Uncertainty quantification using DAKOTA

Clara García-Sánchez

Ivan Pađen

Nadine Hobeika

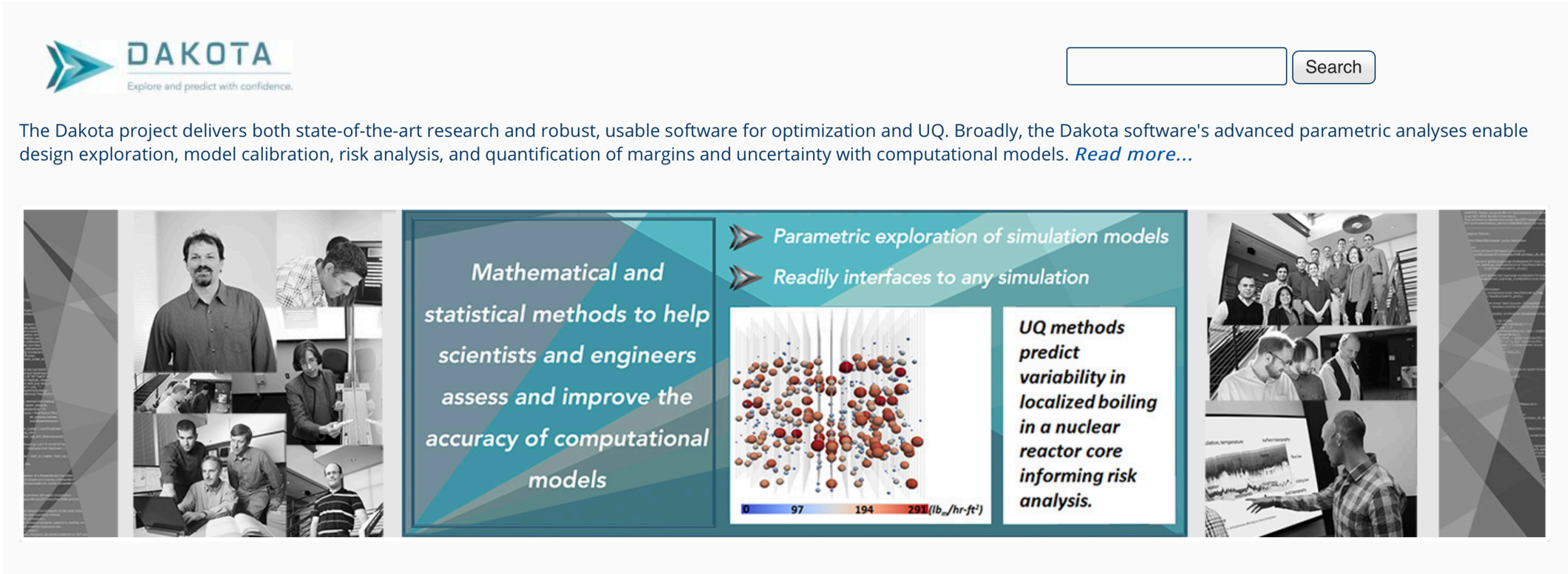
Past contributors:
Jorge Sousa
Catherine Gorlé

TU Delft

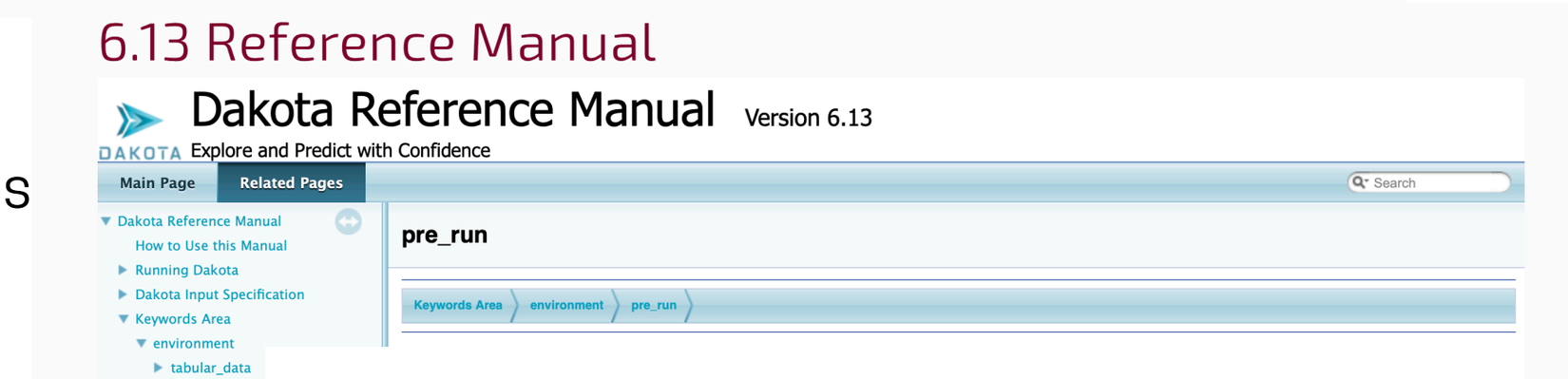
DAKOTA
Explore and predict with confidence

Sandia National Laboratories

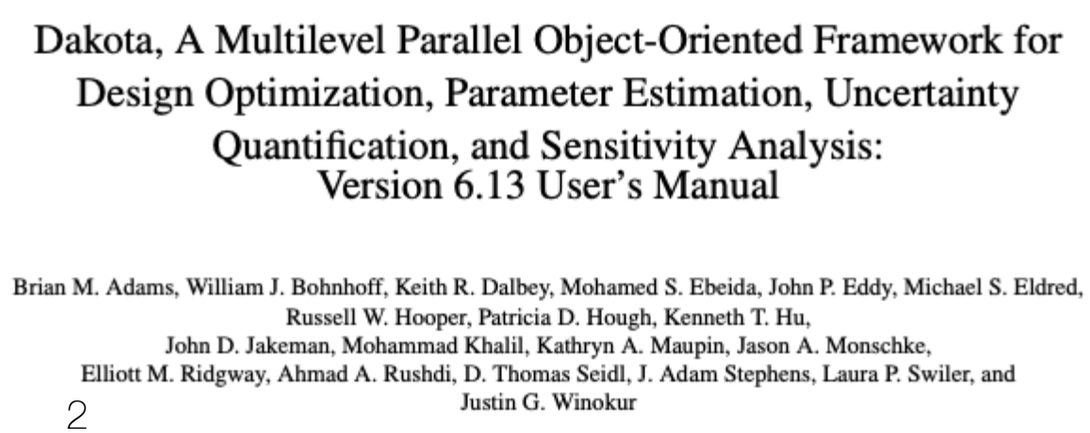
About DAKOTA



Online reference manual can be very useful, but sometimes a bit cryptic…



…when in doubt check the users manual in .pdf


Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.13 User's Manual

Brian M. Adams, William J. Bohnhoff, Keith R. Dalbey, Mohamed S. Ebeida, John P. Eddy, Michael S. Eldred, Russell W. Hooper, Patricia D. Hough, Kenneth T. Hu, John D. Jakeman, Mohammad Khalil, Kathryn A. Maupin, Jason A. Monschke, Elliott M. Ridgway, Ahmad A. Rushdi, D. Thomas Seidl, J. Adam Stephens, Laura P. Swiler, and Justin G. Winokur

TU Delft

About DAKOTA

https://dakota.sandia.gov/get-started

You can test the installation by running a simple example

Get Started

INSTALL DAKOTA

Download Dakota

Install Pre-built Dakota (recommended)

To get you up and running quickly, download the **binary executable** for your platform.

- Install Unix Variant Binary
- Install Windows Executable

Build from Source Code (advanced)

For custom installations, download and build Dakota from **source code**.

Read more

SET ENVIRONMENT

Set Environment

Set up platform variables for easier access to Dakota executables, examples, and tests.

- Unix Variants
- Windows

Read more

TEST INSTALLATION

Basic Test

Make sure Dakota runs. You should see Dakota version information when you type:

```
dakota -v
```

Other Tests

Run a larger suite of tests (requires Perl 5.x).

Read more

SIMPLE EXAMPLE

If you are new to Dakota, try the following example to demonstrate the basics of running Dakota.

This example demonstrates how to get started using Dakota either through the GUI or through the command-line.

You can also find this information in the Dakota User's Manual, Chapter 2, Quickstart, with links to related sections.

Read more

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

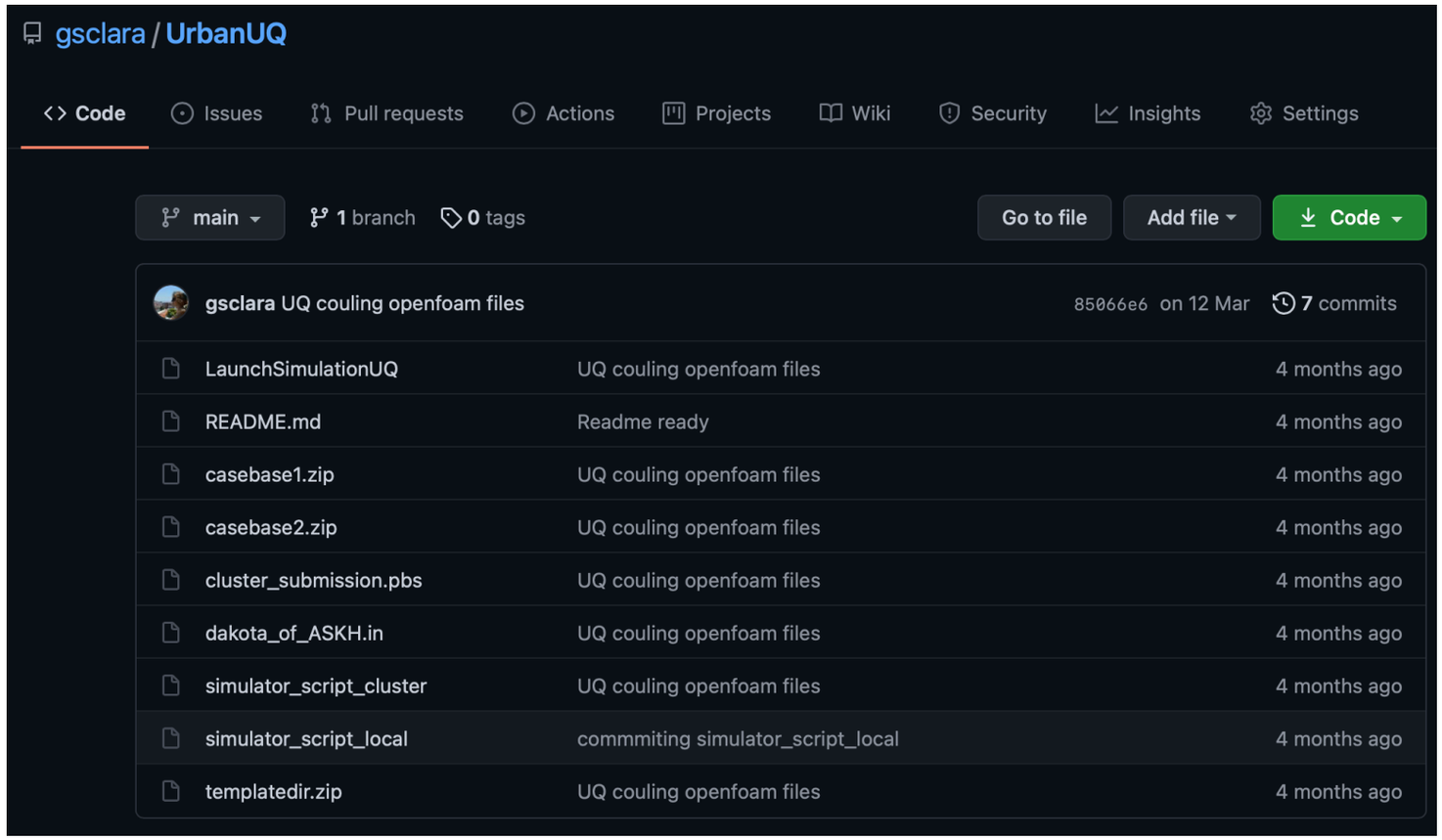
Running DAKOTA with OpenFOAM

Inside DAKOTA package there are some examples:

```
claragarciasan@gilfoyle:/opt/dakota-6.13$ pwd
/opt/dakota-6.13
claragarciasan@gilfoyle:/opt/dakota-6.13$ ls
bin  include  lib  share
claragarciasan@gilfoyle:/opt/dakota-6.13$ cd share/
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ ls
dakota  eigen3
claragarciasan@gilfoyle:/opt/dakota-6.13/share$ cd dakota/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ ls
Bash  COPYRIGHT  INSTALL  LICENSE  LICENSE_HOPSPACK  Python  README  VERSION  examples  test
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota$ cd examples/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ ls
README  advanced  cantilever  compiled_interfaces  eval_surrogate  generate_output.sh  hdf5  hopspack  linked_interfaces  parallelism  platforms  script_interfaces  training  users
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples$ cd script_interfaces/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ ls
Abaqus  Matlab  Nastran  Python  Rsmas  Scilab  TankExamples  VisualBasic  generic  text_book_wd
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces$ cd generic/
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$ ls
dakota_cleanup  dakota_rosenbrock.in  dakota_rosenbrock_windows.in  dprepro  rosenbrock_bb  simulator_script  simulator_script.bat  templatedir
claragarciasan@gilfoyle:/opt/dakota-6.13/share/dakota/examples/script_interfaces/generic$
```

Bash script coupling with OpenFOAM can be found the following repo:

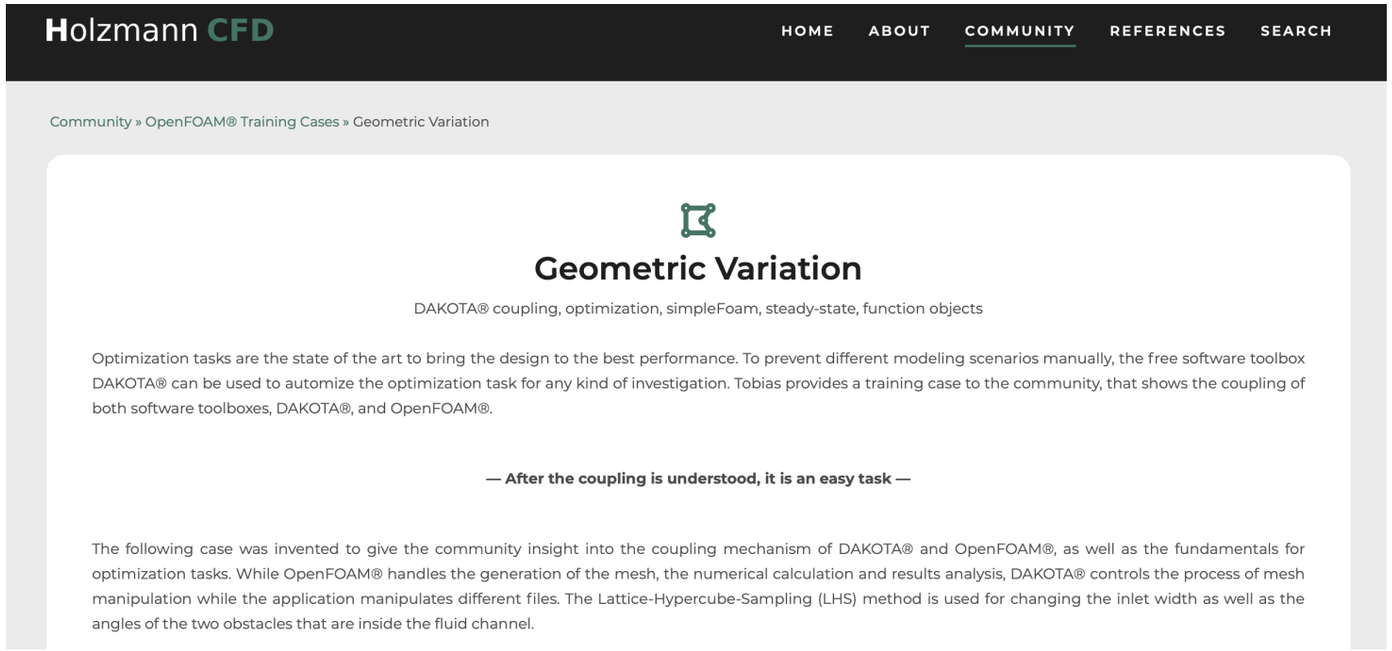
https://github.com/gsclara/UrbanUQ



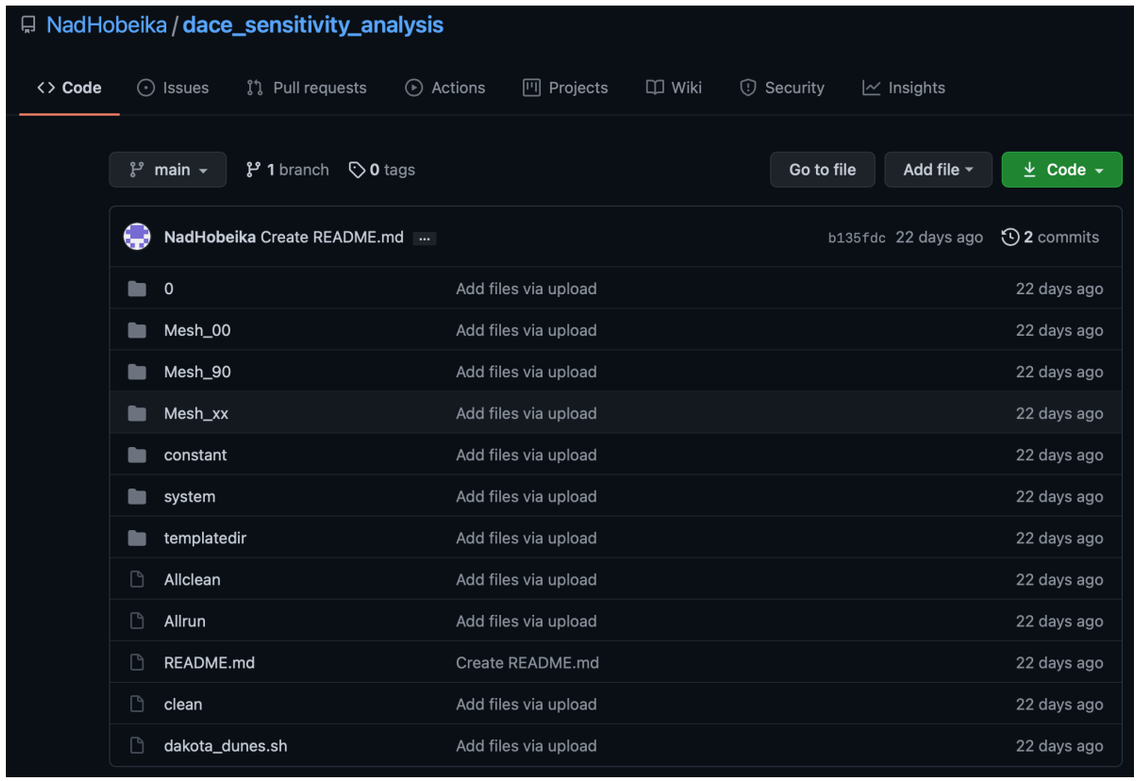
Running DAKOTA with OpenFOAM

"More elegant" (bash & python) coupling with OpenFOAM for optimisation can be found in:

https://holzmann-cfd.com/community/training-cases/geometric-variation



https://github.com/NadHobeika/dace_sensitivity_analysis



Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. Dakota input file and running options
5. UQ methodology
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

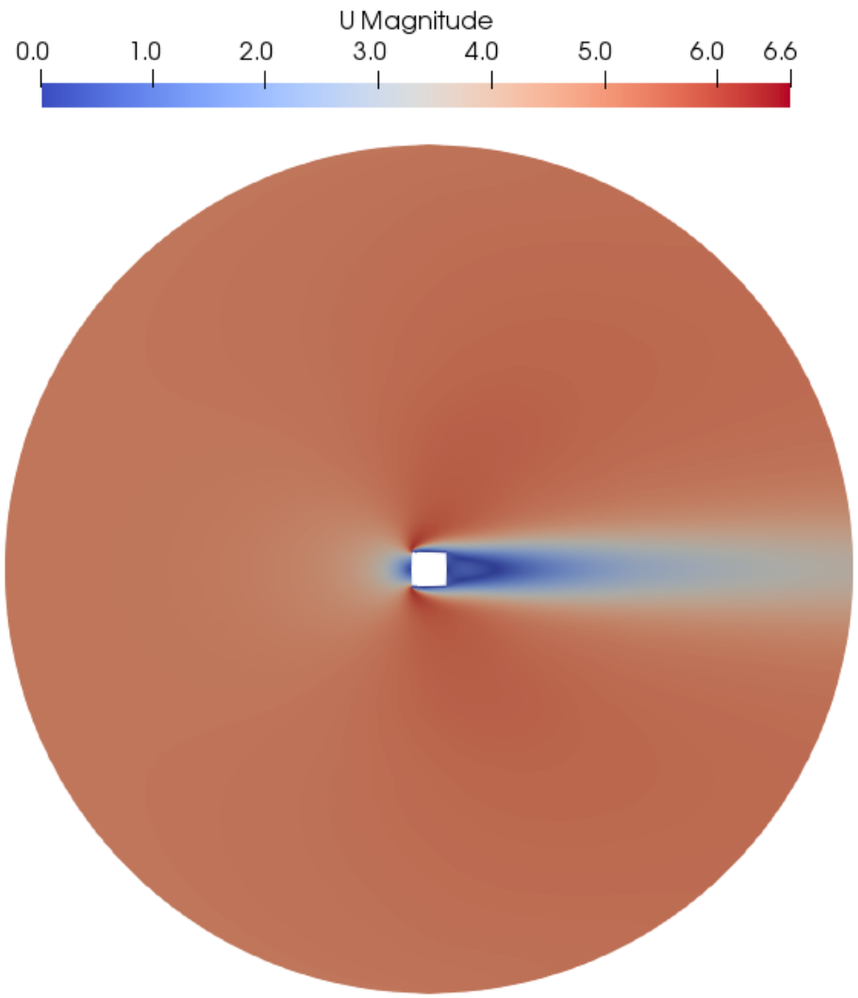
Here it is the outline of the tutorial:

1. **Introduce the run test case**
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. Dakota input file and running options
5. UQ methodology
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM

1. Run test case

- 2D flow around a cube within a circular domain
- Multiple wind directions available without mesh change
- Avoid mesh uncertainties
- Simulation is run 3000 iterations
- Turbulence model: kepsilon with standard constants
- <u>Uncertain variables:</u>
  - Inflow velocity
  - Inflow angle

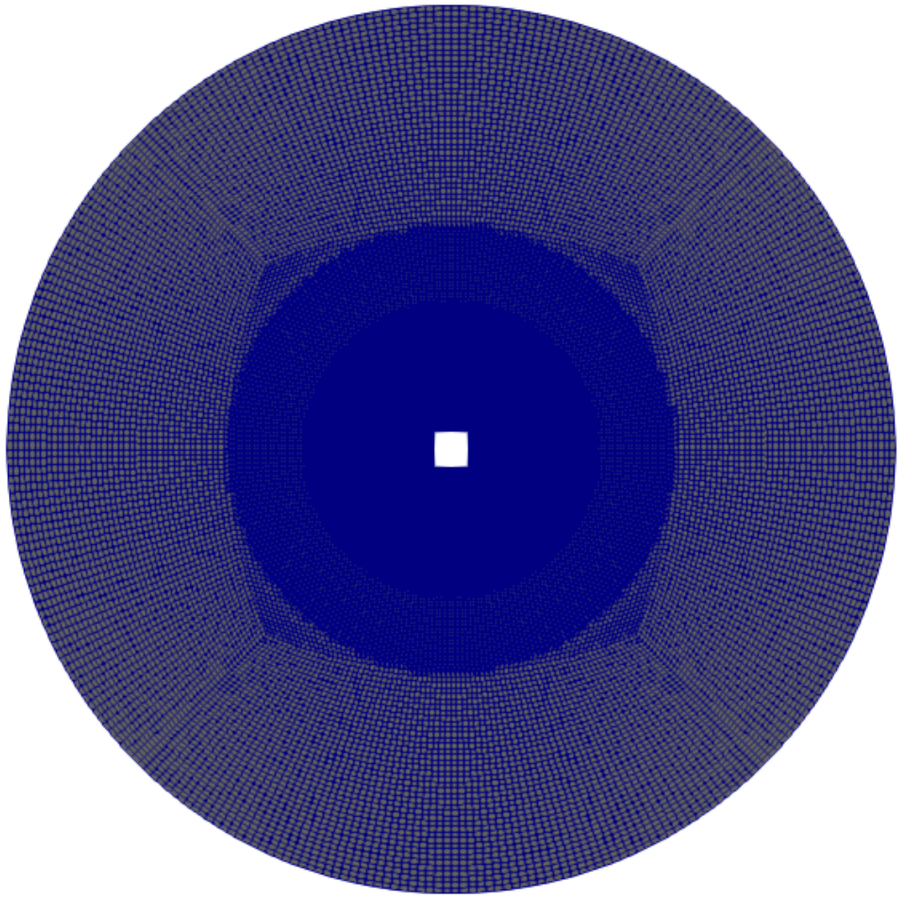


TU Delft

Running DAKOTA with OpenFOAM

1. Run test case

- Initial blockMesh generated with m4 application: https://www.ehsanmadadi.com/cylinder-mesh/



You can download the file from blockMeshDict.m4. .once you downloaded the file move the file to:

```
$case/system/blockMeshDict.m4
```

Then executing the m4 macro will generate the blockMeshDict in the same folder:

```
m4 blockMeshDict.m4 > blockMeshDict
```

As it is obvious, blockMesh should be executed afterwards.

- After running blockMesh, we run SnappyHexMesh with refinement areas, which creates more than one cell in vertical direction (almost 2D case, few changes in vertical)

- Total number of cells: ~400k

Running DAKOTA with OpenFOAM

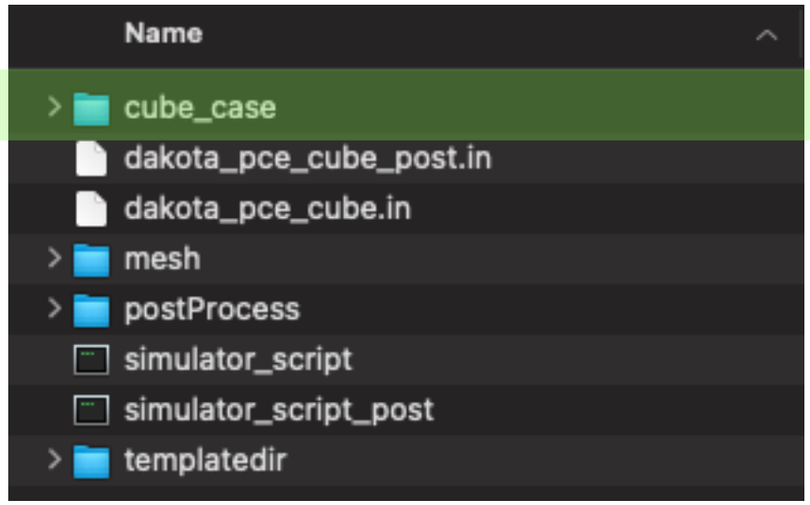
Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

1. Introduce the run test case
2. **Look at the folder for Dakota and the different files**
3. Simulator script line by line
4. Dakota input file and running options
5. UQ methodology
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM
2. Dakota folder structure

Template case

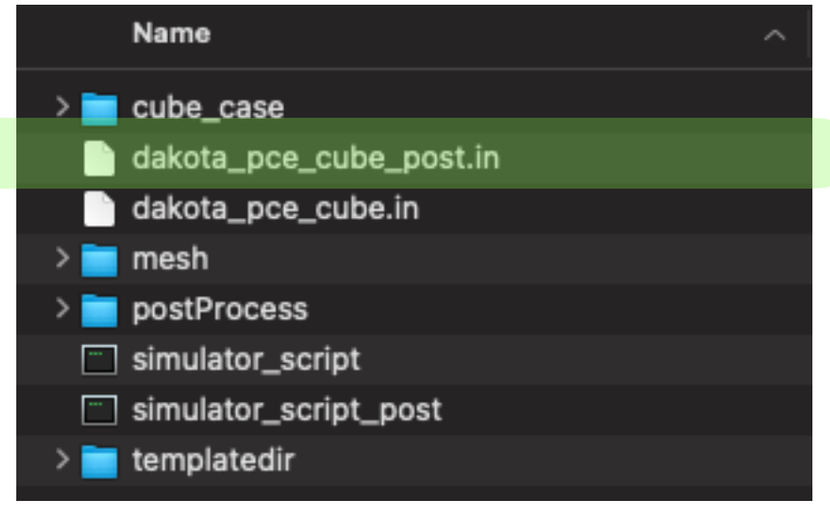


- This is the case that Dakota will copy to run the different evaluations
- For OF this means 0, constant and system folders (or additional folders you need)
- Important to only include the really necessary files, to reduce as maximum as possible the space usage
- In this case we have an empty folder in constant called polyMesh where we will link the mesh, instead of copying it

Running DAKOTA with OpenFOAM

2. Dakota folder structure

Alternative input file options that calls a simulator script with only postProcess



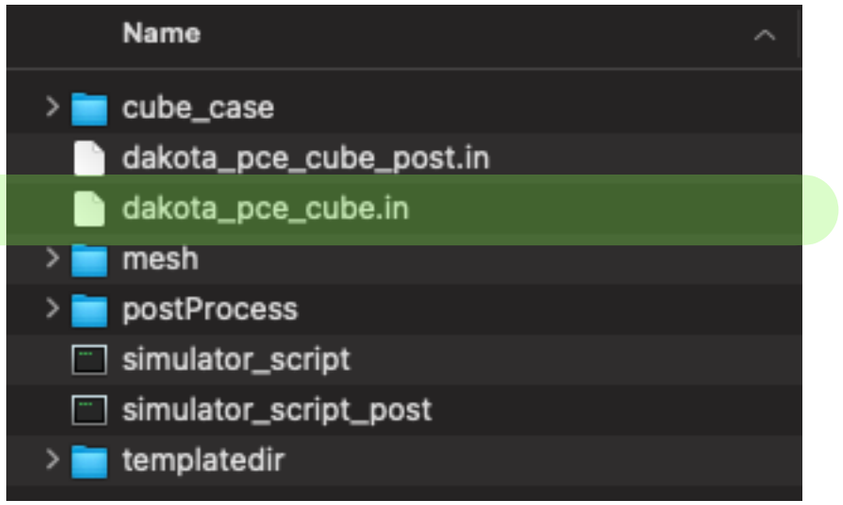

```
claragarciasan@gilfoyle:/ssd/gsclara/dakotaTutorial/dakotaCases/workdir.1$ dakota -help
usage: dakota [options and <args>]
        -help (Print this summary)
        -version (Print DAKOTA version number)
        -input <$val> (REQUIRED DAKOTA input file $val)
        -preproc [$val] (Pre-process input file with pyprepro or tool $val)
        -output <$val> (Redirect DAKOTA standard output to file $val)
        -error <$val> (Redirect DAKOTA standard error to file $val)
        -parser <$val> (Parsing technology: nidr[strict][:dumpfile])
        -no_input_echo (Do not echo DAKOTA input file)
        -check (Perform input checks)
        -pre_run [$val] (Perform pre-run (variables generation) phase)
        -run [$val] (Perform run (model evaluation) phase)
        -post_run [$val] (Perform post-run (final results) phase)
        -read_restart [$val] (Read an existing DAKOTA restart file $val)
        -stop_restart <$val> (Stop restart file processing at evaluation $val)
        -write_restart [$val] (Write a new DAKOTA restart file $val)
```

- Pre_run, run, post_run options only available to some applications
- What can you do if you failed in the postProcess and do not want to re-run all the simulations?

Running DAKOTA with OpenFOAM

2. Dakota folder structure

Input file options

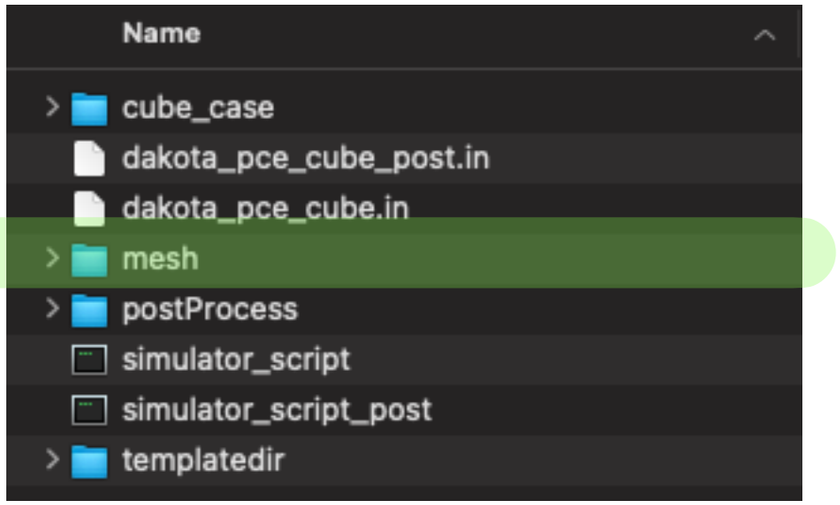


- Most important choices about DAKOTA, we will look deeply into this in a few slides

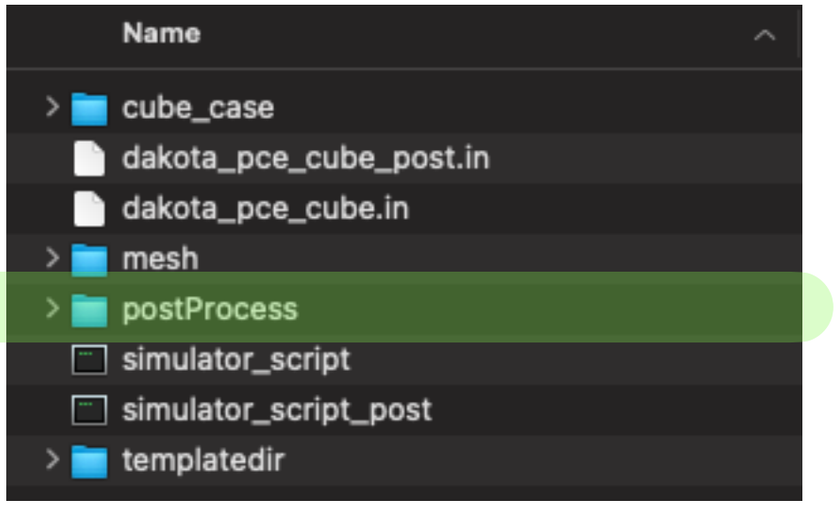
Running DAKOTA with OpenFOAM
2. Dakota folder structure

Mesh folder

- Mesh files tend to be the heaviest when running CFD
- If your mesh will not change, this is great news because you can just link those files and save in storage



Running DAKOTA with OpenFOAM
2. Dakota folder structure



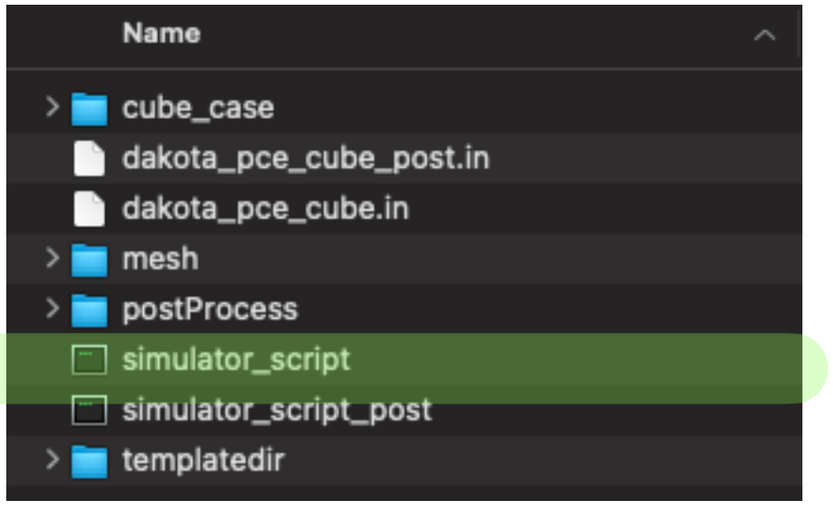
PostProcess folder

- Sometimes the output of OF is not enough and you need to perform some postProcess (unless you implement the postProcessing approach in OF)

- In this case this folder contained two python rutines that will be used to compute velocity magnitude and direction from the velocity surfaces extracted

Running DAKOTA with OpenFOAM
2. Dakota folder structure

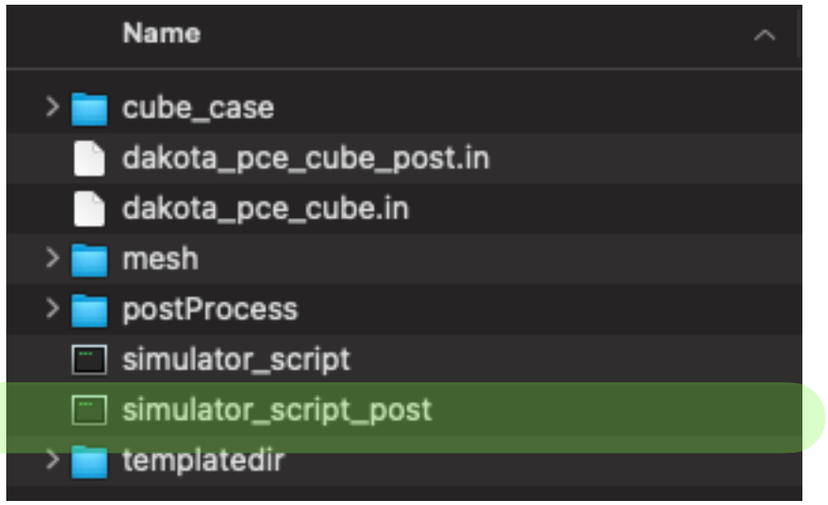
Simulator script



- This is the most important part of the folder, this script in bash includes all the commands dakota will execute

- We will talk further about it in a few slides

Running DAKOTA with OpenFOAM
2. Dakota folder structure

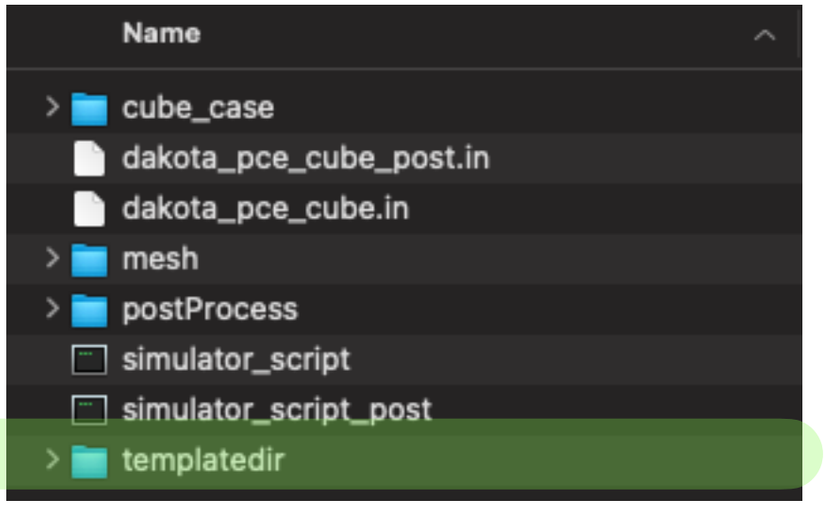


Alternative simulator script

- This is the most important part of the folder, this script in bash includes all the commands dakota will execute

- In this simulator script we just removed some commands so we can jump directly to the post processing step

Running DAKOTA with OpenFOAM
2. Dakota folder structure

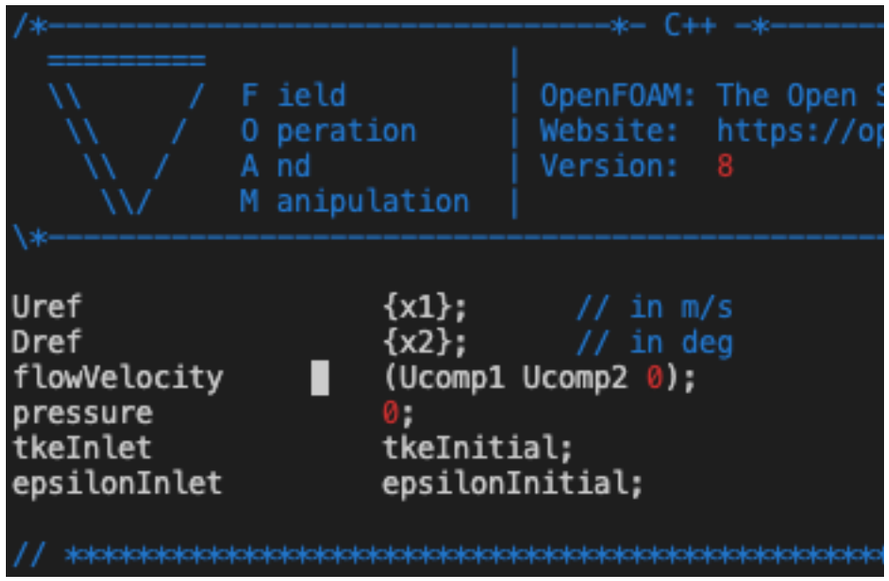


Template folder

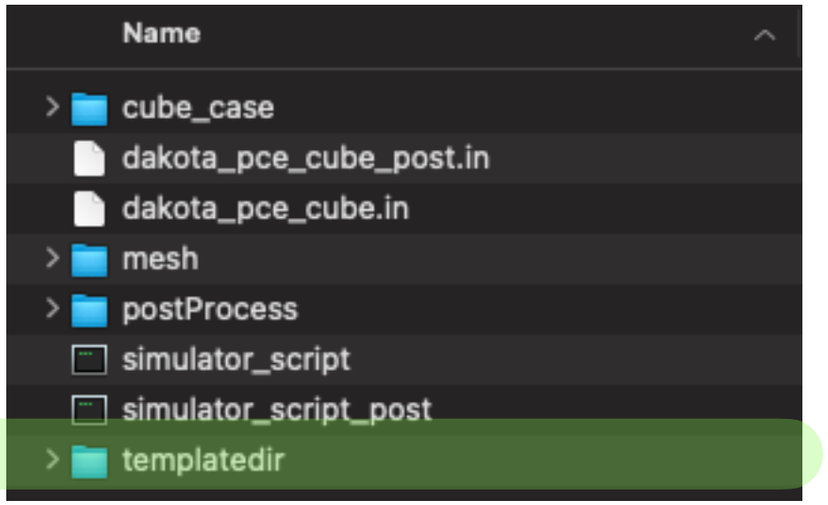
- This folder contains initialConditions.in, so basically the file where the uncertain variables are indicated

- Here the variables are indicated with symbolic names that will be used in the input file

- If we take a look we can see that looks like:



Running DAKOTA with OpenFOAM
2. Dakota folder structure

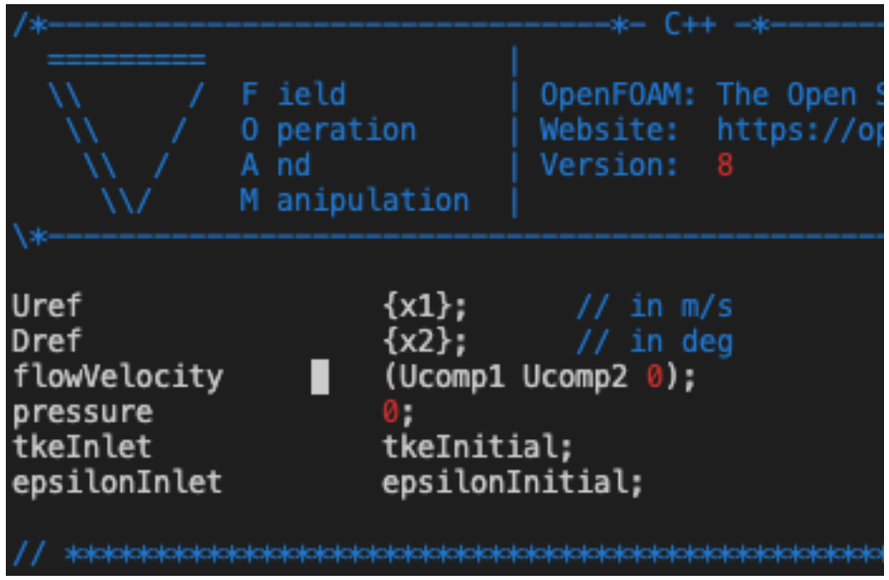


Template folder

- This folder contains initialConditions.in, so basically the file where the uncertain variables are indicated

- Here the variables are indicated with symbolic names that will be used in the input file

- If we take a look we can see that looks like:



Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. **Simulator script line by line**
4. Dakota input file and running options
5. UQ methodology
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM
3. Simulator script

We assign $1 and $2 to infile and outfile as a safety measure, since we found out that otherwise dakota may fail because other applications use $1 and $2 as well

$1 = parameters in
$2 = results.out back to dakota

```bash
#!/bin/bash
# $1 is params.in FROM Dakota
# $2 is results.out returned to Dakota

# Description
#   Run the dakota tool and run the process chain
# ------------------------------------------------------------------------------

# Assign $1 and $2 to variables and reset because they seem to interferewith
# OpenFOAM's postProcess and sourcing
# ------------------------------------------------------------------------------
infile=$1
outfile=$2
set --

# Dakota change the parameters in this file
# ------------------------------------------------------------------------------
dprepro $infile initialConditions.template initialConditions.in

# Run simulation with new parameter set
# ------------------------------------------------------------------------------

    # Get uref, udir, z0 + loop number
    #------------------------------------------------------------------------------
    Uref=$(grep 'Uref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    Dref=$(grep 'Dref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    loopNumber=$(echo $infile | awk -F. '{print $NF}')

    # Transform the scientific notation into a readable format for | bc
    # Here we remove e or E with *10^
    #------------------------------------------------------------------------------
    Uref=`echo $Uref | sed -e 's/[eE]+*/\*10\^/'`
    Dref=`echo $Dref | sed -e 's/[eE]+*/\*10\^/'`

    # Optical stuff
    #------------------------------------------------------------------------------
    Uref=`echo "scale=4; $Uref" | bc`
    Dref=`echo "scale=4; $Dref" | bc`

    # Output the parameters that are used now
    #------------------------------------------------------------------------------
    >&2 echo -e "   ++++ Evaluate sample $loopNumber"
    >&2 echo -e "   |"
    >&2 echo -e "   |--> Uref = $Uref [m/s]"
    >&2 echo -e "   |--> beta = $Dref [deg]"
```

Running DAKOTA with OpenFOAM
3. Simulator script

Dprepro or pyprepro are two template preprocessing tools

- We use them as a black box
- They transform the symbolic variables in initialConditions.template (x1,x2) to their correspondent values in each evaluation in initialConditions.n

```bash
#!/bin/bash
# $1 is params.in FROM Dakota
# $2 is results.out returned to Dakota

# Description
#   Run the dakota tool and run the process chain
# ------------------------------------------------------------------------------

# Assign $1 and $2 to variables and reset because they seem to interferewith
# OpenFOAM's postProcess and sourcing
# ------------------------------------------------------------------------------
infile=$1
outfile=$2
set --

# Dakota change the parameters in this file
# ------------------------------------------------------------------------------
dprepro $infile initialConditions.template initialConditions.in

# Run simulation with new parameter set
# ------------------------------------------------------------------------------

    # Get uref, udir, z0 + loop number
    #------------------------------------------------------------------------------
    Uref=$(grep 'Uref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    Dref=$(grep 'Dref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    loopNumber=$(echo $infile | awk -F. '{print $NF}')

    # Transform the scientific notation into a readable format for | bc
    # Here we remove e or E with *10^
    #------------------------------------------------------------------------------
    Uref=`echo $Uref | sed -e 's/[eE]+*/\*10\^/'`
    Dref=`echo $Dref | sed -e 's/[eE]+*/\*10\^/'`

    # Optical stuff
    #------------------------------------------------------------------------------
    Uref=`echo "scale=4; $Uref" | bc`
    Dref=`echo "scale=4; $Dref" | bc`

    # Output the parameters that are used now
    #------------------------------------------------------------------------------
    >&2 echo -e "   ++++ Evaluate sample $loopNumber"
    >&2 echo -e "   |"
    >&2 echo -e "   |--> Uref = $Uref [m/s]"
    >&2 echo -e "   |--> beta = $Dref [deg]"
```

Running DAKOTA with OpenFOAM
3. Simulator script

Some necessary bash scripting for this specific case

- We get Uref
- We get Dref
- We get the number of evaluation

```bash
#!/bin/bash
# $1 is params.in FROM Dakota
# $2 is results.out returned to Dakota

# Description
#   Run the dakota tool and run the process chain
# ------------------------------------------------------------------------------

# Assign $1 and $2 to variables and reset because they seem to interferewith
# OpenFOAM's postProcess and sourcing
# ------------------------------------------------------------------------------
infile=$1
outfile=$2
set --

# Dakota change the parameters in this file
# ------------------------------------------------------------------------------
dprepro $infile initialConditions.template initialConditions.in

# Run simulation with new parameter set
# ------------------------------------------------------------------------------

    # Get uref, udir, z0 + loop number
    #------------------------------------------------------------------------------
    Uref=$(grep 'Uref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    Dref=$(grep 'Dref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    loopNumber=$(echo $infile | awk -F. '{print $NF}')

    # Transform the scientific notation into a readable format for | bc
    # Here we remove e or E with *10^
    #------------------------------------------------------------------------------
    Uref=`echo $Uref | sed -e 's/[eE]+*/\*10\^/'`
    Dref=`echo $Dref | sed -e 's/[eE]+*/\*10\^/'`

    # Optical stuff
    #------------------------------------------------------------------------------
    Uref=`echo "scale=4; $Uref" | bc`
    Dref=`echo "scale=4; $Dref" | bc`

    # Output the parameters that are used now
    #------------------------------------------------------------------------------
    >&2 echo -e "   ++++ Evaluate sample $loopNumber"
    >&2 echo -e "   |"
    >&2 echo -e "   |--> Uref = $Uref [m/s]"
    >&2 echo -e "   |--> beta = $Dref [deg]"
```

Running DAKOTA with OpenFOAM
3. Simulator script

Some additional bash manipulation to make the numbers look pretty in case they are not, and some additional printing (not strictly necessary)

```bash
#!/bin/bash
# $1 is params.in FROM Dakota
# $2 is results.out returned to Dakota

# Description
#   Run the dakota tool and run the process chain
# ------------------------------------------------------------------------------

# Assign $1 and $2 to variables and reset because they seem to interferewith
# OpenFOAM's postProcess and sourcing
# ------------------------------------------------------------------------------
infile=$1
outfile=$2
set --

# Dakota change the parameters in this file
# ------------------------------------------------------------------------------
dprepro $infile initialConditions.template initialConditions.in

# Run simulation with new parameter set
# ------------------------------------------------------------------------------

    # Get uref, udir, z0 + loop number
    #------------------------------------------------------------------------------
    Uref=$(grep 'Uref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    Dref=$(grep 'Dref' initialConditions.in | awk ' {print substr($2, 0, length($2)-1)}')
    loopNumber=$(echo $infile | awk -F. '{print $NF}')

    # Transform the scientific notation into a readable format for | bc
    # Here we remove e or E with *10^
    #------------------------------------------------------------------------------
    Uref=`echo $Uref | sed -e 's/[eE]+*/\*10\^/'`
    Dref=`echo $Dref | sed -e 's/[eE]+*/\*10\^/'`

    # Optical stuff
    #------------------------------------------------------------------------------
    Uref=`echo "scale=4; $Uref" | bc`
    Dref=`echo "scale=4; $Dref" | bc`

    # Output the parameters that are used now
    #------------------------------------------------------------------------------
    >&2 echo -e "   ++++ Evaluate sample $loopNumber"
    >&2 echo -e "   |"
    >&2 echo -e "   |--> Uref = $Uref [m/s]"
    >&2 echo -e "   |--> beta = $Dref [deg]"
```

Running DAKOTA with OpenFOAM

3. Simulator script

Compute velocity components, necessary to run the OF case

```
# choose the type of case depending on the wind direction
#-----------------------------------------------------------------------------
pi=`echo "4*a(1)" |bc -l`
Ucomp1=`echo "c($Dref*$pi/180)*$Uref" | bc -l`
Ucomp2=`echo "s($Dref*$pi/180)*$Uref" | bc -l`
cp -r ../cube_case/* .
cd constant/polyMesh/
cp -rs ../../../mesh/polyMesh/* .
cd ../../
sed 's/Ucomp1/'${Ucomp1}'/g' initialConditions.in > initialConditions.1
sed 's/Ucomp2/'${Ucomp2}'/g' initialConditions.1 > initialConditions.2
vkconstant=0.41;
Href=10;
z0=0.01;
uStar=`echo "$Uref*$vkconstant/l(($Href+$z0)/$z0)" | bc -l`
tke=`echo "$uStar*$uStar/0.3" | bc -l`
eps=`echo "$uStar*$uStar*$uStar/($vkconstant*($Href+$z0))" | bc -l`
sed 's/tkeInitial/'${tke}'/g' initialConditions.2 > initialConditions.3
sed 's/epsilonInitial/'${eps}'/g' initialConditions.3 > initialConditions.in
cp -r initialConditions.in 0/include/initialConditions
rm initialConditions.3 initialConditions.2 initialConditions.1 initialConditions.in

# Run the simulation
#-----------------------------------------------------------------------------
>&2 echo "    |--> Start simulation"
source /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
touch case$loopNumber.foam

#-----------------------------------------------------------------------------
>&2 echo "    |--> simpleFoam"
decomposePar >& log.decomposePar
wait
mpirun -np 8 simpleFoam -parallel >& log.SF

#-----------------------------------------------------------------------------
>&2 echo "    |--> Reconstruct case"
reconstructParMesh -constant
reconstructPar -latestTime
>&2 echo "    |--> End simulation"
```

Running DAKOTA with OpenFOAM

3. Simulator script

Copy the template case (cube_case) and link the mesh files

```
# choose the type of case depending on the wind direction
#-----------------------------------------------------------------------------
pi=`echo "4*a(1)" |bc -l`
Ucomp1=`echo "c($Dref*$pi/180)*$Uref" | bc -l`
Ucomp2=`echo "s($Dref*$pi/180)*$Uref" | bc -l`
cp -r ../cube_case/* .
cd constant/polyMesh/
cp -rs ../../../mesh/polyMesh/* .
cd ../../
sed 's/Ucomp1/'${Ucomp1}'/g' initialConditions.in > initialConditions.1
sed 's/Ucomp2/'${Ucomp2}'/g' initialConditions.1 > initialConditions.2
vkconstant=0.41;
Href=10;
z0=0.01;
uStar=`echo "$Uref*$vkconstant/l(($Href+$z0)/$z0)" | bc -l`
tke=`echo "$uStar*$uStar/0.3" | bc -l`
eps=`echo "$uStar*$uStar*$uStar/($vkconstant*($Href+$z0))" | bc -l`
sed 's/tkeInitial/'${tke}'/g' initialConditions.2 > initialConditions.3
sed 's/epsilonInitial/'${eps}'/g' initialConditions.3 > initialConditions.in
cp -r initialConditions.in 0/include/initialConditions
rm initialConditions.3 initialConditions.2 initialConditions.1 initialConditions.in

# Run the simulation
#-----------------------------------------------------------------------------
>&2 echo "    |--> Start simulation"
source /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
touch case$loopNumber.foam

#-----------------------------------------------------------------------------
>&2 echo "    |--> simpleFoam"
decomposePar >& log.decomposePar
wait
mpirun -np 8 simpleFoam -parallel >& log.SF

#-----------------------------------------------------------------------------
>&2 echo "    |--> Reconstruct case"
reconstructParMesh -constant
reconstructPar -latestTime
>&2 echo "    |--> End simulation"
```

Running DAKOTA with OpenFOAM
3. Simulator script

Set the velocity components in the initialConditions file that we are building from the initialConditions.in

```
# choose the type of case depending on the wind direction
#-----------------------------------------------------------------------------
pi=`echo "4*a(1)" |bc -l`
Ucomp1=`echo "c($Dref*$pi/180)*$Uref" | bc -l`
Ucomp2=`echo "s($Dref*$pi/180)*$Uref" | bc -l`
cp -r ../cube_case/* .
cd constant/polyMesh/
cp -rs ../../../mesh/polyMesh/* .
cd ../../
sed 's/Ucomp1/'${Ucomp1}'/g' initialConditions.in > initialConditions.1
sed 's/Ucomp2/'${Ucomp2}'/g' initialConditions.1 > initialConditions.2
vkconstant=0.41;
Href=10;
z0=0.01;
uStar=`echo "$Uref*$vkconstant/l(($Href+$z0)/$z0)" | bc -l`
tke=`echo "$uStar*$uStar/0.3" | bc -l`
eps=`echo "$uStar*$uStar*$uStar/($vkconstant*($Href+$z0))" | bc -l`
sed 's/tkeInitial/'${tke}'/g' initialConditions.2 > initialConditions.3
sed 's/epsilonInitial/'${eps}'/g' initialConditions.3 > initialConditions.in
cp -r initialConditions.in 0/include/initialConditions
rm initialConditions.3 initialConditions.2 initialConditions.1 initialConditions.in

# Run the simulation
#-----------------------------------------------------------------------------
>&2 echo "    |--> Start simulation"
source /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
touch case$loopNumber.foam

#-----------------------------------------------------------------------------
>&2 echo "    |--> simpleFoam"
decomposePar >& log.decomposePar
wait
mpirun -np 8 simpleFoam -parallel >& log.SF

#-----------------------------------------------------------------------------
>&2 echo "    |--> Reconstruct case"
reconstructParMesh -constant
reconstructPar -latestTime
>&2 echo "    |--> End simulation"
```

Running DAKOTA with OpenFOAM

3. Simulator script

Compute the initial conditions for turbulence as if this was a typical 3D case where we care about Href, z0, κ, and friction velocity

```bash
# choose the type of case depending on the wind direction
#-----------------------------------------------------------------------------
pi=`echo "4*a(1)" |bc -l`
Ucomp1=`echo "c($Dref*$pi/180)*$Uref" | bc -l`
Ucomp2=`echo "s($Dref*$pi/180)*$Uref" | bc -l`
cp -r ../cube_case/* .
cd constant/polyMesh/
cp -rs ../../../mesh/polyMesh/* .
cd ../../
sed 's/Ucomp1/'${Ucomp1}'/g' initialConditions.in > initialConditions.1
sed 's/Ucomp2/'${Ucomp2}'/g' initialConditions.1 > initialConditions.2
vkconstant=0.41;
Href=10;
z0=0.01;
uStar=`echo "$Uref*$vkconstant/l(($Href+$z0)/$z0)" | bc -l`
tke=`echo "$uStar*$uStar/0.3" | bc -l`
eps=`echo "$uStar*$uStar*$uStar/($vkconstant*($Href+$z0))" | bc -l`
sed 's/tkeInitial/'${tke}'/g' initialConditions.2 > initialConditions.3
sed 's/epsilonInitial/'${eps}'/g' initialConditions.3 > initialConditions.in
cp -r initialConditions.in 0/include/initialConditions
rm initialConditions.3 initialConditions.2 initialConditions.1 initialConditions.in

# Run the simulation
#-----------------------------------------------------------------------------
>&2 echo "    |--> Start simulation"
source /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
touch case$loopNumber.foam

#-----------------------------------------------------------------------------
>&2 echo "    |--> simpleFoam"
decomposePar >& log.decomposePar
wait
mpirun -np 8 simpleFoam -parallel >& log.SF

#-----------------------------------------------------------------------------
>&2 echo "    |--> Reconstruct case"
reconstructParMesh -constant
reconstructPar -latestTime
>&2 echo "    |--> End simulation"
```

Running DAKOTA with OpenFOAM

3. Simulator script

Simulation run:

- Source OF
- Decompose
- Run solver
- Reconstruct

```
# choose the type of case depending on the wind direction
#-----------------------------------------------------------------------------
pi=`echo "4*a(1)" |bc -l`
Ucomp1=`echo "c($Dref*$pi/180)*$Uref" | bc -l`
Ucomp2=`echo "s($Dref*$pi/180)*$Uref" | bc -l`
cp -r ../cube_case/* .
cd constant/polyMesh/
cp -rs ../../../mesh/polyMesh/* .
cd ../../
sed 's/Ucomp1/'${Ucomp1}'/g' initialConditions.in > initialConditions.1
sed 's/Ucomp2/'${Ucomp2}'/g' initialConditions.1 > initialConditions.2
vkconstant=0.41;
Href=10;
z0=0.01;
uStar=`echo "$Uref*$vkconstant/l(($Href+$z0)/$z0)" | bc -l`
tke=`echo "$uStar*$uStar/0.3" | bc -l`
eps=`echo "$uStar*$uStar*$uStar/($vkconstant*($Href+$z0))" | bc -l`
sed 's/tkeInitial/'${tke}'/g' initialConditions.2 > initialConditions.3
sed 's/epsilonInitial/'${eps}'/g' initialConditions.3 > initialConditions.in
cp -r initialConditions.in 0/include/initialConditions
rm initialConditions.3 initialConditions.2 initialConditions.1 initialConditions.in

# Run the simulation
#-----------------------------------------------------------------------------
>&2 echo "    |--> Start simulation"
source /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
touch case$loopNumber.foam

#-----------------------------------------------------------------------------
>&2 echo "    |--> simpleFoam"
decomposePar >& log.decomposePar
wait
mpirun -np 8 simpleFoam -parallel >& log.SF

#-----------------------------------------------------------------------------
>&2 echo "    |--> Reconstruct case"
reconstructParMesh -constant
reconstructPar -latestTime
>&2 echo "    |--> End simulation"

#-----------------------------------------------------------------------------
>&2 echo "    |--> Post-process with python"
python ../postProcess/compute_surface_Umag.py > output_Utheta.txt
python ../postProcess/compute_surface_Udir.py >> output_Utheta.txt
```

Running DAKOTA with OpenFOAM

3. Simulator script

Perform the post-process:

- We use a python routine
- Give back the results to Dakota within a file where both variables are collapsed in the order they are in the input file

```
# Postprocess (with python)
#-----------------------------------------------------------------------
>&2 echo "    |--> Post-process with python"
python ../postProcess/compute_Umag_Udir.py
#-----------------------------------------------------------------------
>&2 echo "    |--> Send output back to dakota"
mv output_Utheta.txt $outfile
echo "Finished simulation in:"
pwd
```

Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. **Dakota input file and running options**
5. UQ methodology
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

Environment

Method

Variables

Interface

Responses

```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
        local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

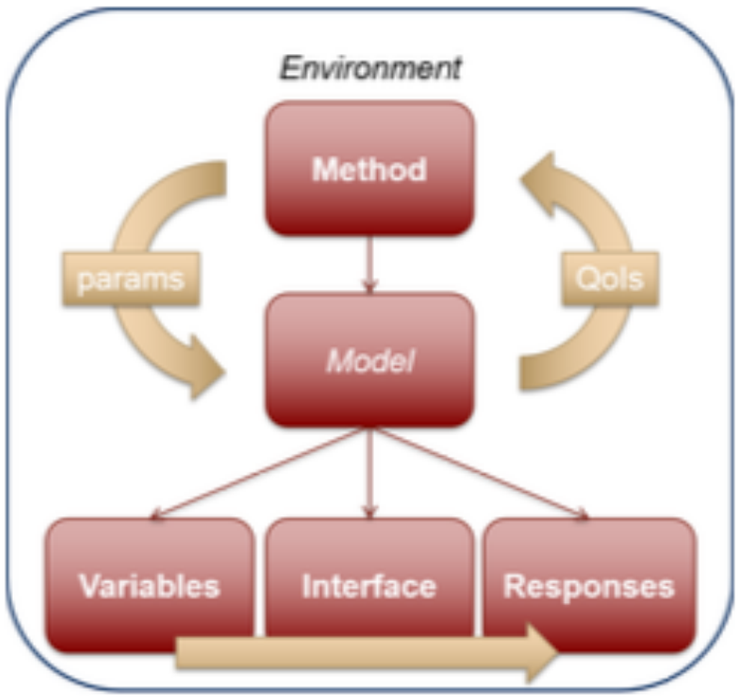


Figure 2.3: Relationship between the six blocks, for a simple study.

```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
          local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

Environment

- Graphical output if any
- Tabular data output

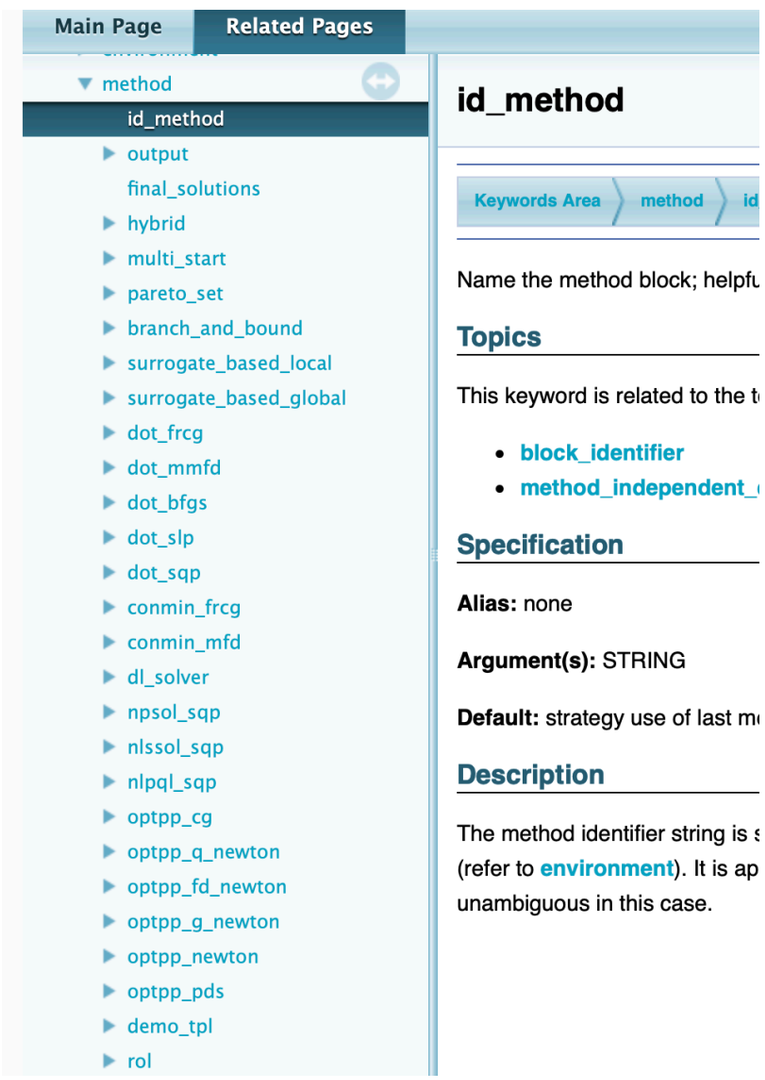
```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
        local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

Method

- Which iterative method will dakota use




```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
          local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
          local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Variables

- Number, type and characteristics od the parameters that will be varied by dakota
- The variables can be design (optimisation/calibration), uncertain (UQ/sensitivity) or state variables (usually fixed)
- They can be continuous or discrete
- Will talk more in a few slides

TU Delft

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

Interface

- Specifies the simulation code that will be used to map the variables into responses
- It also specifies how dakota will pass data to and from that code

```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
        local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

Interface

- Fork or system executions can be used to invoke instances of a simulation code external to dakota (like OF)

- Asynchronous means the use of asynchronous protocols (i.e. background system calls, nonblocking forks, POSIX threads) when evaluations or analysis are invoked
- evaluation_concurrency is the number of evaluations that take place at the same time
- Analysis driver is the main script as we talked before, it drives what happens here
- link_files is where the files with symbolic variables are located

```
interface
  fork
    asynchronous
    evaluation_concurrency = 2
        local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector
```

Running DAKOTA with OpenFOAM

4. Dakota input file and running options

- Type of data that the interface will send back to dakota
- response_functions are used in UQ and sensitivity
- response_functions indicates the number of response functions (every point of our surface)
- Length refers to the length of that end file we obtained were we collapsed velocity magnitude and direction (every point of our surface *2)

Responses

```
environment
    #graphics
    tabular_graphics_data
    tabular_graphics_file = 'table_out.dat'

method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'

interface
  fork
    asynchronous
    evaluation_concurrency = 2
        local_evaluation_scheduling static
    analysis_driver = 'simulator_script'
    parameters_file = 'parameters.in'
    results_file    = 'results.out'
    work_directory directory_tag file_tag
    named 'workdir' file_save  directory_save
    link_files = 'templatedir/*'
    deactivate active_set_vector

responses
  num_response_functions = 26494
  field_responses = 26494
  length = 26494*2
  no_gradients
  no_hessians
```

Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. Dakota input file and running options
5. **UQ methodology**
6. Postprocessing
7. Statistic analysis

Running DAKOTA with OpenFOAM

5. UQ methodology

- Polynomial chaos
  - Quadrature order = 4
  - dimension_preference = 1 1 (indicates the polynomial resolution, same number means evaluations)
  - Samples: number of approximate function evaluations to perform on the emulator model to compute statistics
  - Number_response_levels: all points in surface and the steps from response_levels
  - response_levels: values for which Dakota will compute statistics
  - variance_based_decomp: Sobol indices

```
method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'
```

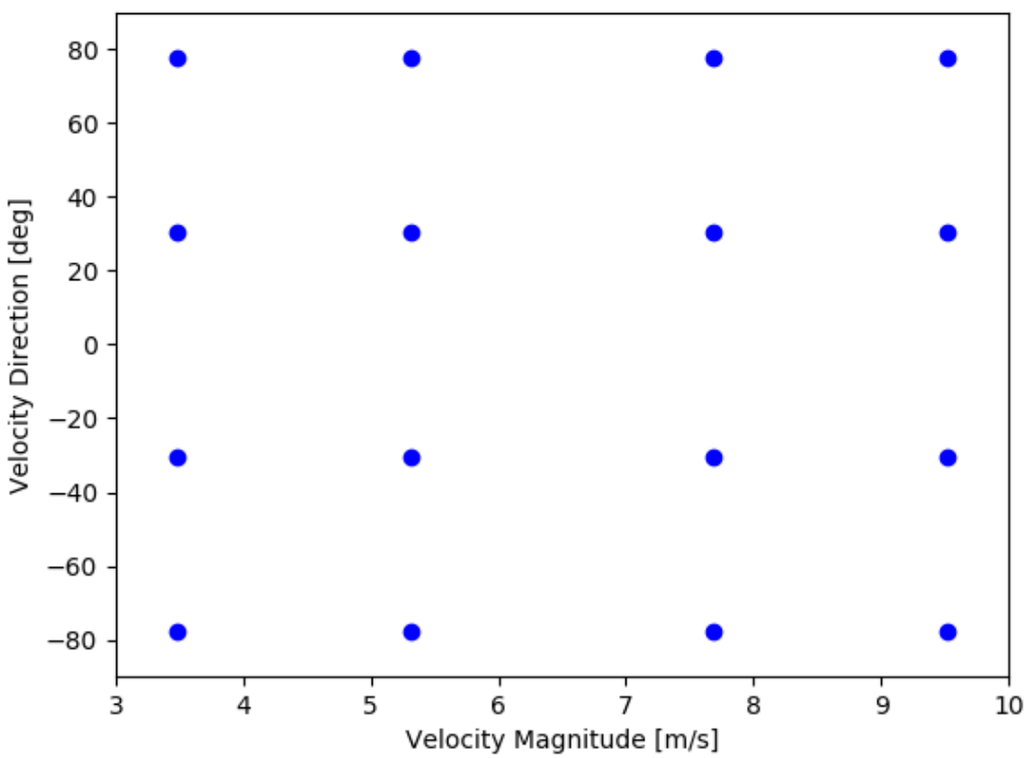
TU Delft

Running DAKOTA with OpenFOAM

5. UQ methodology

- Polynomial chaos
  - Quadrature order = 4
  - dimension_preference = 1 1 (indicates the polynomial resolution, same number means evaluations). It sets the specified order, other dimensions will be reduced in proportion

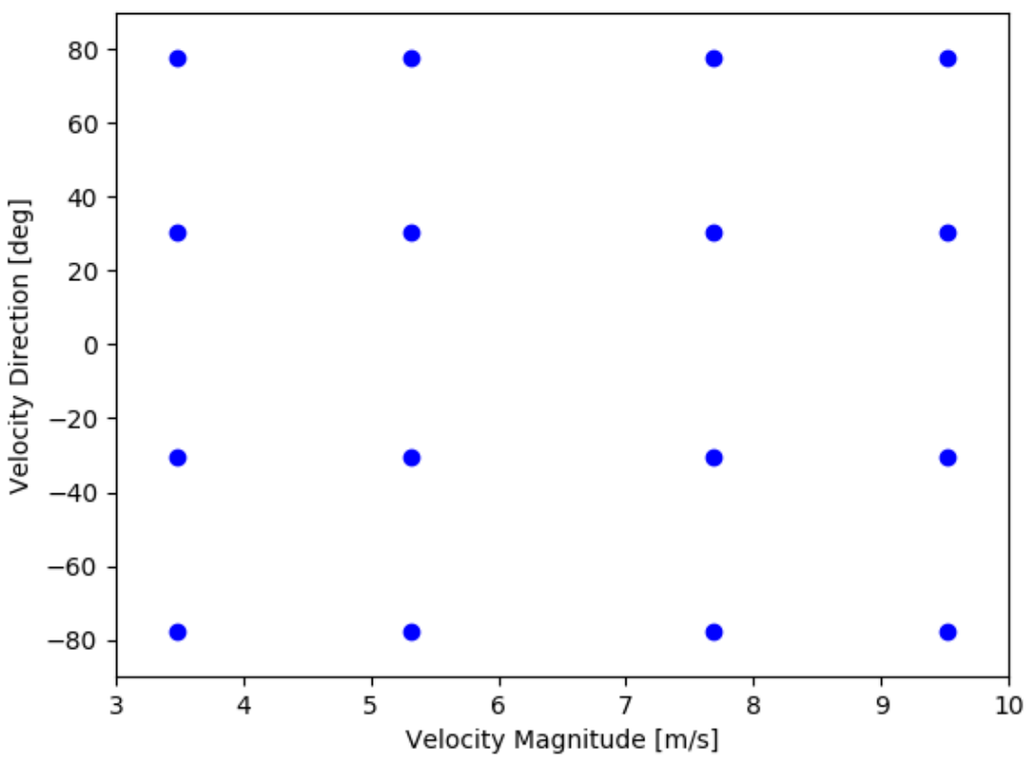
```
method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'
```



Running DAKOTA with OpenFOAM
5. UQ methodology

- Gauss quadratures are the default option
- They become Genz-Keister/ Gauss-Patterson nested rules in case of p_refinement or nested option

```
method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'
```

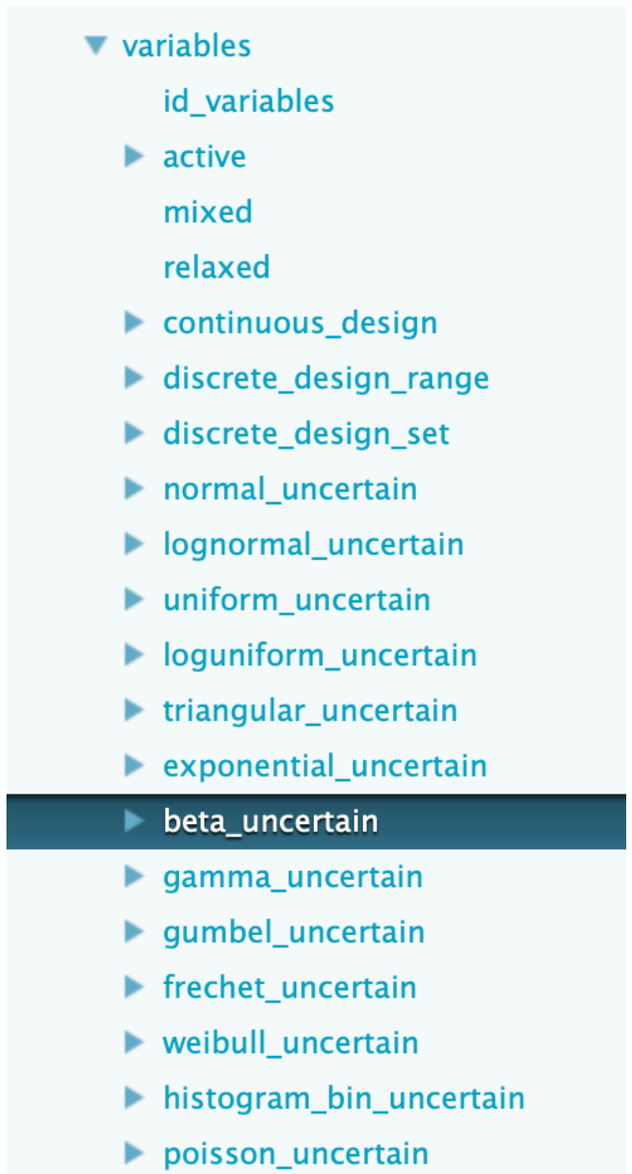


- Expansion order is quadrature_order -1
- Quadrature_order support vector input, which may be necessary for multi fidelity set-up

Running DAKOTA with OpenFOAM

5. UQ methodology

- beta_uncertain
  - Number specifies the number of uncertain variables



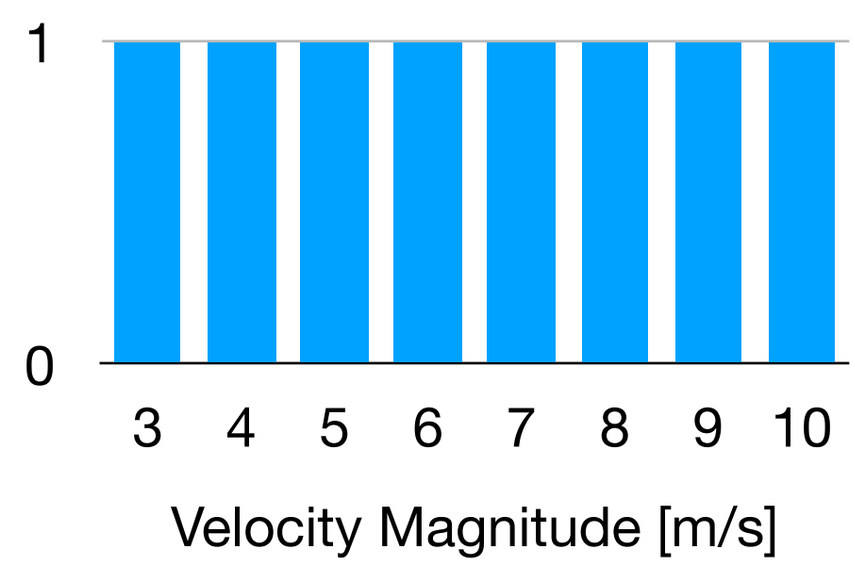

```
9   method
10    polynomial_chaos
11      quadrature_order = 4
12      dimension_preference = 1 1
13      samples = 10000
14      seed = 12347 rng rnum2
15      num_response_levels = 26494*91
16                            26494*8
17      response_levels = 26494*-90:2:90
18                        26494*3:1:10
19      variance_based_decomp
20
21  variables
22    beta_uncertain = 2
23      alphas = 1. 1.
24      betas = 1. 1.
25      lower_bounds = 3 -90
26      upper_bounds = 10 90
27      descriptors  = 'x1' 'x2'
```

Running DAKOTA with OpenFOAM
5. UQ methodology

- beta_uncertain
  - Alphas and betas are the parameters, in this case the variables are uniform
  - Lower and Upper bounds specify the minimum and maximum values for the uncertain parameter (ranges)
  - Descriptors are the symbolic names used

```
method
  polynomial_chaos
    quadrature_order = 4
    dimension_preference = 1 1
    samples = 10000
    seed = 12347 rng rnum2
    num_response_levels = 26494*91
                          26494*8
    response_levels = 26494*-90:2:90
                      26494*3:1:10
    variance_based_decomp

variables
  beta_uncertain = 2
    alphas = 1. 1.
    betas = 1. 1.
    lower_bounds = 3 -90
    upper_bounds = 10 90
    descriptors  = 'x1' 'x2'
```

- Beta distribution is helpful when the actual distribution of an uncertain variable is unknown, but we know very well the bounds, mean and std (common scenario)



Running DAKOTA with OpenFOAM

Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

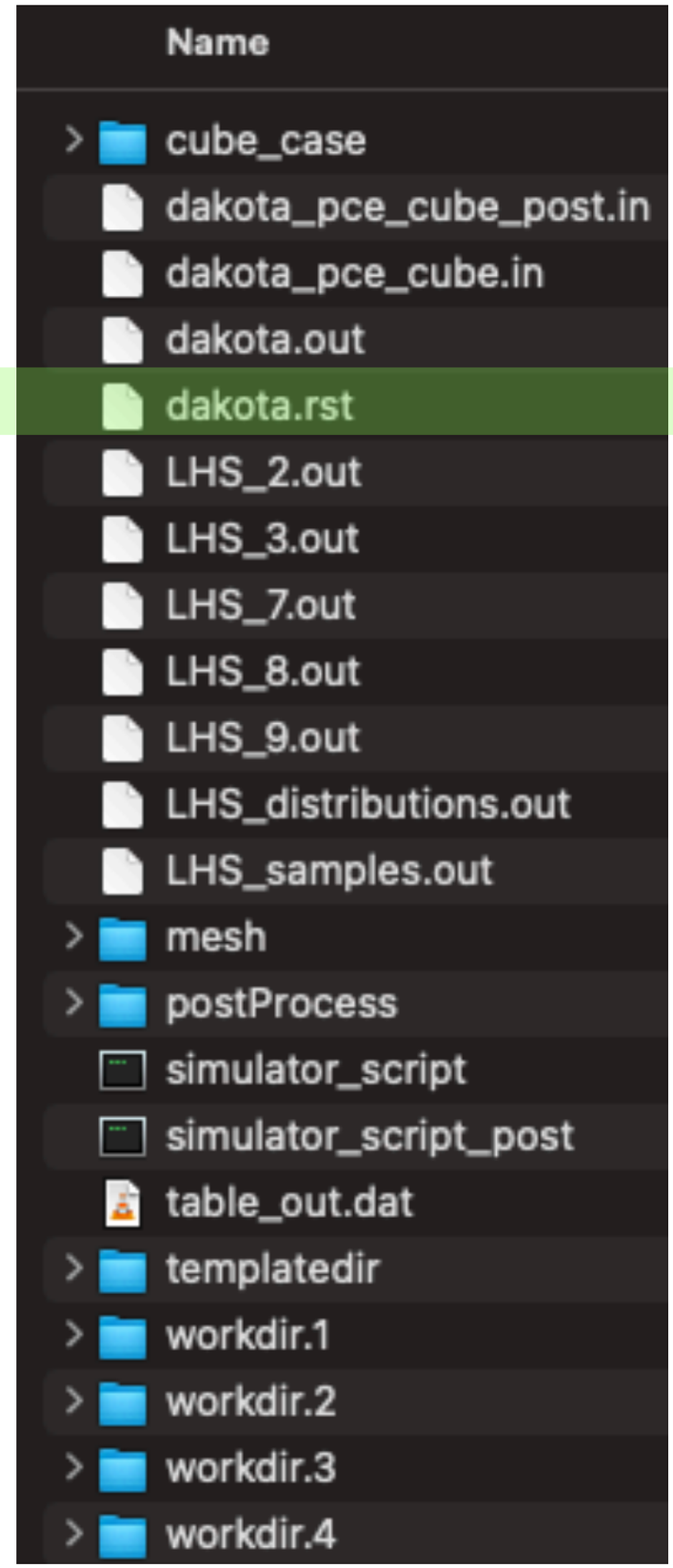
Here it is the outline of the tutorial:

1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. Dakota input file and running options
5. UQ methodology
6. **Postprocessing**
7. Statistic analysis

Running DAKOTA with OpenFOAM
6. Postprocessing

Dakota restart

- It is useful in some cases to restart the dakota run and add for example more samples
- You can find more info about this in incremental sampling in the manual

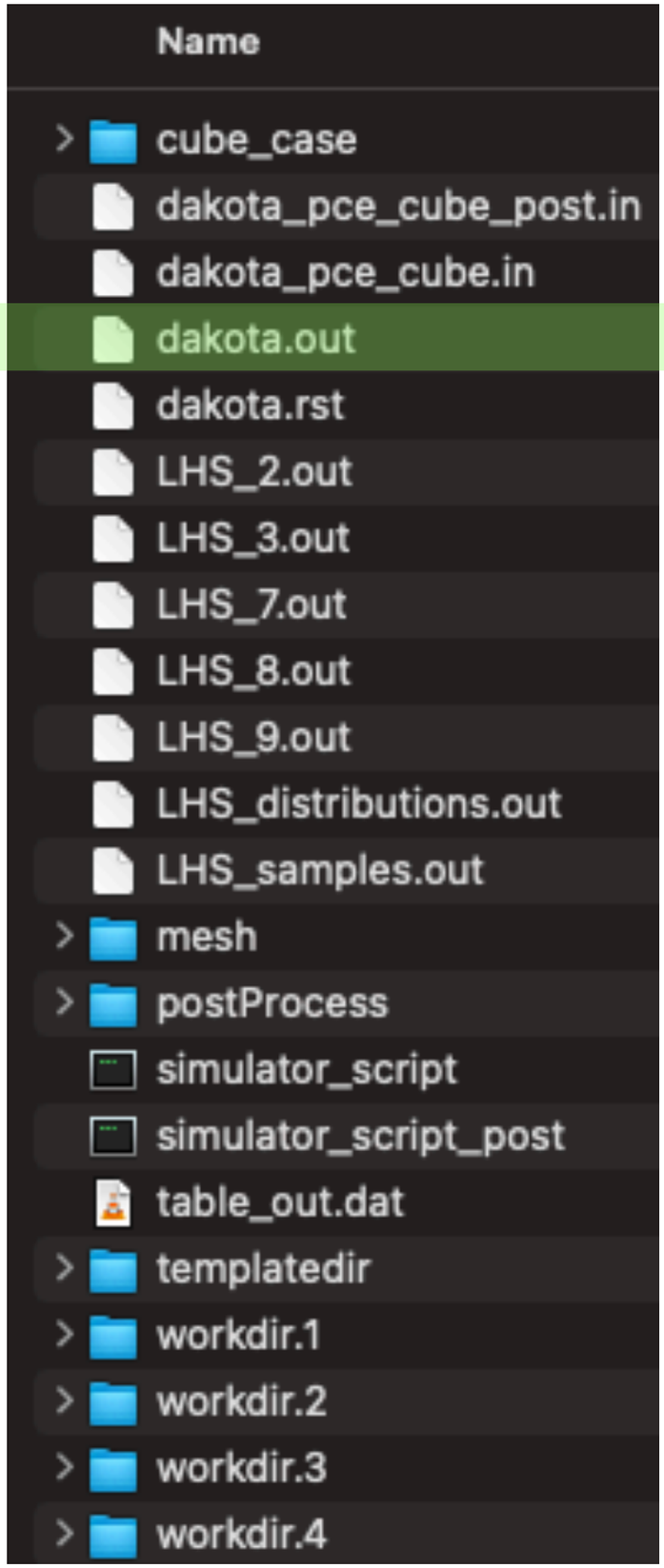


Running DAKOTA with OpenFOAM

6. Postprocessing

Dakota output

- File that is obtained by piping the Dakota command: dakota -i dakota_pce_cube.in -o dakota.out > dakota.out



Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out →

```
Active response data for evaluation 1:
Active set vector = { 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                      0.0000000000e+00 response_fn_1_1
                      0.0000000000e+00 response_fn_2_1
                      0.0000000000e+00 response_fn_3_1
                      0.0000000000e+00 response_fn_4_1
                      0.0000000000e+00 response_fn_5_1
                      0.0000000000e+00 response_fn_6_1
                      0.0000000000e+00 response_fn_7_1
                      0.0000000000e+00 response_fn_8_1
                      0.0000000000e+00 response_fn_9_1
                      0.0000000000e+00 response_fn_10_1
                      0.0000000000e+00 response_fn_11_1
                      0.0000000000e+00 response_fn_12_1
                      0.0000000000e+00 response_fn_13_1
                      0.0000000000e+00 response_fn_14_1
                      0.0000000000e+00 response_fn_15_1
                      0.0000000000e+00 response_fn_16_1
                      0.0000000000e+00 response_fn_17_1
                      0.0000000000e+00 response_fn_18_1
```

These are actually the surfaces that we are extracting as output

↓

Last 3 lines of evaluation 1 response:

```
                      8.0234707389e+00 response_fn_5659_1
                      7.8452939729e+00 response_fn_5660_1
                      7.0984730748e+00 response_fn_5661_1
```

Last numbers in the surface output (U_zNormal.vtk)

```
1.66411 7.50772 0 1.88988 7.79772 3.07945e-21 1.72629 7.65301 1.20248e-19 1.40291
6.95846 6.2418e-22
```

```
In [60]: np.sqrt(1.40291**2+6.95846**2)
Out[60]: 7.098473218918276
```

Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out →

These are the polynomial coefficients, using them we reconstruct the surface for our results (kind of a ROM)

```
NonD lhs Samples = 10000 Seed (user-specified) = 12347
Beginning Approximate Fn Evaluations...
<<<<< Function evaluation summary: 16 total (16 new, 0 duplicate)
------------------------------------------------------------------------
 Coefficients of Polynomial Chaos Expansion for response_fn_49_1:
        coefficient   u1   u2
        -----------  ----  ----
   2.5503669761e+00  Pab0  Pab0
   1.3750551517e+00  Pab1  Pab0
   3.6381081834e-04  Pab2  Pab0
  -1.5523696820e-04  Pab3  Pab0
   1.1933506449e+00  Pab0  Pab1
   6.4203252070e-01  Pab1  Pab1
  -1.5885476230e-04  Pab2  Pab1
   1.3682035087e-04  Pab3  Pab1
  -2.8914557639e+00  Pab0  Pab2
  -1.5584552487e+00  Pab1  Pab2
  -3.5862646297e-04  Pab2  Pab2
   2.3121267394e-04  Pab3  Pab2
  -3.7568069828e+00  Pab0  Pab3
  -2.0229453724e+00  Pab1  Pab3
   1.6260365765e-04  Pab2  Pab3
  -3.5626584184e-04  Pab3  Pab3
```

Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out →

```
NonD lhs Samples = 10000 Seed (user-specified) = 12347
Beginning Approximate Fn Evaluations...
<<<<< Function evaluation summary: 16 total (16 new, 0 duplicate)
-----------------------------------------------------------------------
 Coefficients of Polynomial Chaos Expansion for response_fn_49_1:
        coefficient   u1   u2
        -----------  ----  ----
   2.5503669761e+00  Pab0  Pab0
   1.3750551517e+00  Pab1  Pab0
   3.6381081834e-04  Pab2  Pab0
  -1.5523696820e-04  Pab3  Pab0
   1.1933506449e+00  Pab0  Pab1
   6.4203252070e-01  Pab1  Pab1
  -1.5885476230e-04  Pab2  Pab1
   1.3682035087e-04  Pab3  Pab1
  -2.8914557639e+00  Pab0  Pab2
  -1.5584552487e+00  Pab1  Pab2
  -3.5862646297e-04  Pab2  Pab2
```

Quadrature order was 4 in input (so maximum Pab3)

$$u(\omega) = u_0 \hat{P}_0 + \sum_{i_1=1}^{p} u_{i_1} \hat{P}_1(Q_{i_1}) + \sum_{i_1=1}^{p} \sum_{i_2=1}^{p} u_{i_1} u_{i_2} \hat{P}_2(Q_{i_1} Q_{i_2})$$

Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out →

```
NonD lhs Samples = 10000 Seed (user-specified) = 12347
Beginning Approximate Fn Evaluations...
<<<<< Function evaluation summary: 16 total (16 new, 0 duplicate)
------------------------------------------------------------------------
 Coefficients of Polynomial Chaos Expansion for response_fn_49_1:
        coefficient   u1   u2
        -----------  ----  ----
   2.5503669761e+00  Pab0  Pab0
   1.3750551517e+00  Pab1  Pab0
   3.6381081834e-04  Pab2  Pab0
  -1.5523696820e-04  Pab3  Pab0
   1.1933506449e+00  Pab0  Pab1
   6.4203252070e-01  Pab1  Pab1
  -1.5885476230e-04  Pab2  Pab1
   1.3682035087e-04  Pab3  Pab1
  -2.8914557639e+00  Pab0  Pab2
  -1.5584552487e+00  Pab1  Pab2
  -3.5862646297e-04  Pab2  Pab2
```

Quadrature order was 4 in input (so maximum Pab3)

u1 = [0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]

u2 = [0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]

```
for i in range(0:len(u1)):
    P(i)=coef(i).*legendreP(u1(i),q1)*legendreP(u2(i),q2)
end
output=sum(P);
```

$$u(\omega) = u_0 \hat{P}_0 + \sum_{i_1=1}^{p} u_{i_1} \hat{P}_1(Q_{i_1}) + \sum_{i_1=1}^{p} \sum_{i_2=1}^{p} u_{i_1} u_{i_2} \hat{P}_2(Q_{i_1} Q_{i_2})$$

TU Delft

Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out →

```
NonD lhs Samples = 10000 Seed (user-specified) = 12347
Beginning Approximate Fn Evaluations...
<<<<< Function evaluation summary: 16 total (16 new, 0 duplicate)
------------------------------------------------------------------------
 Coefficients of Polynomial Chaos Expansion for response_fn_49_1:
        coefficient   u1   u2
        -----------  ----  ----
   2.5503669761e+00  Pab0  Pab0
   1.3750551517e+00  Pab1  Pab0
   3.6381081834e-04  Pab2  Pab0
  -1.5523696820e-04  Pab3  Pab0
   1.1933506449e+00  Pab0  Pab1
   6.4203252070e-01  Pab1  Pab1
  -1.5885476230e-04  Pab2  Pab1
   1.3682035087e-04  Pab3  Pab1
  -2.8914557639e+00  Pab0  Pab2
  -1.5584552487e+00  Pab1  Pab2
  -3.5862646297e-04  Pab2  Pab2
```

Quadrature order was 4 in input (so maximum Pab3)

u1 = [0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]

u2 = [0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]

```
for i in range(0:len(u1)):
    P(i)=coef(i).*legendreP(u1(i),q1)*legendreP(u2(i),q2)
end
output=sum(P);
```

→ q1,q2 need to be normalise by minimum and maximum values

$$u(\omega) = u_0 \hat{P}_0 + \sum_{i_1=1}^{p} u_{i_1} \hat{P}_1(Q_{i_1}) + \sum_{i_1=1}^{p} \sum_{i_2=1}^{p} u_{i_1} u_{i_2} \hat{P}_2(Q_{i_1} Q_{i_2})$$

TU Delft

Running DAKOTA with OpenFOAM

6. Postprocessing

dakota.out →

```
Statistics derived analytically from polynomial expansion:

Moment statistics for each response function:
                            Mean     Variance    3rdCentral    4thCentral
response_fn_1_1
  expansion:   0.0000000000e+00  0.0000000000e+00
  integration: 0.0000000000e+00  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
```

```
response_fn_89_1
  expansion:   2.7300277131e+00  2.5499181739e+00
  integration: 2.7300277131e+00  2.5499181739e+00  1.1750715544e+00 -3.1913186439e-02
```

Computed mean, variance, skewness and kurtosis for the PCE expansion

Expansion: integration using the values from PCE (interpolant)
Integration: integration using the values from the original response values

Running DAKOTA with OpenFOAM

6. Postprocessing

dakota.out →

```
Global sensitivity indices for each response function:
response_fn_1_1 Sobol' indices not available due to negligible variance
response_fn_2_1 Sobol' indices not available due to negligible variance
response_fn_3_1 Sobol' indices not available due to negligible variance
response_fn_4_1 Sobol' indices not available due to negligible variance
response_fn_5_1 Sobol' indices not available due to negligible variance
response_fn_6_1 Sobol' indices not available due to negligible variance
response_fn_7_1 Sobol' indices not available due to negligible variance
response_fn_8_1 Sobol' indices not available due to negligible variance
response_fn_9_1 Sobol' indices not available due to negligible variance
```

```
response_fn_29_1 Sobol' indices:
                                  Main            Total
                     1.2587411245e-01  2.0299585129e-01 x1
                     7.9700414871e-01  8.7412588755e-01 x2
                         Interaction
                     7.7121738835e-02 x1 x2
```

Then we get the direct and indirect Sobol indices

Variable 2 has largest effect, while the interaction is the lowest effect

Running DAKOTA with OpenFOAM
6. Postprocessing

dakota.out

```
Cumulative Distribution Function (CDF) for response_fn_5654_1:
     Response Level  Probability Level  Reliability Index  General Rel Index
     --------------  -----------------  -----------------  -----------------
   3.0000000000e+00   1.0000000000e-04
   4.0000000000e+00   1.3190000000e-01
   5.0000000000e+00   2.7220000000e-01
   6.0000000000e+00   4.1280000000e-01
   7.0000000000e+00   5.5100000000e-01
   8.0000000000e+00   6.9370000000e-01
   9.0000000000e+00   8.3210000000e-01
   1.0000000000e+01   9.7130000000e-01
```

Now we can also retrieve the CFD for the each point, and what is their probability for certain velocity magnitude

It is a cumulative pdf, so it tells you which ranges of velocity will have larger probabilities

Running DAKOTA with OpenFOAM

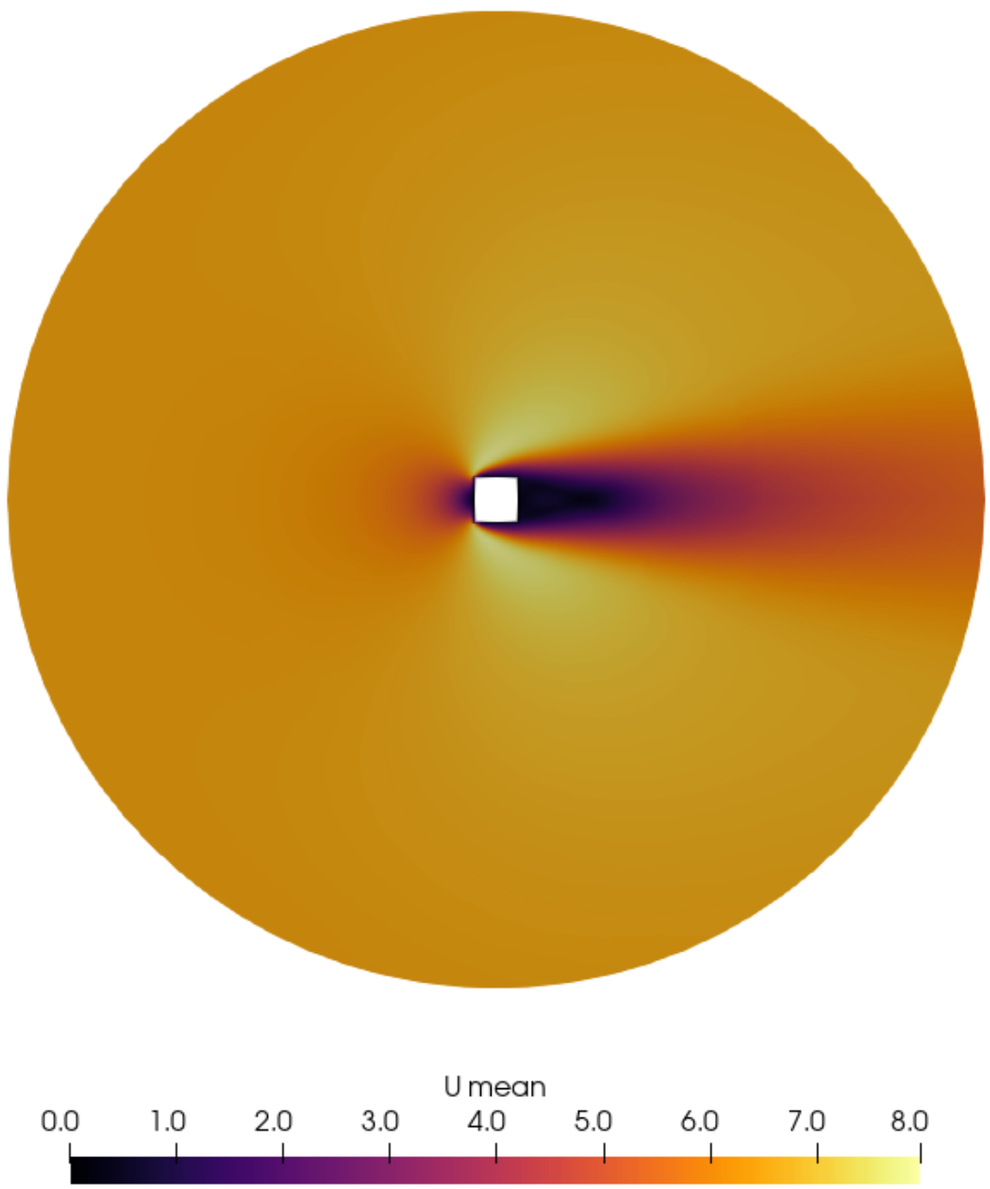
Today we are gonna be looking at the process to run UQ with OpenFOAM following an adapted version between the bash and the more elegant approach

Here it is the outline of the tutorial:

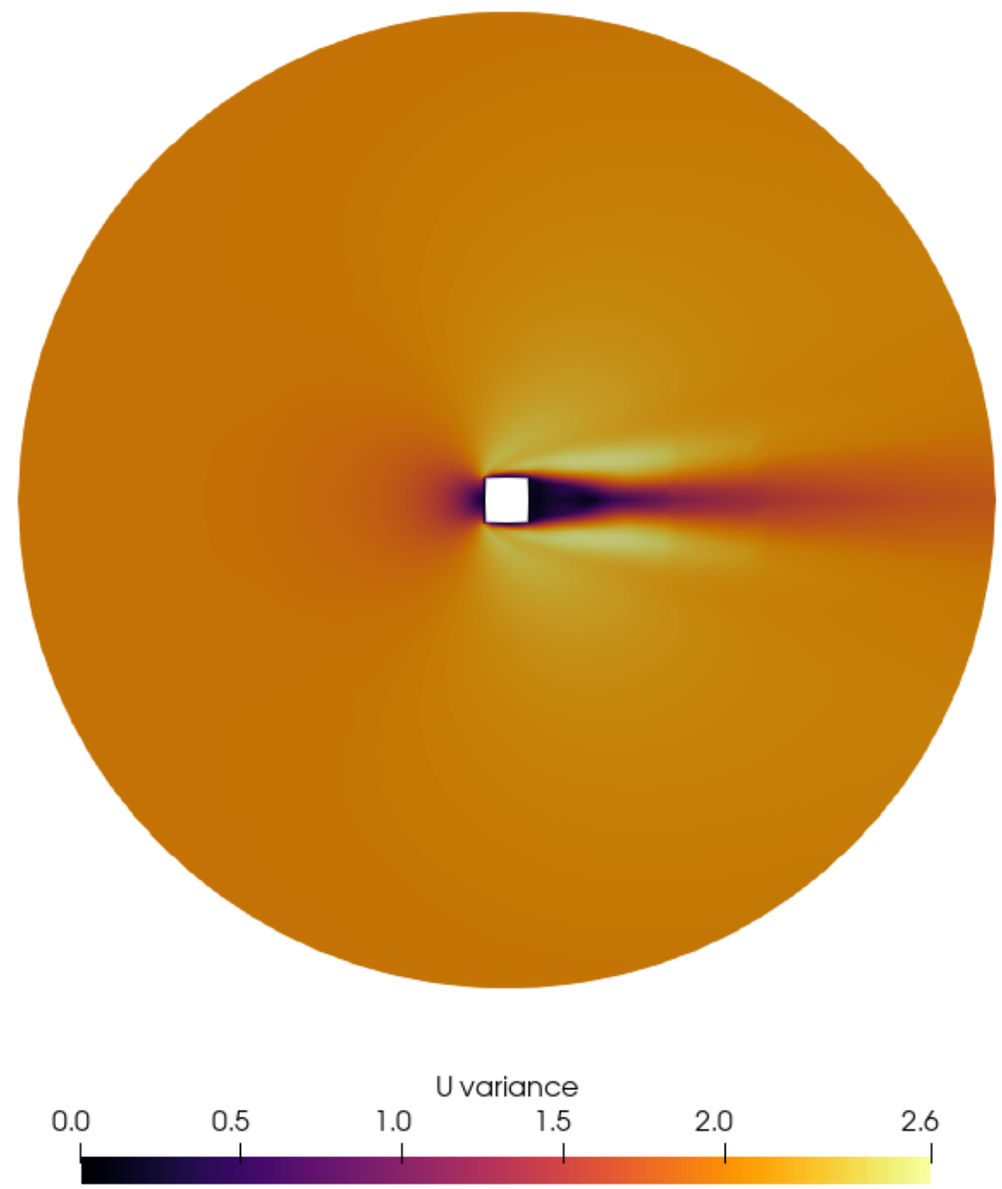
1. Introduce the run test case
2. Look at the folder for Dakota and the different files
3. Simulator script line by line
4. Dakota input file and running options
5. UQ methodology
6. Postprocessing
7. **Statistic analysis**

Running DAKOTA with OpenFOAM
6. Statistic Analysis

Velocity mean
(1st order moment)

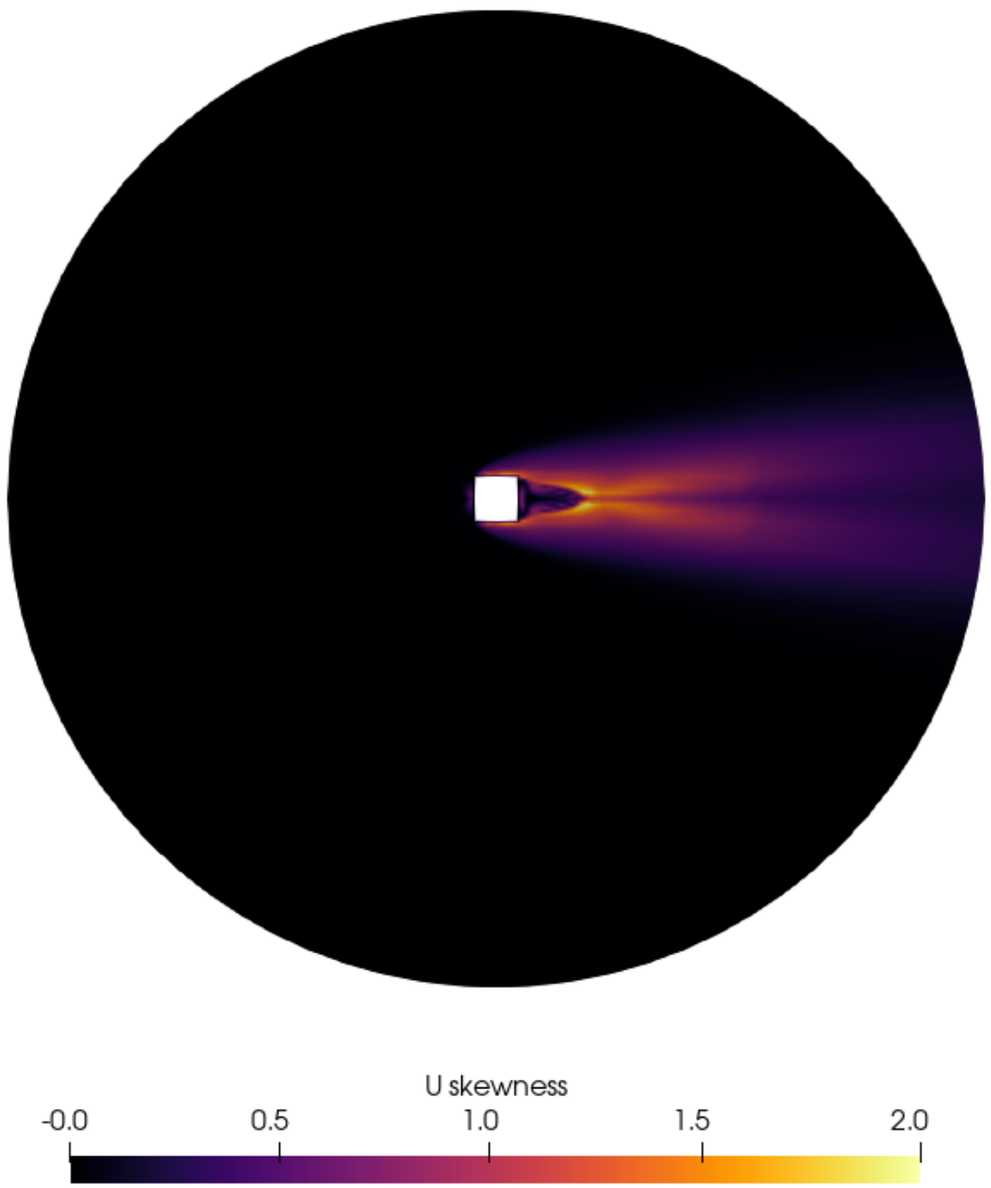


Velocity variance
(2nd order moment)

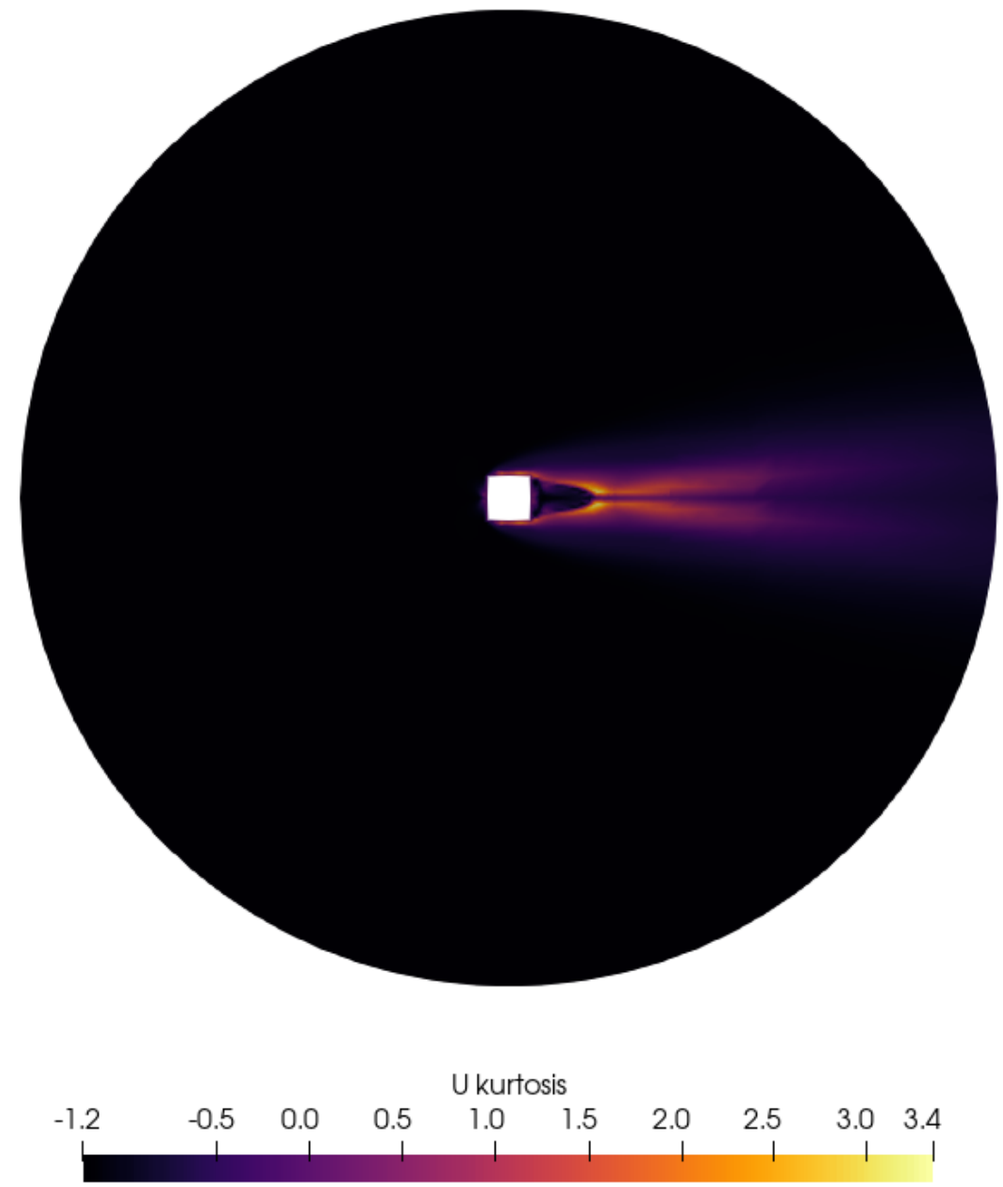


Running DAKOTA with OpenFOAM
6. Statistic Analysis

Velocity skewness
(3rd order moment)

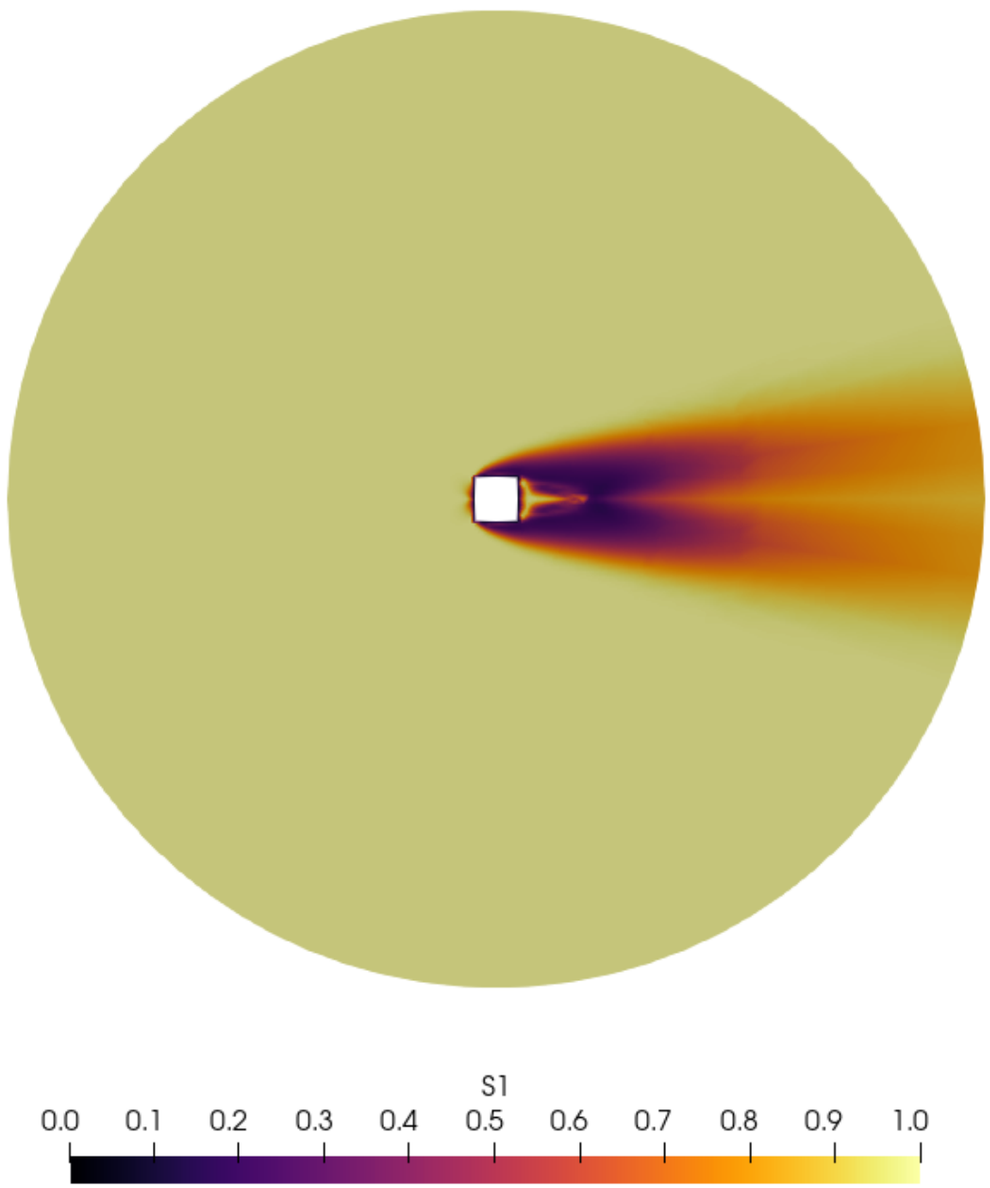


Velocity kurtosis
(4th order moment)

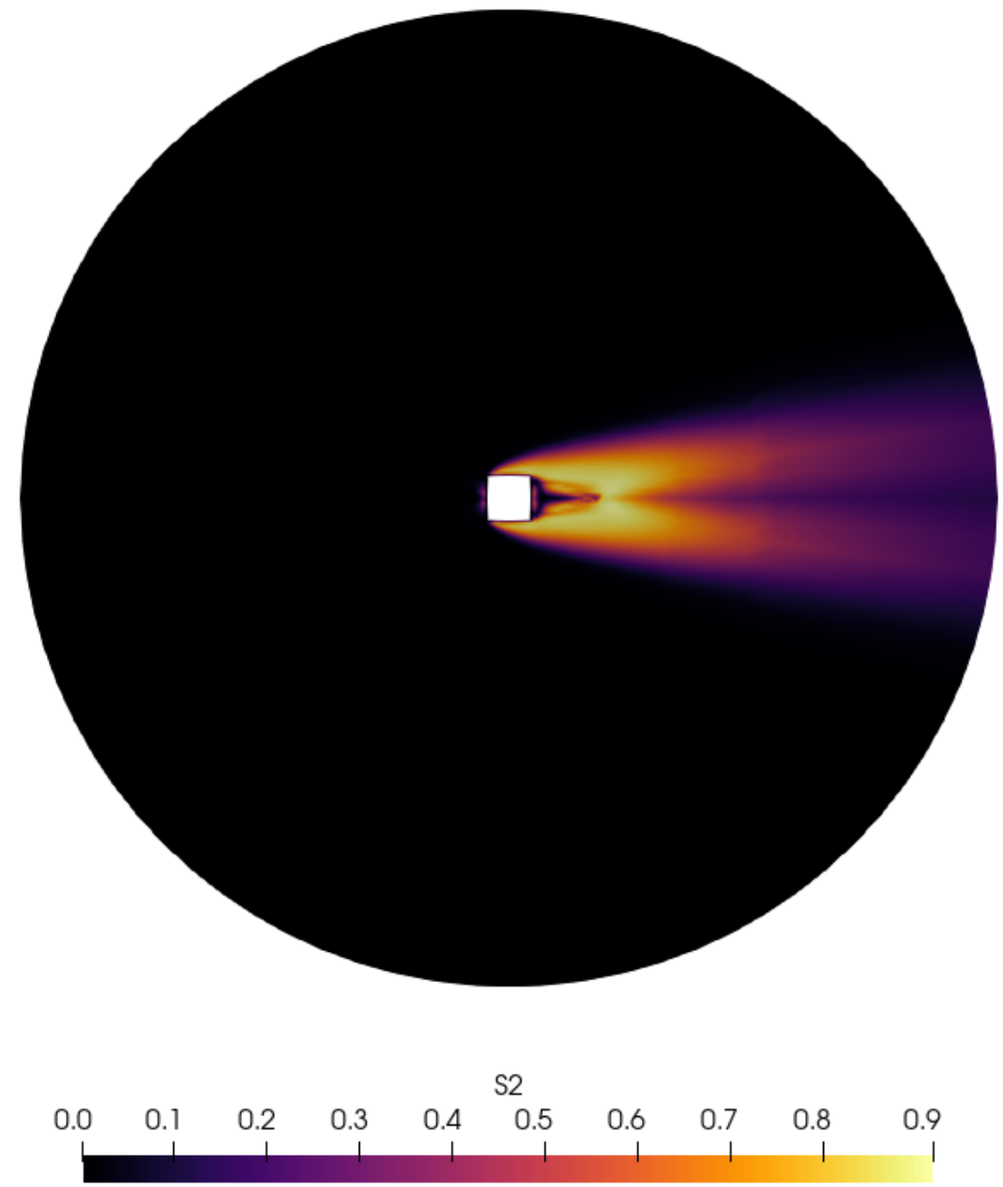


Running DAKOTA with OpenFOAM
6. Statistic Analysis

Total sobol 1
(Velocity magnitude)



Total sobol 2
(Velocity direction)



Thank you!
Any questions?
C.Garcia-Sanchez@tudelft.nl

Clara García-Sánchez

Ivan Pađen

Nadine Hobeika

DAKOTA

Explore and predict with confidence

Sandia National Laboratories

TU Delft

Running DAKOTA with OpenFOAM
5. UQ methodology

The number of beta uncertain variables, the alpha and beta parameters, and the distribution upper and lower bounds are required specifications, while the variable descriptors is an optional specification. The beta distribution can be helpful when the actual distribution of an uncertain variable is unknown, but the user has a good idea of the bounds, the mean, and the standard deviation of the uncertain variable. The density function for the beta distribution is

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{(x-L)^{\alpha-1}(U-x)^{\beta-1}}{(U-L)^{\alpha+\beta-1}}$$

where $\Gamma(\alpha)$ is the gamma function and $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ is the beta function. To calculate the mean and standard deviation from the alpha, beta, upper bound, and lower bound parameters of the beta distribution, the following expressions may be used.

$$\mu = L + \frac{\alpha}{\alpha+\beta}(U-L)$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}(U-L)^2$$

Solving these for α and β gives:

$$\alpha = (\mu - L)\frac{(\mu-L)(U-\mu)-\sigma^2}{\sigma^2(U-L)}$$

$$\beta = (U - \mu)\frac{(\mu-L)(U-\mu)-\sigma^2}{\sigma^2(U-L)}$$

Note that the uniform distribution is a special case of this distribution for parameters $\alpha = \beta = 1$.

TU Delft