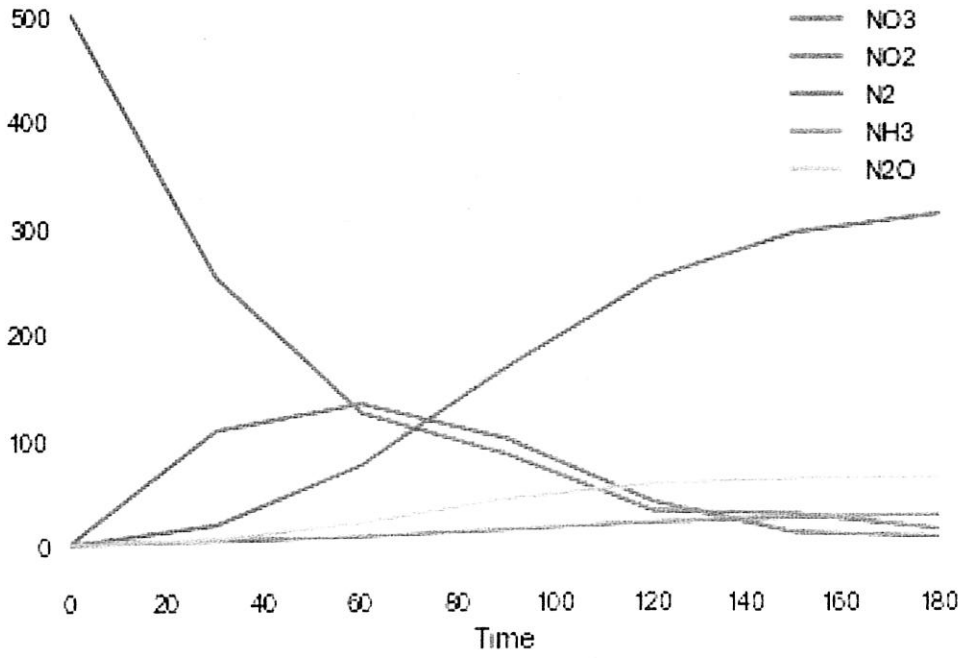
Out[1]:

Time	NO3	NO2	N2	NH3	N2O
0	500.00	0.00	0.00	0.00	0.00

```
30    250.95  107.32   18.51   3.33   4.98
60    123.66  132.33   74.85   7.34  20.14
90     84.47   98.81  166.19  13.14  42.10
120    30.24   38.74  249.78  19.54  55.98
150    27.94   10.42  292.32  24.07  60.65
180    13.54    6.11  309.50  27.26  62.54
```

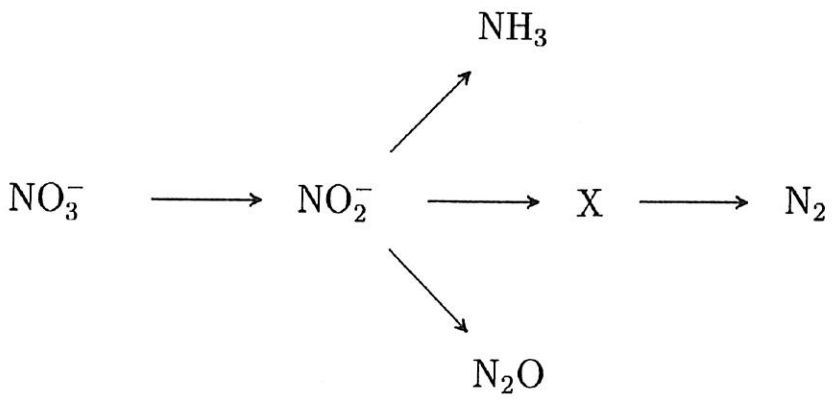
Let's visualize the data using Matplotlib:

```
In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
        %matplotlib inline
        catalysis_data.plot()

Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x11a495f10>
```



The theory of catalytic reactions guarantees that the total mass must be conserved. However, this is not the case in our dataset:

```
In [3]: catalysis_data.sum(axis=1)

Out[3]: Time
        0      500.00
        30     385.09
```




```
60     358.32
90     404.71
120    394.28
150    415.40
180    418.95
dtype: float64
```

This inconsistency suggests the existence of an intermediate unobserved reaction product X. (Katsounaros, 2012) suggested that the following reaction path shown in the following figure.

The dynamical system associated with the reaction is:

$$\begin{aligned}
\frac{d[NO_3^-]}{dt} &= -k_1\,[NO_3^-]\,,\\
\frac{d[NO_2^-]}{dt} &= k_1\,[NO_3^-] - (k_2 + k_4 + k_5)[NO_2^-],\\
\frac{d[X]}{dt} &= k_2\,[NO_2^-] - k_3[X],\\
\frac{d[N_2]}{dt} &= k_3\,[X]\,,\\
\frac{d[NH_3]}{dt} &= k_4\,[NO_2^-]\,,\\
\frac{d[N_2O]}{dt} &= k_5\,[NO_2^-]\,,
\end{aligned}$$

where $[\cdot]$ denotes the concentration of a quantity, and $k_i > 0, i = 1, ...5$ are the *kinetic rate constants*.

1.5.1 Questions

1) Assume that you are a chemical engineer and that you are assigned the task of designing a reactor for the conversion of nitrate to nitrogen. Before you start designing, you collect on information in an attempt to characterize your state of knowledge about the problem. How many different sources of uncertainty can you think of?

2) Which of these uncertainties would you characterize as aleatoric uncertainties and which as epistemic?

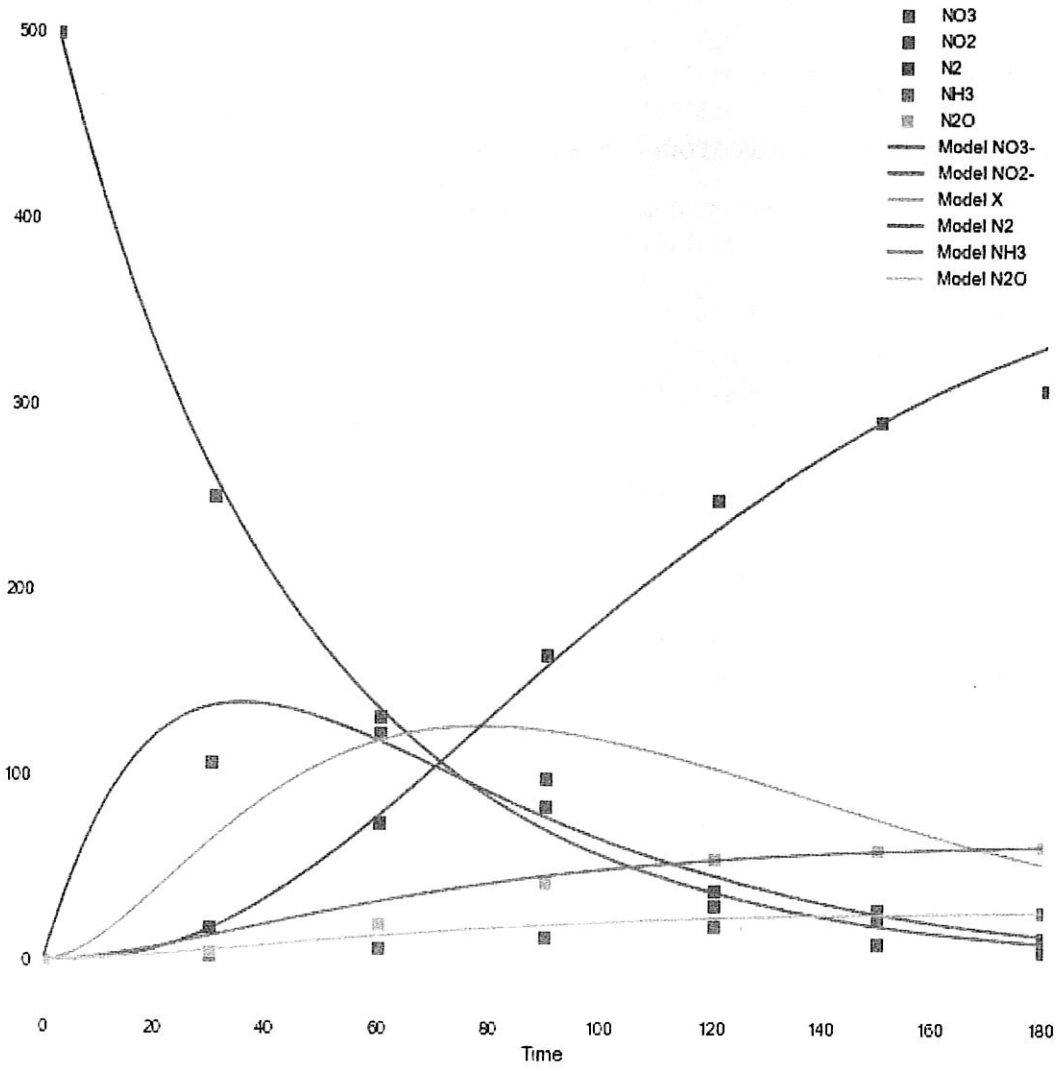
3) Is the distinction between aleatory and epistemic uncertainties always clear cut?

1.5.2 Computational Model

We will develop a generic computational model for the solution of dynamical systems and we will use it to study the catalysis problem. The code relies on the Fourth-order Runge-Kutta method and is a modified copy of `http://www.math-cs.gordon.edu/courses/ma342/python/diffeq.py` developed by Jonathan Senning. The code solves:

$$\begin{aligned} \dot{\mathbf{y}} &= f(\mathbf{y}, t), \\ \mathbf{y}(0) &= \mathbf{y}_0. \end{aligned}$$

```
In [4]: import numpy as np
        def rk45( f, y0, t, args=() ):
            """Fourth-order Runge-Kutta method with error estimate.

            USAGE:
                y = rk45(f, x0, t, args=())

            INPUT:
                f     - function of x and t equal to dx/dt.  x may be multivalued,
                        in which case it should a list or a NumPy array.  In this
                        case f must return a NumPy array with the same dimension
                        as x.
                y0    - the initial condition(s).  Specifies the value of x when
                        t = t[0].  Can be either a scalar or a list or NumPy array
                        if a system of equations is being solved.
                t     - list or NumPy array of t values to compute solution at.
                        t[0] is the the initial condition point, and the difference
                        h=t[i+1]-t[i] determines the step size h.
                args  - any other parameters of the function f.

            OUTPUT:
                y     - NumPy array containing solution values corresponding to each
                        entry in t array.  If a system is being solved, x will be
                        an array of arrays.

            NOTES:
                This version is based on the algorithm presented in "Numerical
                Mathematics and Computing" 6th Edition, by Cheney and Kincaid,
                Brooks-Cole, 2008.
            """

            # Coefficients used to compute the independent variable argument of f

            c20 =   2.500000000000000e-01  #  1/4
            c30 =   3.750000000000000e-01  #  3/8
            c40 =   9.230769230769231e-01  #  12/13
            c50 =   1.000000000000000e+00  #  1
            c60 =   5.000000000000000e-01  #  1/2
```



This is the calibration problem.

1.6.1 Questions

1) Obviously, you do not want to be calibrating models by hand. Can you think of a *natural* way to calibrate a model?

2) No matter what we do, we cannot really match the data to the model exactly? List at least two reasons why this is the case?

1.6.2 Uncertainty Propagation

As discussed in Question 2 above, there various reasons why a model cannot be calibrated perfectly. Some of these are:

- lack of data;

- the existence of measurement noise;
- the fact that the model is just not perfect.

Ignoring for the moment the possibility that the moment is just bluntly wrong, we see that the lack of data or the presence of noise will induce some uncertainty in the values of the calibrated parameters. We are going to represent uncertainty on parameters by assigning a probability density on them. There are systematic ways of estimating the uncertainty induced because of the calibration process, but this will not concern us now. For the moment, assume that somebody told us that the uncertainty in the scaled parameters ξ_i of the model is as follows:

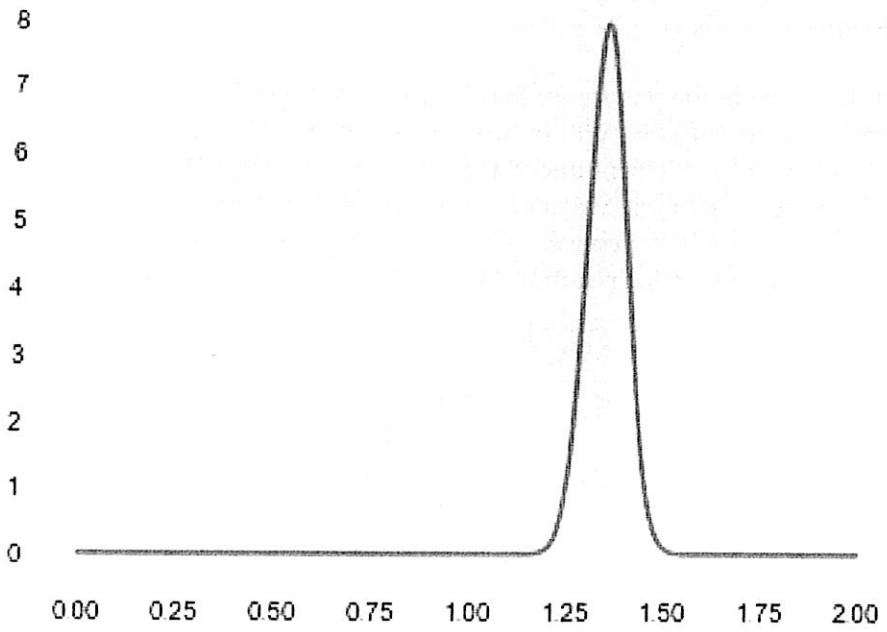
Variable	Value
ξ_1	1.35 ± 0.05
ξ_2	1.65 ± 0.08
ξ_3	1.34 ± 0.11
ξ_4	-0.16 ± 0.16
ξ_5	-3.84 ± 0.20

But what does this information actually mean? As we will discuss in the following lectures, this information can be used to assign a probability density on each one of these parameters, say $p(\xi_i)$, that *models* our state of knowledge about them. For example, let us assume that our state of knowledge about ξ_1 is given by a Gaussian probability density:

$$p(\xi_1) = \mathcal{N}(\xi_1 | \mu_1 = 1.35, \sigma^2 = 0.05^2),$$

which we can visualize as follows:

```
In [7]: import scipy.stats
        from scipy.stats import norm
        xi1 = np.linspace(-0, 2, 200)
        plt.plot(xi1, norm.pdf(xi1, loc=1.35, scale=0.05))

Out[7]: [<matplotlib.lines.Line2D at 0x11a81e790>]
```

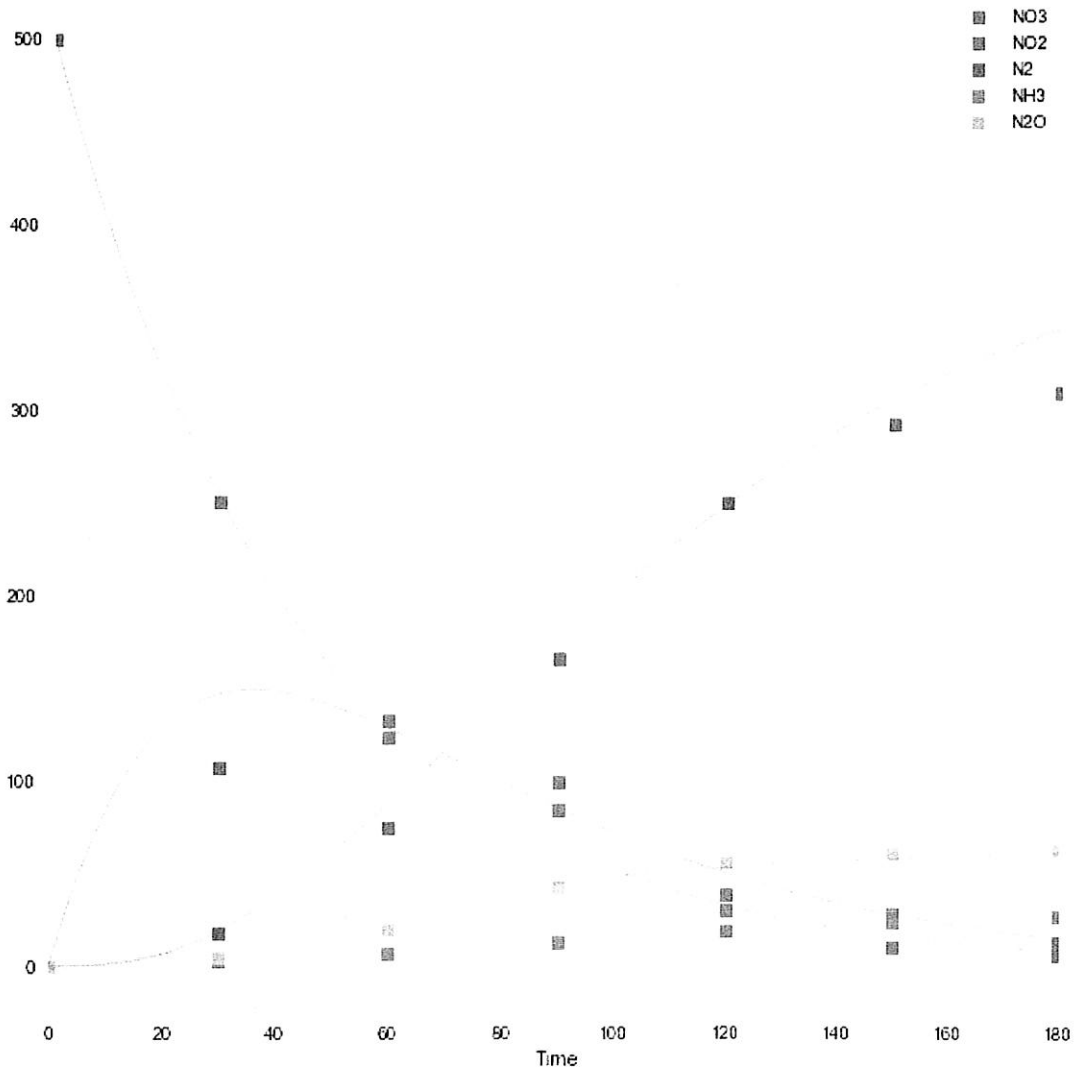


This means that we do not beleive that the value of the parameter can be less than 1.0 or greater than 1.6. Note that, we are deliberately trying to avoid the use of the term "random". There is nothing random in our example. Probability models a state of knowledge.

How does this uncertainty propagate through the model? We will study this question with a simple numerical experiment. We are going to assign Gaussian probability densities on all the ξ_i's, sample them a few times, and run our catalysis model for each one.

```
In [8]: def plot_samples(mu1 = 1.359, sig1=0.055,
                        mu2 = 1.657, sig2=0.086,
                        mu3 = 1.347, sig3=0.118,
                        mu4 = -.162, sig4=0.167,
                        mu5 = -1.009, sig5=0.368,
                        num_samples=1):
            """
            Take a few samples of the model to study uncertainty propagation.
            """
            fig, ax = plt.subplots(figsize=(10, 10))
            catalysis_data.plot(ax=ax, style='s')
            t = np.linspace(0, 180, 100)
            for i in xrange(num_samples):
                xi1 = norm.rvs(loc=mu1, scale=sig1)
                xi2 = norm.rvs(loc=mu2, scale=sig2)
                xi3 = norm.rvs(loc=mu3, scale=sig3)
                xi4 = norm.rvs(loc=mu4, scale=sig4)
                xi5 = norm.rvs(loc=mu5, scale=sig5)
                kappa = np.exp([xi1, xi2, xi3, xi4, xi5]) / 180.
                y = rk45(f_catalysis, (500., 0., 0., 0., 0., 0.), t, args=(kappa,))
```

```
        ax.plot(t, y[:, 0], linewidth=0.5, color=sns.color_palette()[0])#, label='Mode
        ax.plot(t, y[:, 1], linewidth=0.5, color=sns.color_palette()[1])#, label='Mode
        ax.plot(t, y[:, 2], linewidth=0.5, color=sns.color_palette()[5])#, label='Mode
        ax.plot(t, y[:, 3], linewidth=0.5, color=sns.color_palette()[2])#, label='Mode
        ax.plot(t, y[:, 4], linewidth=0.5, color=sns.color_palette()[3])#, label='Mode
        ax.plot(t, y[:, 5], linewidth=0.5, color=sns.color_palette()[4])#, label='Mode
    plt.legend()

interactive(plot_samples, mu1 = (-2, 2, 0.05), sig1=(0.02, 0.4, 0.01),
                                  mu2 = (-2, 2, 0.05), sig2=(0.02, 0.4, 0.01),
                                  mu3 = (-2, 2, 0.05), sig3=(0.02, 0.4, 0.01),
                                  mu4 = (-2, 2, 0.05), sig4=(0.02, 0.4, 0.01),
                                  mu5 = (-2, 2, 0.05), sig5=(0.02, 0.4, 0.01),
          num_samples=(1, 1100, 10))
```



1.6.3 Questions

1) Increase the number of samples from 1, to 10, to 100, to 1000. Each time you get a better description of uncertainty.

2) Ok, the more samples you get the better your predictive error bars. But can you do this with any model? When would you face difficulties with such a program? What if you want to propagate uncertainties through a very complicated model, e.g., a climate model, which may take a few hours to complete a single simulation?

3) Can you come up with any idea of accelerating the uncertainty propagation process?