PHIMECA

… solutions for robust engineering

Uncertainty treatment in dispersion modelling of accidental releases

OpenTURNS Users Day #8. June 12, 2015

Felipe Aguirre Martinez

PP-01-G

Sommaire

- Dispersion modeling in the presence of uncertainty
- Risk assessment methodology for urgent situations
- First results
- Second study – Some lessons learned so far
- Conclusions

First study : École Centrale de Lyon

Vincent Dubourg, Patrick Armand, David Poulet,
Florian Vendel, Sébastien Argence, Thierry Yalamas,
Fabien Brocheton and Perrine Volta

DE LA RECHERCHE À L'INDUSTRIE cea

PHIMECA

NUMTECH

Sillages ENVIRONNEMENT

First study : École Central de Lyon

- *High fidelity* atmospheric dispersion modelling…

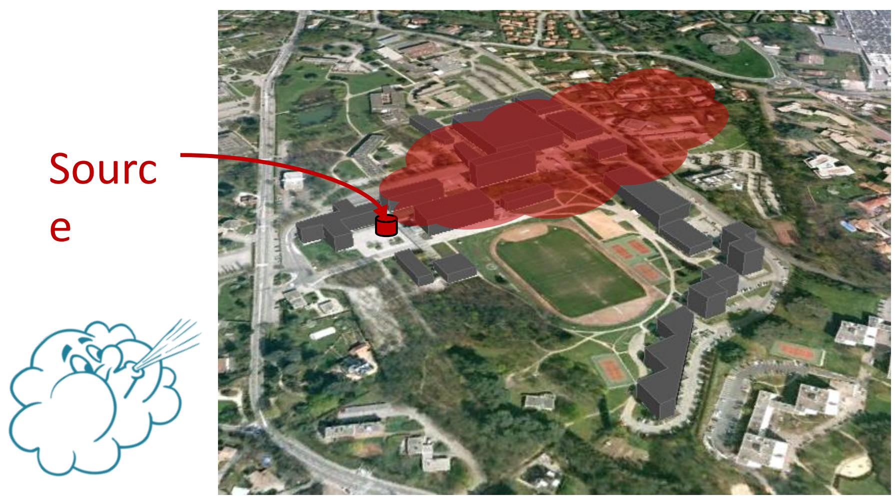
- … increasingly depends on *our knowledge of the exact environmental conditions*.
- Such conditions are unknown to some extent, especially in the case of accidental releases.

> We propose *a risk assessment framework* that accounts for such uncertainty in the form of *probability distributions*.

First study : École Central de Lyon

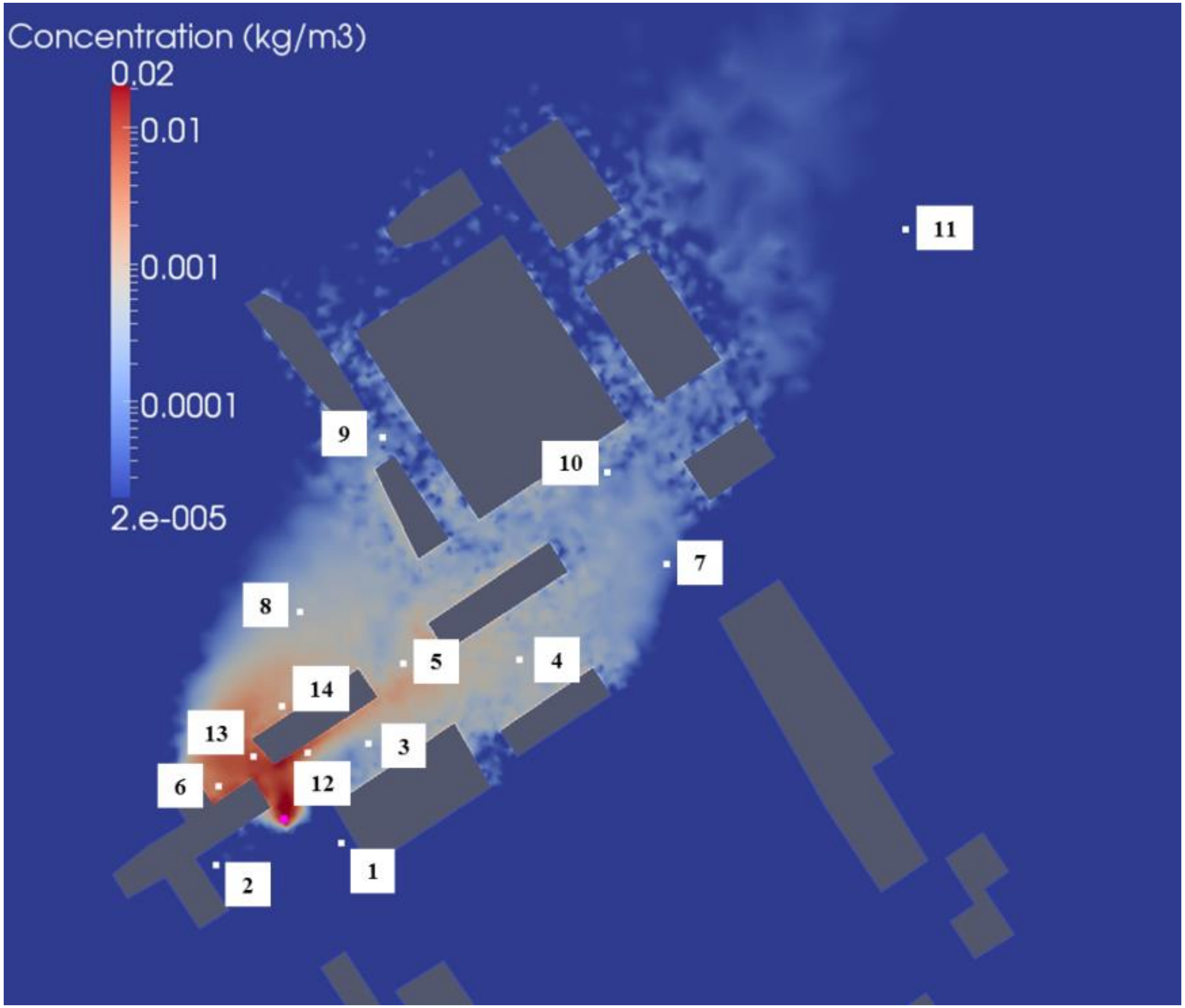
Dispersion modelling

- The exact *source location* is *supposedly known*.
- The release *lasts* 5:00 minutes
- *Meteorological conditions* (wind speed, direction, etc. …) are *uncertain (imprecise)*.
- A *Lagrangian model* (SLAM) is used for simulating the dispersion of the pollutant (assuming a light gas behaviour).
- A *pre-computed CFD database* enables the calculation of the *perturbed wind field* in the constructed area in the vicinity of the source for *a large variety of incident winds* (using multi-linear interpolation).



First study : École Central de Lyon

- Dispersion modelling

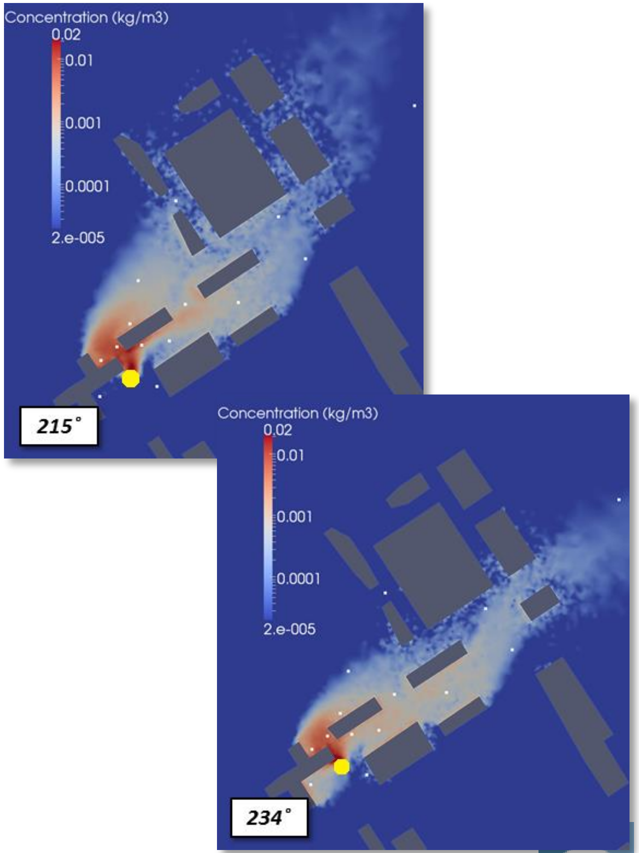


First study : École Central de Lyon

Uncertainty modelling

- The *lack of knowledge* about some parameters describing the release conditions is modelled as *a probability distribution*.
- These variables are assumed independent.

Parameter	Probability distribution
Wind speed	Gaussian with mean 2 $m.s^{-1}$ and standard deviation 0.17 $m.s^{-1}$
Wind direction	Truncated Gaussian with mean 225° and standard deviation 22.15°, over [215°; 234°]
Cloud	Truncated Gaussian with mean 6 octas and standard deviation 1 octa, over [1 octa; 9 octas]
Temperature	Uniform over [14°C; 16°C]
Emitted quantity	Uniform over [70 $kg.s^{-1}$; 130 $kg.s^{-1}$]
Source height	Uniform over [1.75 m; 2,25 m]



Dispersion & uncertainty modelling

Quantity of interest for risk assessment

- We consider the *cumulated dose causing irreversible effects on human health* according to INERIS recommandations for *phosphine* :

$$D(\boldsymbol{X}, \boldsymbol{p}, t) = \int_0^t C_{\mathrm{PH}_3}(X, \boldsymbol{p}, \tau)^n \mathrm{d}\tau$$

  where :
  - $\boldsymbol{X}$ denotes the random vector of *uncertain release conditions*
  - $\boldsymbol{p}$ and t are the *position* and *exposure time* respectively
  - C_{PH_3} is the *instant phosphine concentration* calculated by SLAM
  - $n = 0.53$ according to INERIS
- The subject is assumed *not to move* during exposure.
- The *risk analysis* consists in estimating:

$$p = \mathrm{Prob}[D(\boldsymbol{X}, \boldsymbol{p}, t) > D_0]$$

  where $D_0 = 20.10$ according to INERIS.

Sommaire

- Dispersion modeling in the presence of uncertainty
- Risk assessment methodology for urgent situations
- First results
- Second study – Some lessons learned so far
- Conclusions

Risk assessment methodology

Brute-force approach

- The spatio-temporal field of exceedance probabilities can be estimated using Monte Carlo sampling :

$$\hat{P}(\boldsymbol{p}, t) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(D\left(\boldsymbol{X}^{(i)}, \boldsymbol{p}, t\right) > D_0\right)$$

- This estimator converges as the number of samples (the number of SLAM runs) increases.

- A minimum of 10 000 samples is required in order to achieve a reasonable coefficient of variation of 32% on a probability of 10^{-3}.

- With 10 minutes per simulation, this would take two months !

Such a large number of SLAM runs is *incompatible with the urgency associated to accidental releases scenarii*.

Risk assessment methodology

We propose to replace SLAM by a *surrogate model*
that is *much faster to evaluate*.

Elements of surrogate modelling

DOE

- Run the model $\mathcal{M}$ on a well-chosen set of input (gathered in an experimental design).
- The purpose is to capture the largest amount of information about the functional relationship between its input $\boldsymbol{x}$ and output $\boldsymbol{y}$.

fit

- Choose a family of surrogate models amongst artificial neural networks (ANN), support vector machine (SVM), Gaussian processes (GP), generalized linear models (LM).
- Compute the surrogate model parameters from the dataset $\mathcal{D} = \left(\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right), i = 1, \ldots, m\right)$.

validate

- Compute summary statistics about the relative error between the original model and its approximation.
- The purpose is to qualify the surrogate model on a bounded domain of the input space.

predict

- Use the surrogate model instead of the original model to speed up uncertainty quantification or optimization post-processings.

Risk assessment methodology

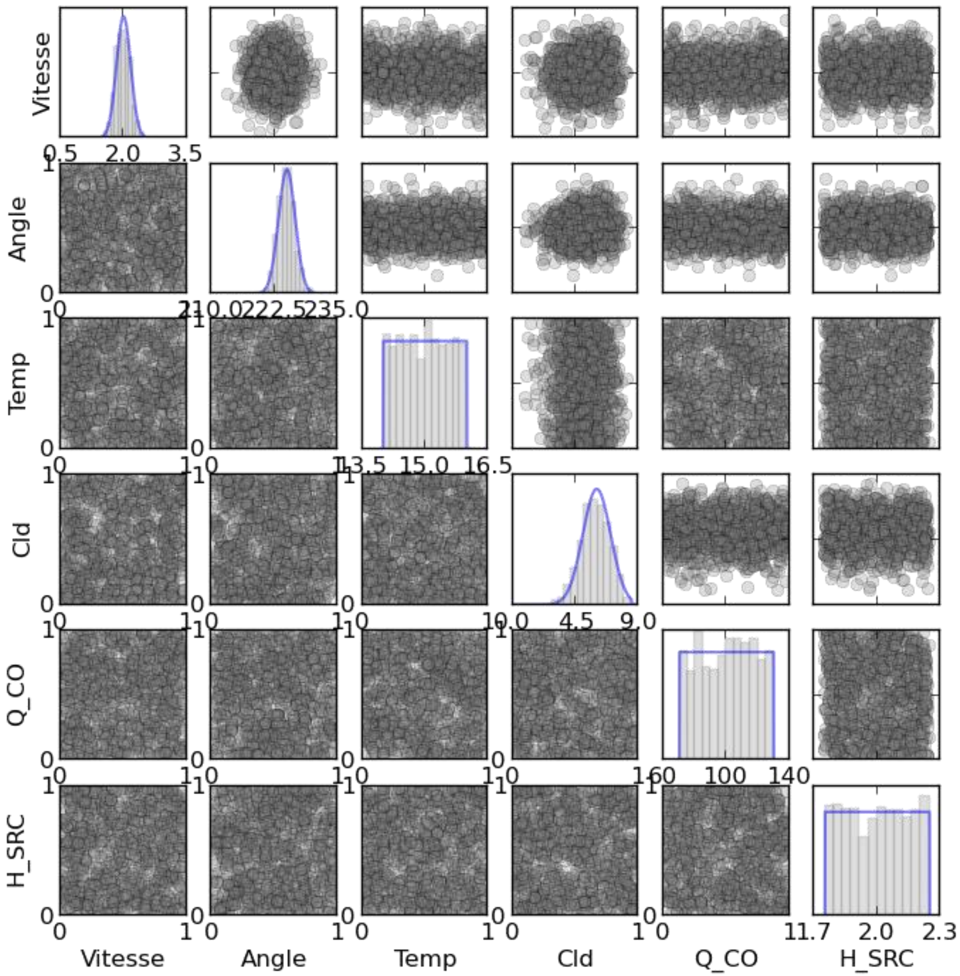
Dimension reduction using principal component analysis (PCA)

- We could apply kriging *for all* $\boldsymbol{p}$ *and* t *over a spatio-temporal grid* in order to surrogate the whole output of SLAM.
- But this would be *heavy/long for dense grids* ($N_x \times N_y \times N_t = 50 \times 50 \times 71$, for the present application)!
- It is proposed to exploit the *significant spatio-temporal correlation* (coherence) that exists in the output of SLAM for reducing its dimension to *a minimal vector of principal components*.
- *Kriging is then applied to each component of the reduced vector* $\boldsymbol{z}$ instead of the original one (the inverse transform is used at predict time).

Sommaire

- Dispersion modeling in the presence of uncertainty
- Risk assessment methodology for urgent situations
- First results
- Second study – Some lessons learned so far
- Conclusions

Design of experiments

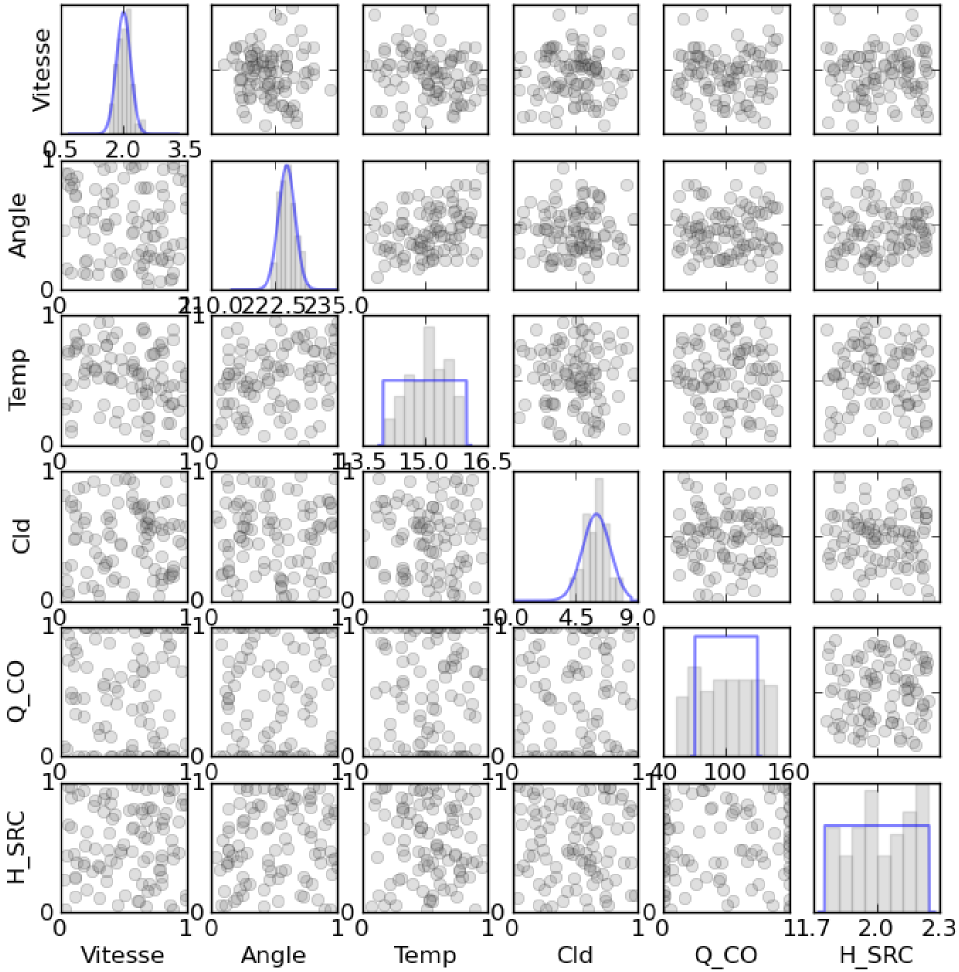


- Monte Carlo experiment used for validation of the surrogate-based approach ($N = 1{,}000$)

- Simulations where distributed on Hyperion (CICT)
  - OpenTURNS Python wrapper
  - PBS scheduler
  - 512 CPUs with 4Go RAM per CPU
  - 1000 simulations finished in ~40 minutes

PHIMECA

Design of experiments

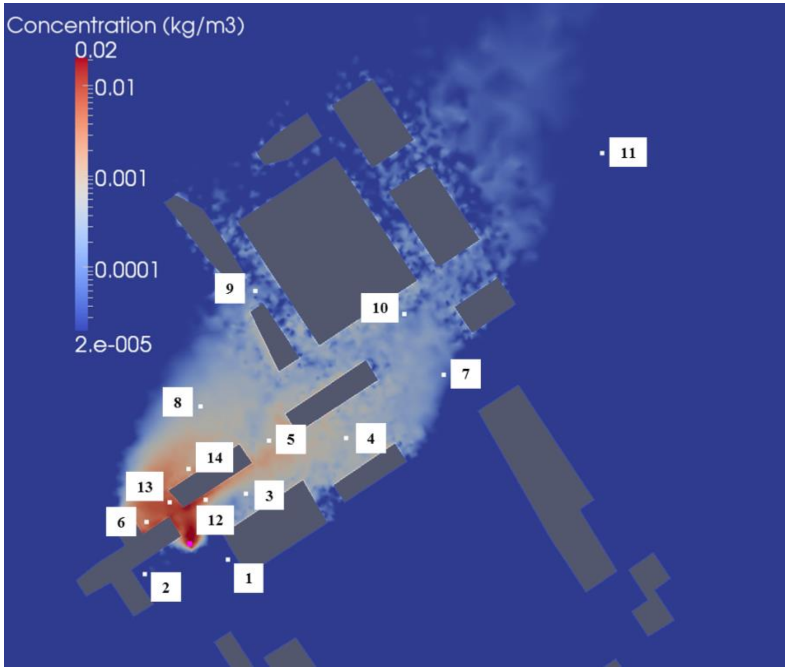


- Design of experiments used for the surrogate modelling
- K-means clustering
- Subset selection in the previous Monte Carlo experiment, $m = 100$

Résultats sur la grille plane

Validation of the surrogate models

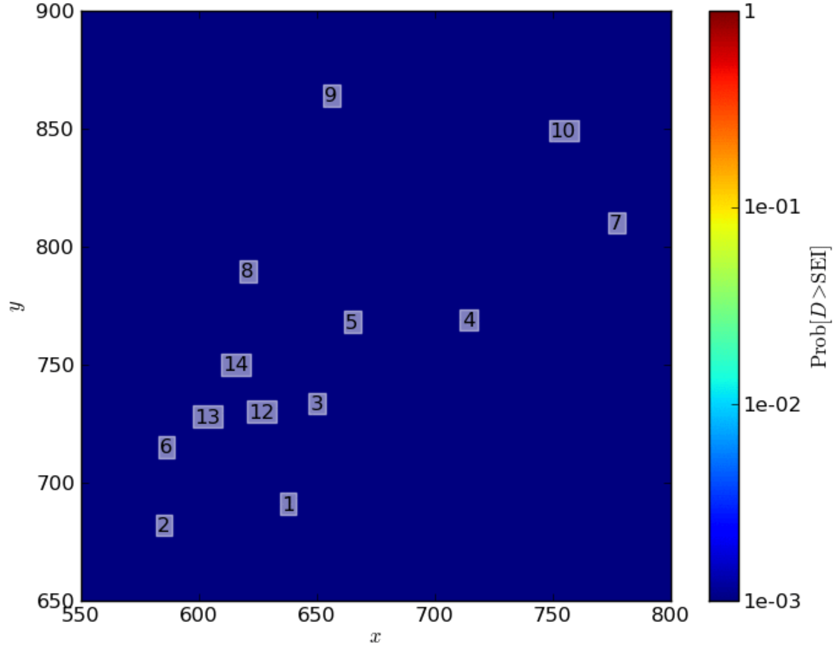
- One surrogate model per time step (PCA over the space)
- Indicators indicate the average over the 10^4 points on the field.



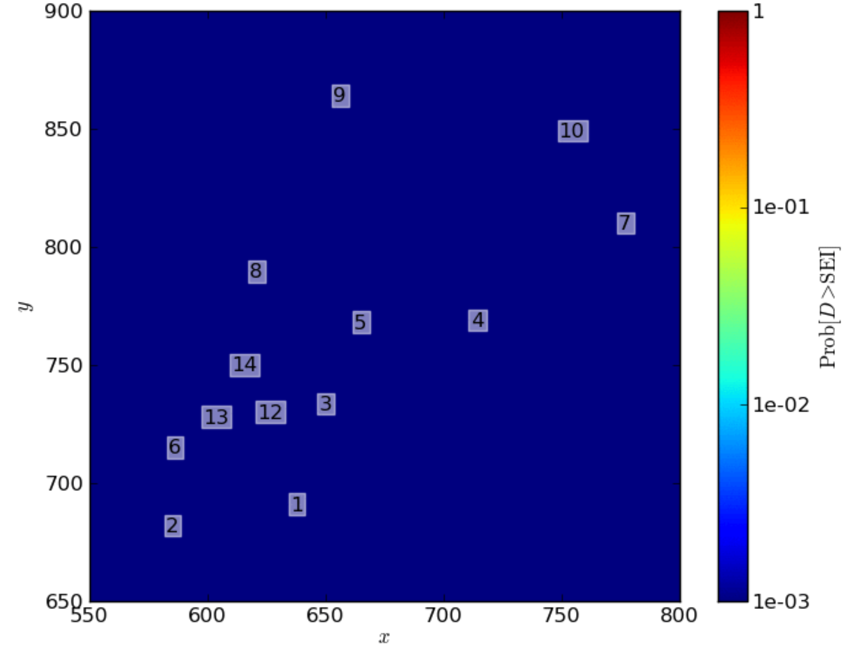
Pas de temps	$r << d$ (d = 2500)	Q^2	R^2 (test)	Pas de temps	$r << d$ (d = 2500)	Q^2	R^2 (test)
1	79	0.92	0.90	**37**	93	0.68	0.56
2	79	0.92	0.90	**38**	93	0.68	0.59
3	79	0.92	0.90	**39**	93	0.69	0.60
4	88	0.81	0.76	**40**	93	0.68	0.59
5	88	0.82	0.76	**41**	93	0.68	0.59
6	89	0.82	0.77	**42**	93	0.68	0.59
7	89	0.72	0.57	**43**	93	0.67	0.58
8	90	0.73	0.64	**44**	93	0.67	0.58
9	91	0.74	0.65	**45**	93	0.67	0.58
10	91	0.65	0.57	**46**	93	0.66	0.57
11	91	0.67	0.59	**47**	93	0.65	0.57
12	92	0.68	0.60	**48**	93	0.65	0.56
13	92	0.64	0.54	**49**	93	0.64	0.54
14	92	0.66	0.60	**50**	93	0.63	0.55
15	92	0.67	0.61	**51**	93	0.62	0.55
16	93	0.65	0.58	**52**	93	0.61	0.54
17	93	0.67	0.60	**53**	93	0.60	0.54
18	93	0.68	0.61	**54**	93	0.59	0.53
19	93	0.67	0.55	**55**	93	0.58	0.53
20	93	0.68	0.60	**56**	93	0.58	0.52
21	93	0.68	0.61	**57**	93	0.57	0.52
22	93	0.67	0.51	**58**	93	0.56	0.50
23	93	0.68	0.60	**59**	93	0.55	0.50
24	93	0.69	0.60	**60**	93	0.54	0.49
25	93	0.67	0.57	**61**	93	0.53	0.49
26	93	0.68	0.60	**62**	93	0.53	0.49
27	93	0.68	0.60	**63**	92	0.52	0.48
28	93	0.68	0.55	**64**	92	0.51	0.48
29	93	0.68	0.59	**65**	92	0.50	0.47
30	93	0.69	0.60	**66**	92	0.49	0.47
31	94	0.68	0.57	**67**	92	0.48	0.46
32	94	0.68	0.59	**68**	92	0.47	0.45
33	94	0.68	0.59	**69**	92	0.47	0.45
34	94	0.68	0.58	**70**	92	0.46	0.44
35	93	0.68	0.59	**71**	92	0.45	0.44
36	93	0.68	0.60				

Results

- Probability of exceeding the threshold dose of irreversible effects



Brute-force approach
$(N = 1{,}000)$



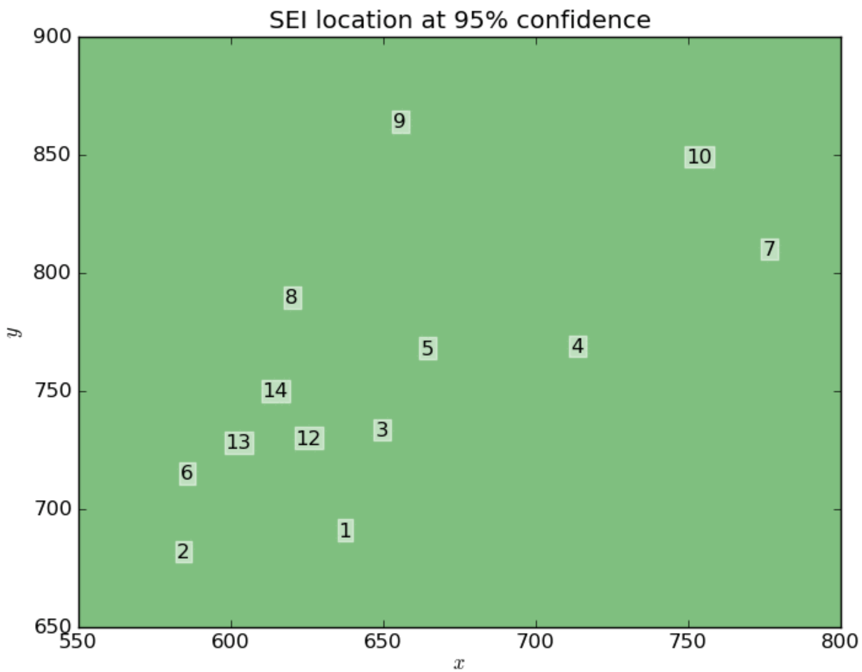
Surrogate-based approach
$(N = 10{,}000)$

- The surrogate-based approach accounts for *the uncertainty in the kriging predictor* (Gaussian) :

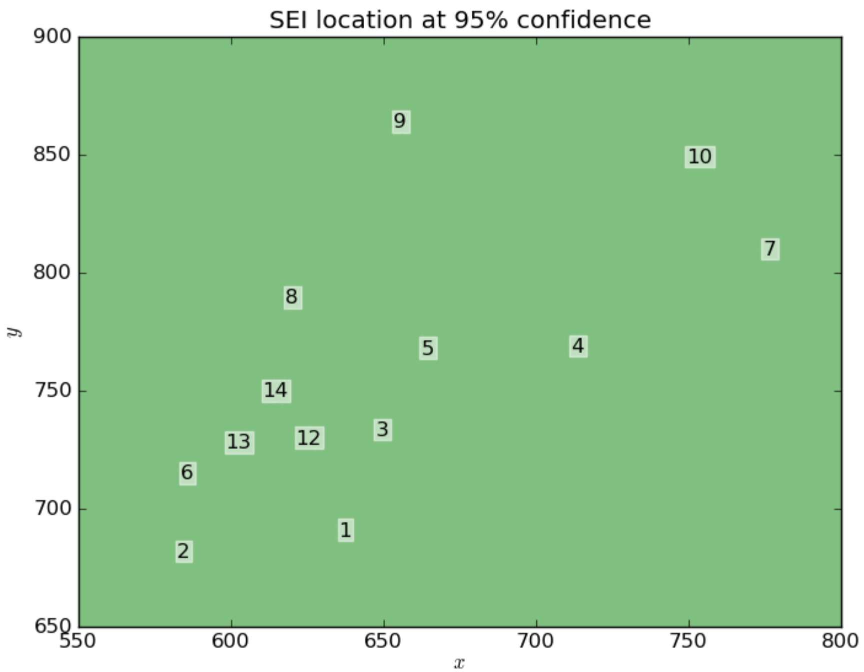
$$\hat{P}(\boldsymbol{p},t) = \frac{1}{N}\sum_{i=1}^{N} 1 - \Phi\left(\frac{D_0 - \mu_{\hat{Y}}\left(\boldsymbol{x}^{(i)},\boldsymbol{p},t\right)}{\sigma_{\hat{Y}}\left(\boldsymbol{x}^{(i)},\boldsymbol{p},t\right)}\right)$$

Results

Risk map



Brute-force approach
$(N = 1{,}000)$

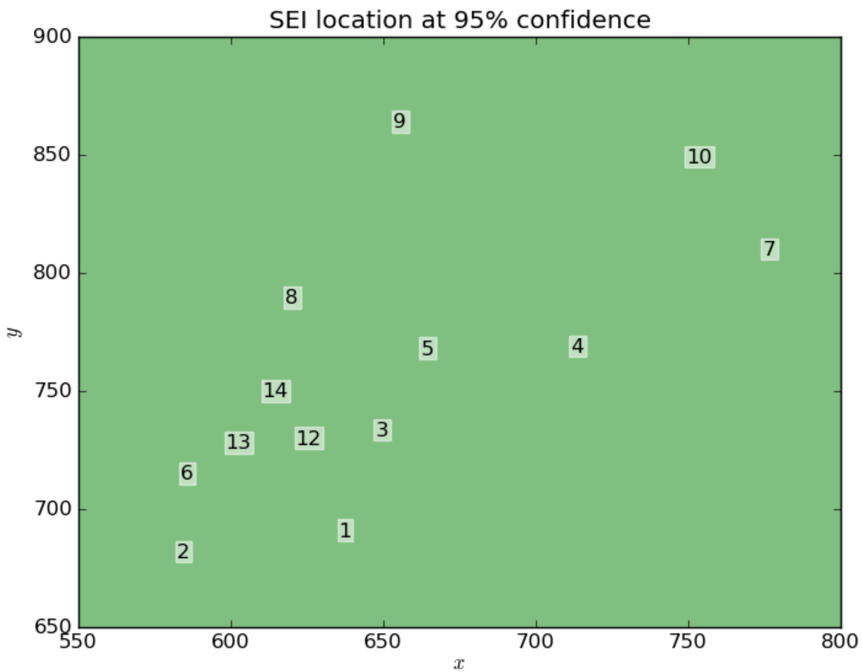


Surrogate-based approach
$(N = 10{,}000)$

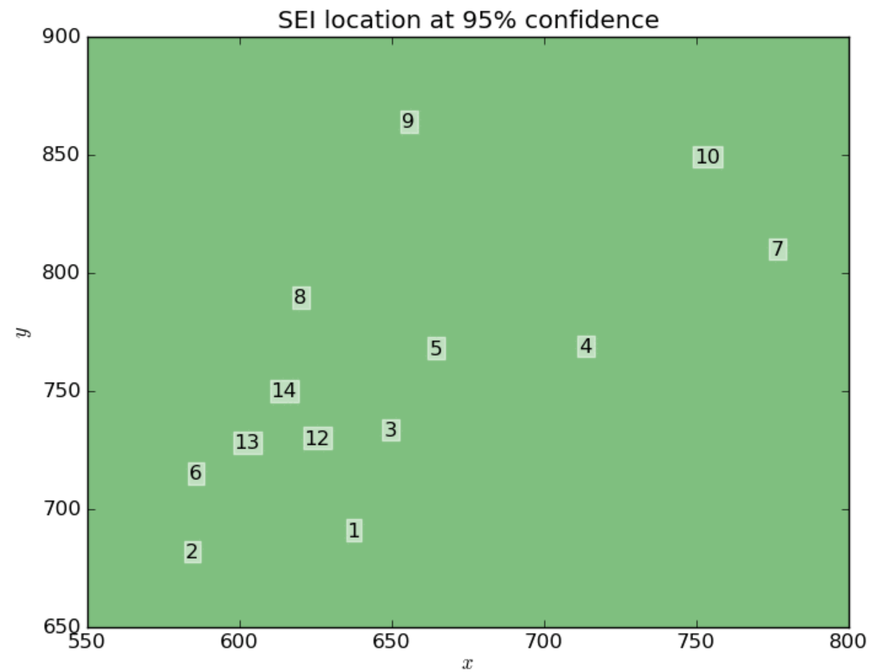
- The probability of exceeding the threshold dose of irreversible effects is :
  - less than 2.5 % in the green zone ;
  - between 2.5 % and 97.5 % in the orange zone ;
  - larger than 97.5 % in the red zone.

Results

Risk map (with different emitted quantity distributions)



$Q_{\mathrm{PH}_3} \sim \mathcal{U}([70\ \mathrm{kg.s}^{-1}; 130\ \mathrm{kg.s}^{-1}])$



$Q_{\mathrm{PH}_3} \sim \mathcal{U}([7\ \mathrm{kg.s}^{-1}; 13\ \mathrm{kg.s}^{-1}])$

- An *arbitrarily large emitted quantity distribution* was first used for *reaching the threshold of irreversible effects in the far field*.
- A *smaller emitted quantity distribution* eventually *augments the spread* of the uncertain (orange) zone.

Sommaire

Second study on a larger scale

- Objective : Apply the same methodology on a city scale
- Specifications:
  - Computational chain with more than one code to run (~200 different calls)
  - Runtime: ~90 minutes over 64 cores…
  - Spatio-temporal grid with :
    - 60 time steps
    - Spatial grid of :
      - One tile 360 x 440 for a simplified test model
      - 63 tiles of size 430 x 430 each for the real model
  - Number of output variables :60 x 63 x 430 x 430 ~= 70 millions !
    - Test model : 60 text files of 7MB each
    - Real model : 1360 text files of 7MB each
  - Distribution of simulations on TGCC through submission scripts

- The big challenge here lays on the complexity of the model.

Parsing big text files

- Each simulation produces :
  - Test model : 60 text files of 7MB each
  - Real model : 1360 text files of 7MB each
- Pandas parses files much faster than any other solution

```
In [4]: %timeit np.atleast_2d(np.loadtxt(conc_file, skiprows=1, usecols=[4]))
1 loops, best of 3: 981 ms per loop

In [5]: %timeit np.atleast_2d(pd.read_csv(conc_file, sep='\s+', usecols=['C[ppmV]'])).T
10 loops, best of 3: 59.1 ms per loop
```

- 16 times faster !
- But it introduces a dependency…. ☹
- Distributing file parsinig

```
from sklearn.externals.joblib import Parallel, delayed
Results = Parallel(n_jobs=30)(delayed(func)(thing) for thing in iterator)
```

Handling and logging errors

- When submitting jobs through submission scripts, you loose track of the execution !

- Protect your wrapper with a try/except structure !

```
class Wrapper(ot.OpenTURNSPythonFunction):
    def _exec(self, X):
        try:
            #Do stuff
        except Exception, e:
            logger.error(e, exc_info=True)
            raise e
        return Y
```

- Or use a decorator ☺ !
  - More details af the end if need be

Handling output

- It is impossible to keep a numerical sample per simulation due to memory limits !
  - Primarily due to the fact that each run is an independent job submission

- Dump results to disk at the end for later post treatment.

- But one run represents ~300 mb per text files → ~1.8Gb….
  - Use gzip for compression and pickle.dump(array, file, protocol=2) for speed

```
def dump_array(array, filename):
    with gzip.open(filename, 'wb') as fh:
        pickle.dump(array, fh, protocol=2)
```

- HDF5 and netCDF to be tested !

Using argsparse

- Usually we run our simulations from an Ipython interpreter…
- But on clusters you often need to go through submission scripts !
- Create a command line interfase of the wrapper using argparse !
  - `python wrapper.py -X 170 3 0.05`
  - `python wrapper.py -MonteCarlo -N 1000`

```
if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser(description="Python wrapper example.")
    parser.add_argument('-X', nargs=3, metavar=('X1', 'X2', 'X3'),
        help='Vector on which the model will be evaluated')

    args = parser.parse_args()
    X = ot.NumericalPoint([float(x) for x in args.X])
    Y = model(X)
    dump_array(X, 'InputSample.pkl')
    dump_array(Y, 'OutputSample.pkl')
```

Conclusion

- *Probabilistic modelling* is used to describe uncertain *release conditions*.
- *Risk* is assessed as the *probability of exceeding a critical dose*.
- *Surrogate modelling* enables a *drastic speed-up* in the production of risk maps :
  - provided the CFD database is already computed (for industrial sites at risk) ;
  - *20 minutes per SLAM run* in the DOE ($\times$ 100 runs, but $\times \frac{1}{N_{CPUs}}$ using HPC) ;
  - *about 12 seconds per time step* for fitting the kriging predictors ;
  - *about 25 seconds per time step* to predict the 10,000 configurations required for the final probability estimation.
- *Kriging* is a convenient surrogate for *incorporating the uncertainty about the surrogate model* in the final risk maps.
- Risk can be represented as *time-varying maps of dose exceedance probabilities*.

The killer wrapper !

- It is able to run on different environments:
  - Workstation
  - Office made heterogenous clusters (e.g., IPython parallel with SSH),
  - HPC through submission scripts (e.g., TGCC, Hyperion ou Poincare)
  - Cloud solutions (e.g. Simulagora ou DominoUp)
- It catches and logs errors for easy debugging
- It can either run or simply prepare runs
  - Usefull when using clusters
- You can use it as a script (argsparse module):
  - `python wrapper.py -X 170 3 0.05`
- It is by default an ot.NumericalMathFunction ! (decorators !)

- It might seem complex, but wrappers are repetitive. A good cookbook might be enough to spread this to the community !

Handling and logging errors

```
from functools import wraps
def debug(func, logger):
    @wraps(func)
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except Exception, e:
            logger.error(e, exc_info=True)
            raise e

    return wrapper


class Wrapper(ot.OpenTURNSPythonFunction):
    @debug(logger)
    def _exec(self, X):
        #Do stuff
        return Y
```

- Take a look at David Beazley's tutorial for PyCon'2013