Lessons learned on dealing with a 16 TB dataset

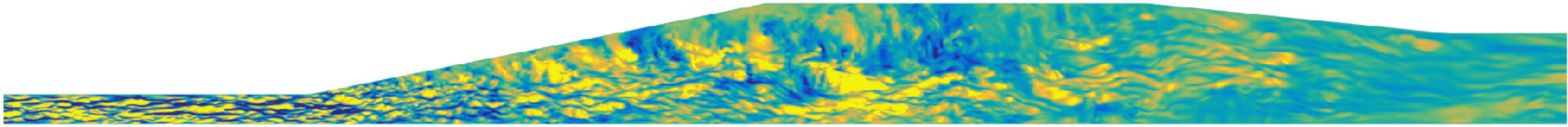
Outcomes of the HiFiTurb project

Arnau Miró & Oriol Lehmkuhl (BSC, CASE)

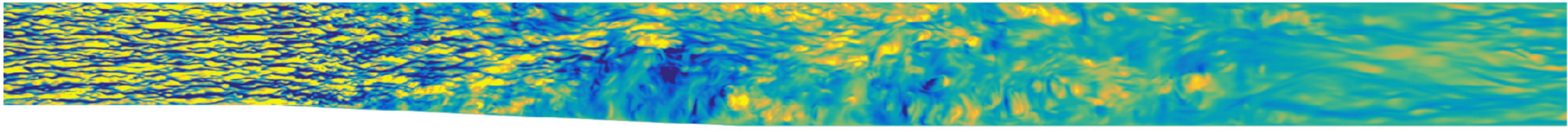
Issue #1: it’s BIG

Lesson #1

A bit of Colorful Fluid Dynamics…



Velocity fluctuations $u' = u - \overline{u}$



Lesson #1

Some facts

- Initial mesh of 31.4 M nodal points (~31 M elements)
- Computed with 1 mesh divisor to reach 270 M nodal points (~250 M elements)
- Total degrees of freedom (DoF) 1000 M
- Run in MareNostrum IV under a RES project:
  - requested a total of 5M hours
  - using 84 computing nodes (i.e., 4023 CPUs)
- The whole computed database is 16 TB on disk
- Post-processed datasets are: 261 GB for the statistics and 1.2 TB for the instantaneous

And this is just the small one…

Lesson #1

Data to extract - turbulent statistics

Quantity		#
Level 1 - averaged Navier-Stokes equations		
Averaged pressure	$\overline{p}$	1
Averaged velocity	$\overline{u}_i$	3
Averaged temperature	$\overline{T}$	1
Averaged shear stress	$\overline{\tau_{ij}}$	6
Averaged heat flux	$\overline{q_i}$	3
Reynolds stress $\mathcal{R}^*_{ij}$	$\rho\overline{u'_i u'_j}$	6
Turbulent heat flux $\mathcal{Q}_i$	$\rho\overline{e' u'_i}$	3
Cumulative total		**23**
Level 1 - additional quantities		
Pressure autocorrelation	$\overline{p'p'}$	1
Temperature autocorrelation	$\overline{T'T'}$	1
Taylor microscale η_T	$\approx \sqrt{5\frac{\mu}{\rho}\frac{\mathcal{R}^*_{ii}}{\overline{\tau'_{ij}\mathcal{S}'_{ij}}}}$	1
Kolmogorov length scale η_K	$\approx \sqrt[4]{\frac{\mu^3/\rho^2}{\overline{\tau'_{ij}\mathcal{S}'_{ij}}}}$	1
Kolmogorov time scale τ_K	$\approx \sqrt{\frac{\mu}{\overline{\tau'_{ij}\mathcal{S}'_{ij}}}}$	1
Cumulative total		**28**

Level 2 - Reynolds stress equations budget terms		
Convection C^*_{ij}	$\left(\mathcal{R}^*_{ij}\overline{u}_k\right)_{,k}$	6
Production P^*_{ij}	$-\left(\mathcal{R}^*_{ik}\overline{u}_{j,k} + \mathcal{R}^*_{jk}\overline{u}_{i,k}\right)$	6
Turbulent diffusion D^{1*}_{ij}	$-\rho\left(\overline{u'_i u'_j u'_k}\right)_{,k}$	6
Turbulent diffusion D^{2*}_{ij}	$-\left(\overline{p'\left(u'_i\delta_{jk} + u'_j\delta_{ik}\right)}\right)_{,k}$	6
Molecular diffusion D^{3*}_{ij}	$\mu\left(\overline{u'_i u'_j}\right)_{,kk}$	6
Pressure strain Φ^*_{ij}	$\overline{p'\left(u'_{i,j} + u'_{j,i}\right)}$	6
Dissipation ϵ^*_{ij}	$2\mu\overline{u'_{i,k}u'_{j,k}}$	6
Cumulative total		**70**
Level 2 - Reynolds stress equations - separate terms		
Triple velocity correlation	$\rho\overline{u'_i u'_j u'_k}$	10
Pressure velocity correlation	$\overline{p'u'_i}$	3
Cumulative total		**83**

Lesson #1

Data to extract - instantaneous data

- Velocity, pressure, gradient of velocity and gradient of pressure
- A total of 16 scalar fields
- That is, 14 GB per time instant
- And a total of 25 instants from the original 1881 instants of the database

Lesson #1

Data to extract - recap

- A total of 83 scalar fields
- Need to perform (in parallel):
  - basic algebraic operations between fields
  - temporal averaging
  - derivatives (gradient)
  - second derivatives (divergence)
- A smart and efficient way to store the data back to disk (in parallel)

Lesson #1: need of an efficient parallel post-processing tool

Issue #2: It is not easy to deal with

Lesson #2

Python to the rescue!

- Developed pyAlya, a Python module for parallel post-processing in Alya
- With pyAlya we can do (in parallel):
  - Read and write native Alya files
  - Compute derivatives and integrals using FEM
  - Perform spatial and temporal averaging
  - Compute turbulent statistics
  - Select domain regions
  - Interpolate results
- Will be the link with other machine learning tools

Lesson #2

Performance results - Python

```
Start: 11:39:29 CET
cr_info (mpi size: 96):
name gradient3D           n     85595 tmin 1.041435e+00 tmax 7.841381e+00 tavg 2.822403e+00 tsum 2.415836e+05
name AlyaMPIO_read        n    341376 tmin 7.532835e-03 tmax 1.841819e+01 tavg 9.532054e-02 tsum 3.254015e+04
name field read           n    170304 tmin 5.460477e-02 tmax 1.846654e+01 tavg 1.908300e-01 tsum 3.249911e+04
name field write          n        96 tmin 2.752678e+01 tmax 3.907820e+01 tavg 2.768814e+01 tsum 2.658062e+03
name AlyaMPIO_write       n      5184 tmin 2.449641e-01 tmax 1.105659e+00 tavg 5.092897e-01 tsum 2.640158e+03
name D3Budget             n        96 tmin 3.600121e-05 tmax 1.227981e+01 tavg 1.212892e+01 tsum 1.164376e+03
name divergence3D         n       285 tmin 1.394428e+00 tmax 4.466482e+00 tavg 3.419647e+00 tsum 9.745994e+02
name D1Budget             n        96 tmin 5.006790e-05 tmax 4.466589e+00 tavg 4.387382e+00 tsum 4.211887e+02
name D2Budget             n        96 tmin 3.530979e-04 tmax 2.747691e+00 tavg 2.718429e+00 tsum 2.609692e+02
name addS1                n   1095445 tmin 1.287460e-05 tmax 2.444792e-02 tavg 1.173372e-04 tsum 1.285364e+02
name tripleCorr           n     84265 tmin 7.228851e-04 tmax 2.089000e-02 tavg 8.183193e-04 tsum 6.895568e+01
name mesh init            n        96 tmin 7.419586e-04 tmax 7.147901e-01 tavg 7.032661e-01 tsum 6.751355e+01
name dissiBudget          n     84265 tmin 6.911755e-04 tmax 1.698899e-02 tavg 7.787160e-04 tsum 6.561851e+01
name mesh elemtList       n        96 tmin 3.678799e-04 tmax 7.000570e-01 tavg 6.537934e-01 tsum 6.276417e+01
name mesh read            n        96 tmin 4.495249e-01 tmax 6.731429e-01 tavg 5.613563e-01 tsum 5.389020e+01
End: 12:35:03 CET
```

Times in seconds

Lesson #2

Cython to the rescue!

- Cython is a bridge between C/C++ and Python
- Code is very similar to python with type definition and some standard C practices (pointers, memory allocation, data structures, etc.)
- Can be interfaced with C/C++/Fortran native libraries
- Can be interfaced with native python code
- The end product are compiled python modules

Lesson #2

Performance results - Cython

```
Start: 13:03:59 CET
cr_info (mpi size: 96):
name AlyaMPIO_read          n    341376 tmin 5.655050e-03 tmax 1.326595e+01 tavg 5.786291e-02 tsum 1.975301e+04
name field read             n    170304 tmin 1.777411e-02 tmax 1.139990e+00 tavg 1.079927e-01 tsum 1.839159e+04
name gradient3D             n     85595 tmin 4.394007e-02 tmax 1.087670e-01 tavg 5.119490e-02 tsum 4.382027e+03
name field write            n        96 tmin 2.203951e+01 tmax 2.253378e+01 tavg 2.204834e+01 tsum 2.116641e+03
name AlyaMPIO_write         n      5184 tmin 1.658051e-01 tmax 6.871901e-01 tavg 4.079710e-01 tsum 2.114921e+03
name mesh read              n        96 tmin 1.374218e+01 tmax 1.488522e+01 tavg 1.431397e+01 tsum 1.374141e+03
name addS1                  n   1095445 tmin 1.287460e-05 tmax 1.634908e-02 tavg 1.081880e-04 tsum 1.185140e+02
name tripleCorr             n     84265 tmin 7.522106e-04 tmax 5.433083e-03 tavg 8.502793e-04 tsum 7.164879e+01
name mesh init              n        96 tmin 6.160736e-04 tmax 3.018630e-01 tavg 2.955244e-01 tsum 2.837034e+01
name dissiBudget            n     84265 tmin 2.529621e-04 tmax 1.633191e-02 tavg 2.990905e-04 tsum 2.520286e+01
name mesh elemtList         n        96 tmin 6.604195e-05 tmax 2.908800e-01 tavg 2.583676e-01 tsum 2.480329e+01
name divergence3D           n       285 tmin 4.394698e-02 tmax 6.934118e-02 tavg 5.665245e-02 tsum 1.614595e+01
name D3Budget               n        96 tmin 3.600121e-05 tmax 1.257901e-01 tavg 1.183716e-01 tsum 1.136367e+01
name D1Budget               n        96 tmin 5.507469e-05 tmax 6.978798e-02 tavg 6.022587e-02 tsum 5.781683e+00
name D2Budget               n        96 tmin 6.794930e-05 tmax 6.109285e-02 tavg 5.185532e-02 tsum 4.978110e+00
End: 13:09:09 CET 2022
```

Lesson #2: to our experience, compiled code is still necessary

Issue #3: Reading the data can be problematic

Dear Arnau,

We have detected that the job 14639359, which was submitted this morning, is making an abusive use of the GPFS filesystem. It is taking much of the bandwidth available, directly affecting other executions running in the system (specially during the first 1:30h).

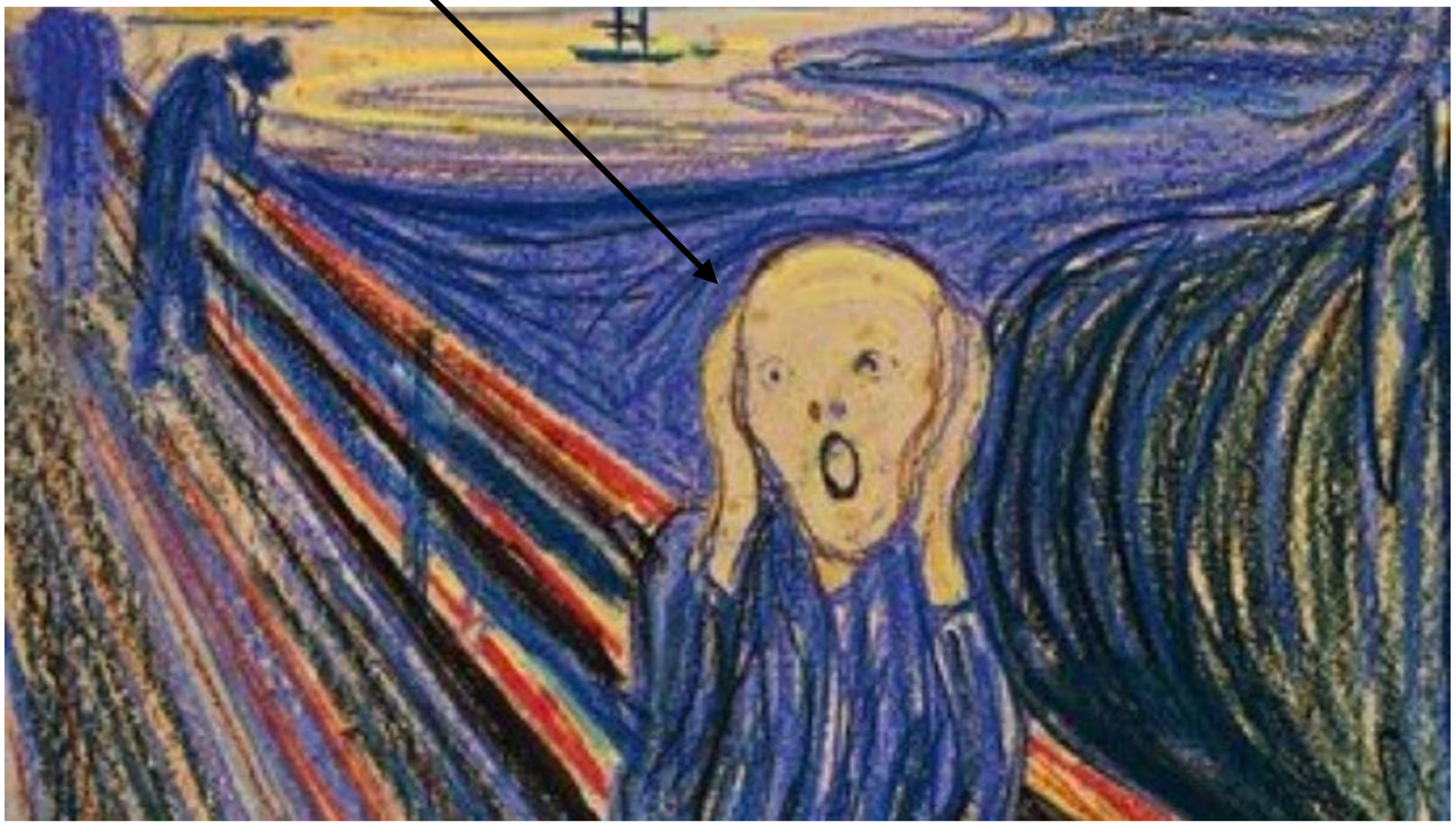
We kindly ask you to please, take al look at this part of the job, and try to minimize the reading overhead. It looks like you are accessing a folder with around 14TB of data, which is more than likely the cause of the issue.

Feel free to contact us to further investigate this problem. If we see again this kind of job, we will be forced to kill it.

Kind regards,

BSC Support Team

this was me…



Ups!

Lesson #3

First naive implementation of the reader

- Was not using MPI-IO
- Each process was opening and closing the file
- Each process was doing file seeks and reads
- The amount of data read was very small, i.e., too many reads of small fragments.

That resulted in 120GB/s of bandwith stress into the GPFS system!!!

Lesson #3

Solution to the problem

- Used MPIO
- Avoided file seeks
- One single read per processor
- Only one processor reads the header and scatters that to the rest
- The amount of data read was not small

That resulted in 45GB/s of bandwith, still high…

Hola, Arnau

acabo de matar el job. Te explico. El throughput de lectura de GPFS estaba muy alto, aunque no al límite como el otro día. Estos valores son acumulados por lo que no tenemos el dato por nodo y hemos de lanzar un proceso paralelo que saque información de los nodos en una lista de sospechosos. En esta ocasión no eráis los únicos y os hemos matado el job para comprobar cómo bajaba el throughput de GPFS y tener un valor más aproximado del uso real de vuestro job. En este caso la carga ha bajado de 90GB/s a 45GB/s, con lo cual podemos determinar que vuestro job estaba consumiendo esa diferencia de 45GB/s. Una mejora muy notable con respecto a los 120GB/s que vimos el otro día. Aun así esa una carga significativa por lo que si coincidieran varios jobs similares el problema aparecería de nuevo.

Ahora os pediremos por favor que volváis a lanzarlo para que veamos una ejecución completa y durante cuánto tiempo se mantiene la carga.

Un saludo,

Hola, Arnau

me alegro haya acabado bien por fin. Al principio de la ejecución hubo bastante carga de GPFS pero no fue responsable tu job.

Como te expliqué ayer no tenemos valores absolutos por job para valorar el nivel de mejora en I/O pero sí que estimamos un **uso menor del que te comenté ayer (<20GB/s).** Por tanto la mejora conseguida es bastante considerable y en condiciones normales no supone ningún problema para GPFS.

Gracias por el esfuerzo de adaptar el código y si surge cualquier problema futuro no dudéis en contactarnos de nuevo.

Un saludo y ¡buen fin de semana!

Lesson #3: having a smart and efficient access to the dataset is important

Issue #4: Even writing the data as well

Hola,

Estic intentant escriure un arxiu usant la versió python paralela (mpio) del hdf5 (per la tool de postprocés en python que estem desenvolupant). L'elecció de hdf5 ve motivada perque l'altra part del consorci (CINECA) ja l'han començat a usar per treure el postprocés.

Em trobo que, pel cas test que tinc, reduir a 1 processador, ordenar i escriure em resulta molt més rápid que escriure el fitxer en paral·lel. En concret, escriure el fitxer en paral·lel amb només 1 array em triga al voltant d'un minut. N'haig d'escriure com uns 30 o més i a més poder-lo usar en malles grans on no sigui viable fer la reducció.

Et passo el summary d'un petit exemple perque ho vegis:

Reduction + serial run:

Thu Nov 19 12:39:55 CET 2020
Thu Nov 19 12:40:05 CET 2020

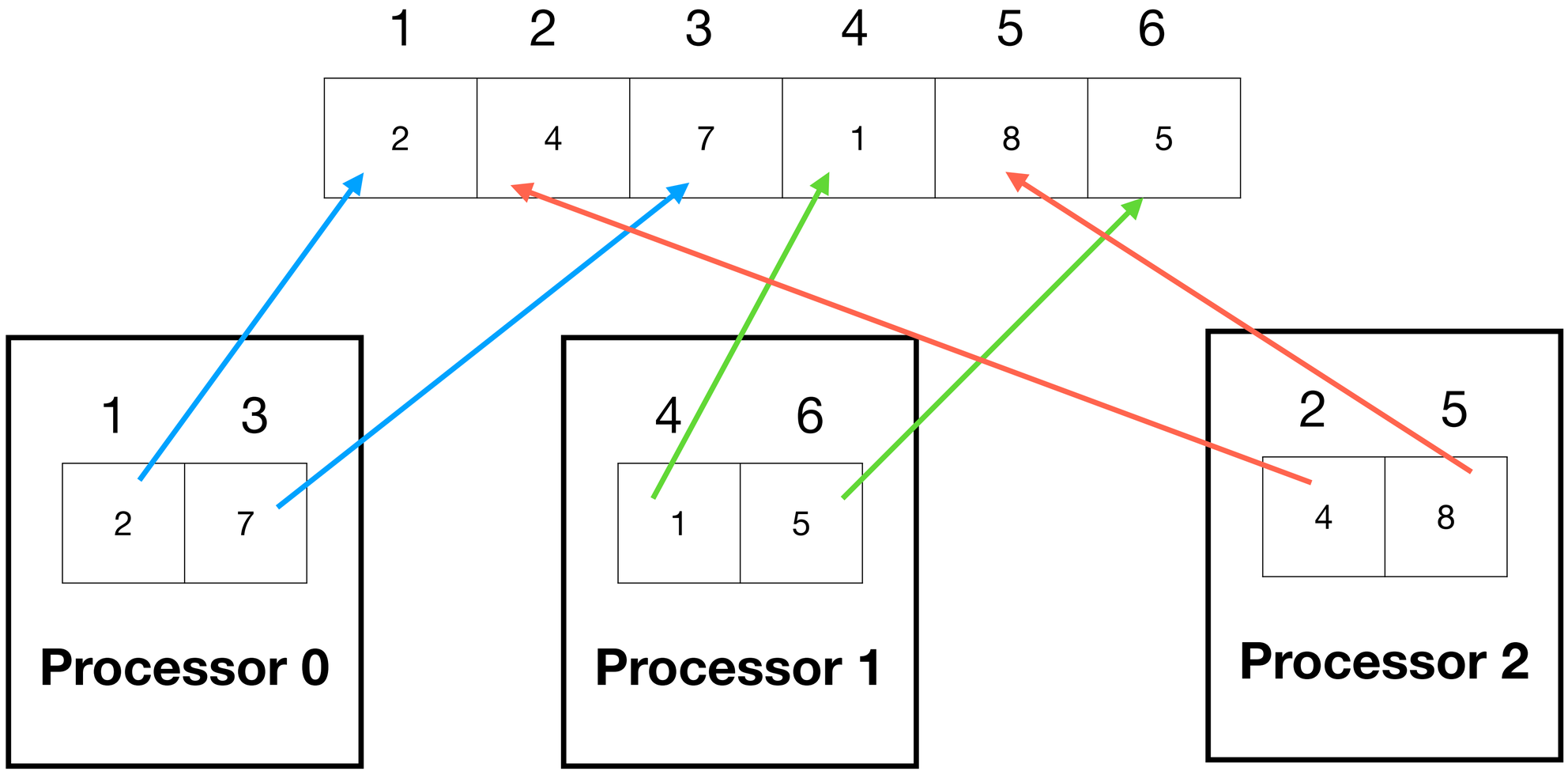
Parallel mpio run:

Thu Nov 19 12:55:27 CET 2020
Thu Nov 19 12:57:14 CET 2020

Moltes gràcies!

Lesson #4

What I was trying to do - at a small scale



Lesson #4

What I was trying to do - at a small scale

Lesson #4

The writing problem

- How to store data efficiently (hdf5 example)
  - Reduction + order + serial write = ~10 sec
  - Parallel MPIO write + order when writing = ~2 min
- If the dataset is too big reduction is not an option (not enough memory)
- If many files need to be written, a 2 min IO is a tough bottleneck

Bones,

si t'anés bé connectar-te al chat del BSC podríem comentar-ho per allà, crec que seria més àgil.

Tens un petit codi d'exemple que treballi només amb h5py i que mostri aquest tema?

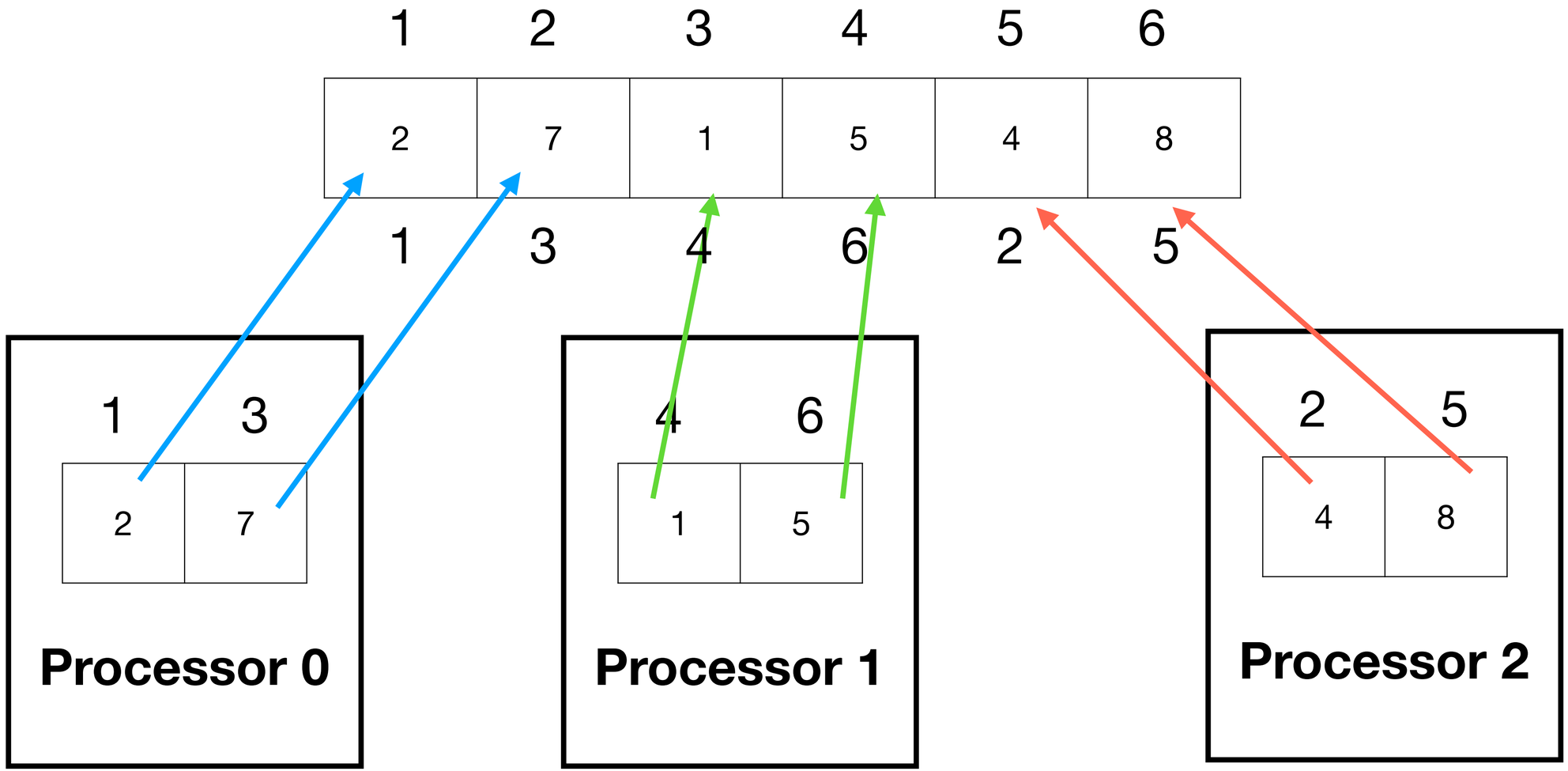
Una de les coses que vaig trobar és que en alguns sistemes era millor fer una escriptura un a un (amb un simple for amb barrier), que no deixar que es gestionés la escriptura en paral·lel automàticament. Una altra cosa a provar seria si aquest problema es manifesta al escriure tots els procesos en el mateix dataset o en datasets diferents, un per a cada procés. **Si el dataset fos col·lectiu, llavors s'hauria de veure si cada procés escriu en posicions contigües o no, ja que escriure en posicions no contigües fa baixar la performance.**

Si em passes un exemple petit que pugui provar a un node de MN4 m'ho miro i ho comentem.

Salutacions,

Lesson #4

How it should be done - contiguous writing



Lesson #4: having a smart and efficient writing of the dataset is important

Lesson #4.1: don’t do unnecessary operations

At this point we can comfortably operate with the dataset

Issue #5: sampling

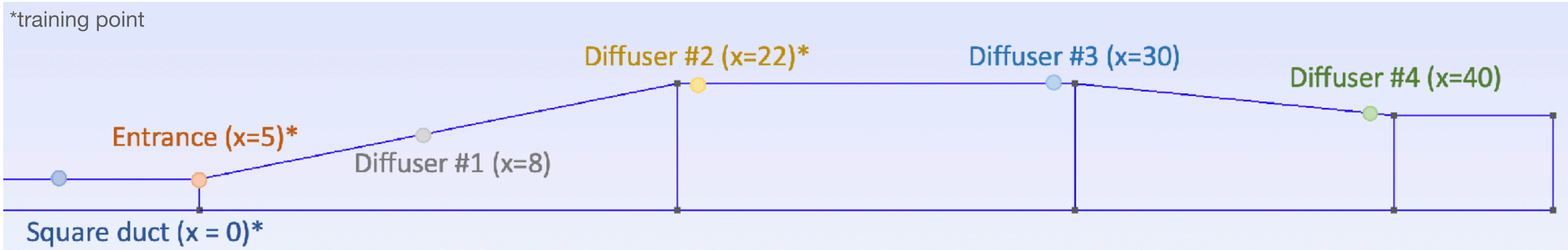
Lesson #5

Sampling, or how to obtain the data

- Dealing with the whole dataset is prohibitive…
  - Need to operate with a large number of cores
  - Dataset is too large to operate with single/few cores
  - Observing the whole dataset may be difficult to interpret
  - Machine learning algorithms don’t necessarily need the whole database
- We need to come up with smart ways of sampling the data

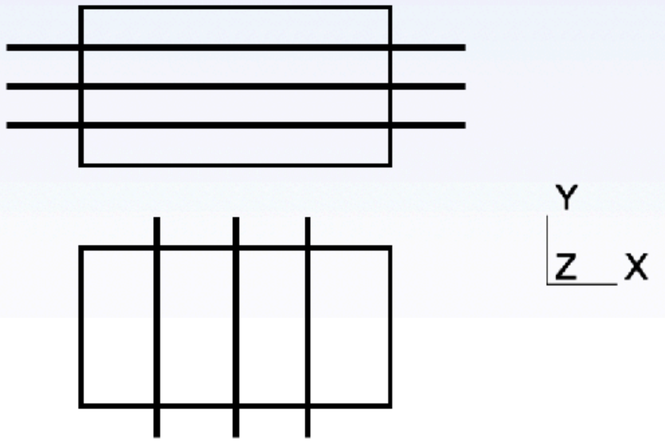
Lesson #5

First approach - Sampling with pyAlya



At each of the locations:

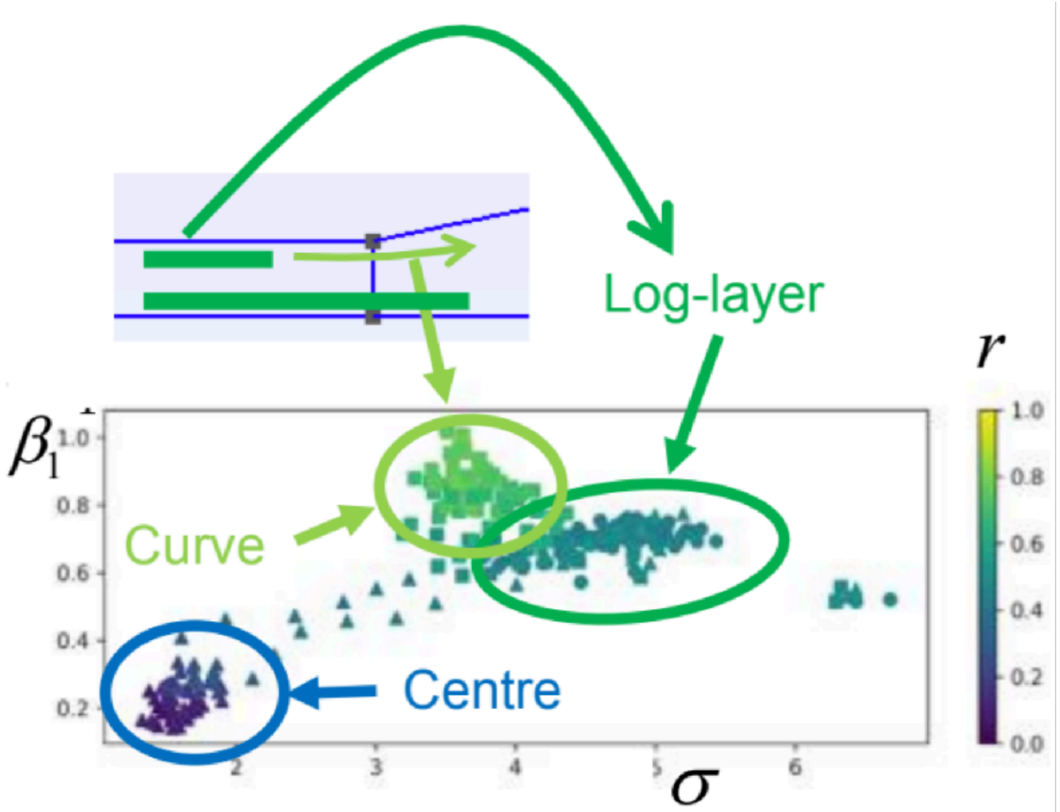
- Data is obtained at 3 vertical lines and 3 horizontal lines.
- Points with $y^+ < 30$ have been filtered out.
- Computed desired parameters on the lines.



Lesson #5

What we can extract from the data

- Key parameters of the flow have been computed and observed
- Analysis from the experts:
  - Different physics can be identified
  - Also on different regions of the flow
  - This is consistent with the theory



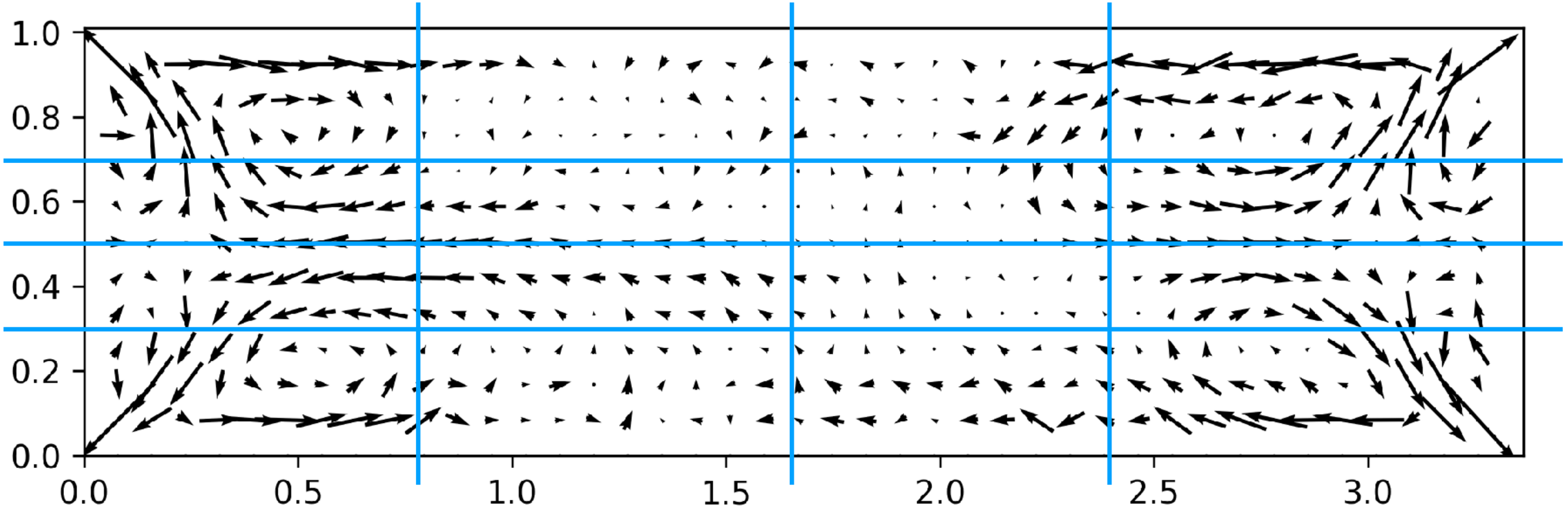
$$\sigma = \frac{sK}{\varepsilon}, \qquad r = \frac{\Omega_{ij}\Omega_{ij}}{s^2}$$

where $s^2 = S_{ij}S_{ij} + \Omega_{ij}\Omega_{ij} = (\nabla U)^2$

Lesson #5

Sampling caveats - Example

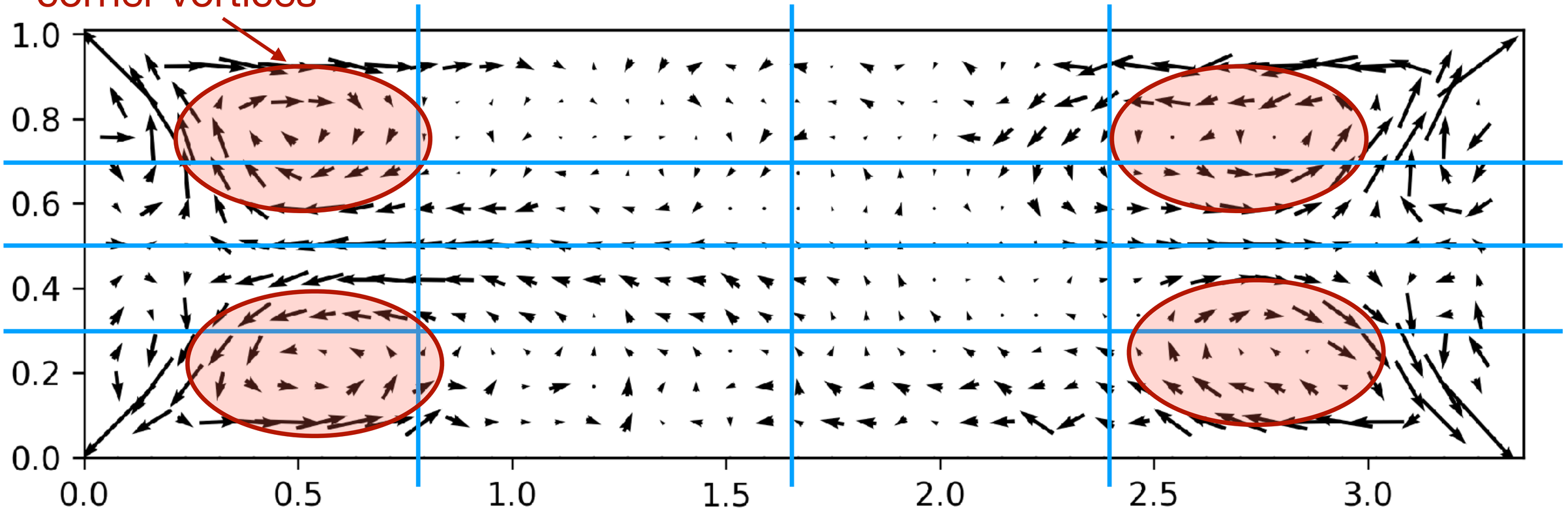
in blue are the sampling lines…



Lesson #5

Sampling caveats - Example

and we miss important features of the flow



We could also sample randomly but the issue remains…

How can we capture all the relevant physics of the dataset?

Is there a smarter way of sampling/reducing the data?

Lesson #5

Variational multi-encoders (VAE) and reduced order models (ROM)

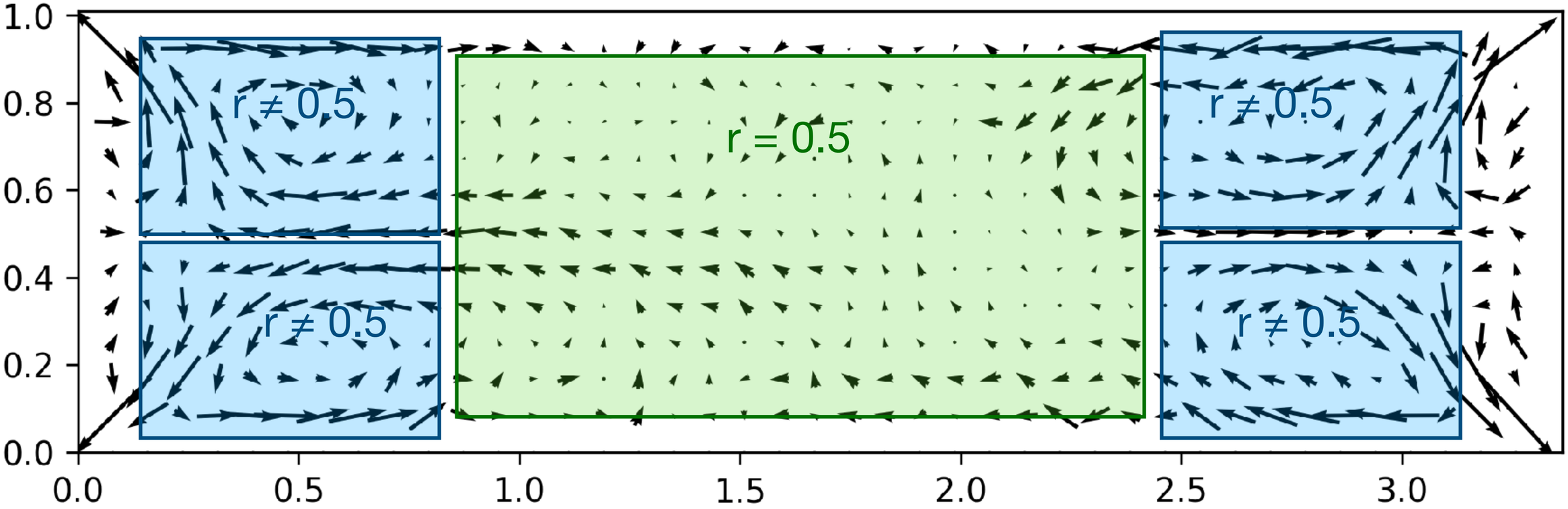
- Can be trained with the full dataset
- Can learn the different underlying physics according to parameters
- Can be used to select/visualise different areas of the dataset
- Can produce a down-sampled dataset with high fidelity **on the region of interest**

Lesson #5

VAE and ROM - Example

r, a measure of the rotation of the flow

r = 0.5 -> the flow is aligned else there is rotation



Let’s see an example of reduced order model

Lesson #5

Proper orthogonal decomposition (POD)

Left singular vectors — Singular values — Right singular vectors

$$X = \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_m \\ \vdots & \vdots & \vdots \end{bmatrix} = \mathrm{U}\Sigma V^T = \begin{bmatrix} \vdots & \vdots & \vdots \\ U_1 & U_2 & U_n \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_m \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots \\ V_1 & V_2 & V_m \\ \vdots & \vdots & \vdots \end{bmatrix}^T$$

(n, m) $\quad$ (n, n) $\quad$ (n, m) $\quad$ (m, m)

(m, m) non zero

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \ldots \geq \sigma_m$$

Truncate $U\Sigma V$ matrices

$$U(n, n) \longrightarrow U_r(n, r)$$

$$\Sigma(n, m) \longrightarrow \Sigma_r(r, r)$$

$$V^T(m, m) \longrightarrow V_r^T(r, m)$$

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \ldots \geq \sigma_r$$

Parameter r can be given as an integer or residual value:

$$EE(r) = \frac{\sigma_{r+1}^2 + \sigma_{r+2}^2 + \cdots + \sigma_m^2}{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_m^2} \leq \epsilon_1$$

Reconstruct the flow with truncated matrices

$$X_{POD} = \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_m \\ \vdots & \vdots & \vdots \end{bmatrix}$$

(n, m)

Calculate error in reconstruction

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m} \left\| x_i - x_{POD\,i} \right\|^2}{\sum_{i=1}^{m} \|x_i\|^2}}$$

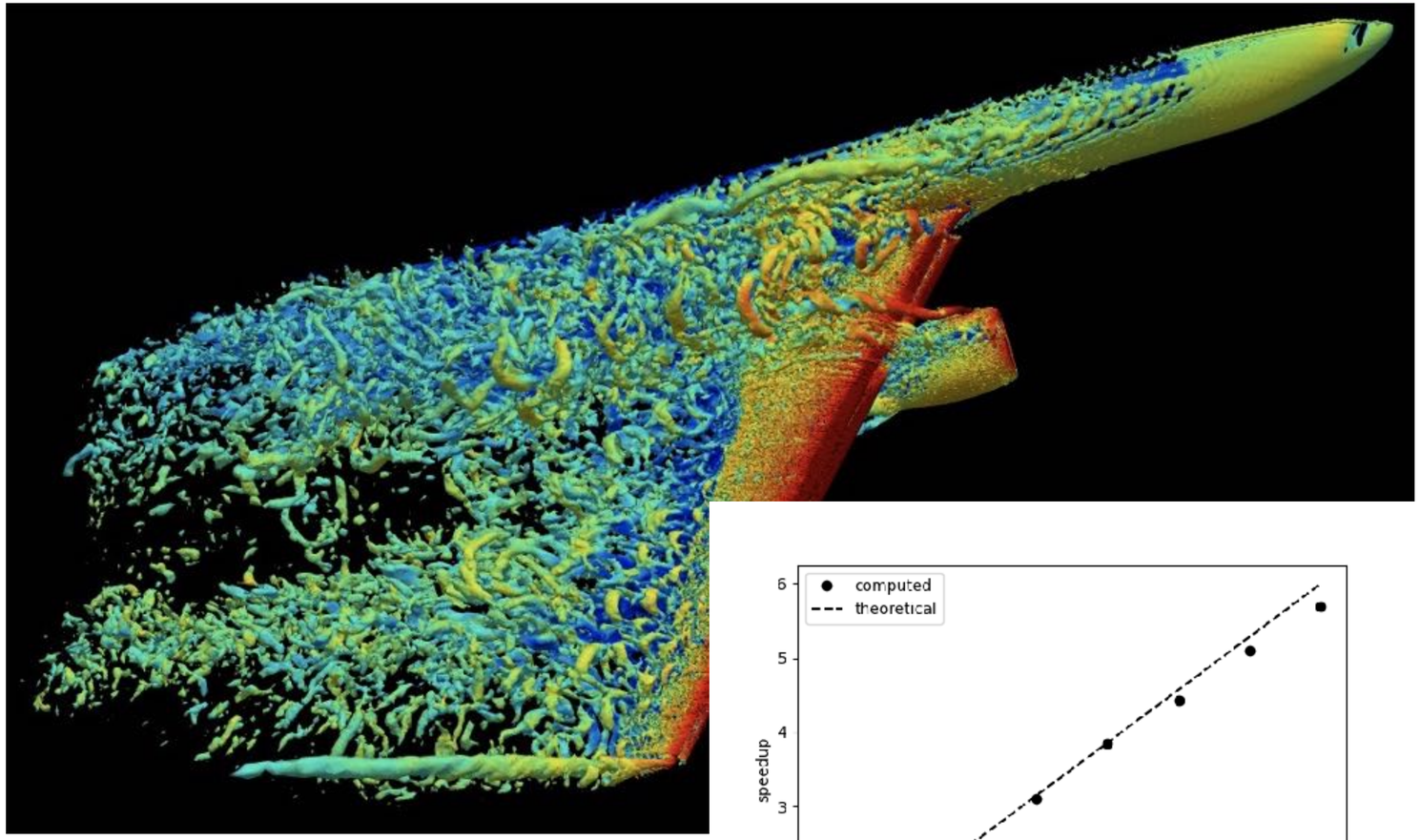
$\|\cdot\| \equiv Euclidean\ norm$

- Plot and animate reconstructed/original flow
- Plot POD modes (U_i) one by one, with (V_i) Fourier spectrum
- Retrieve dominant frequency from the spectrum

Lesson #5

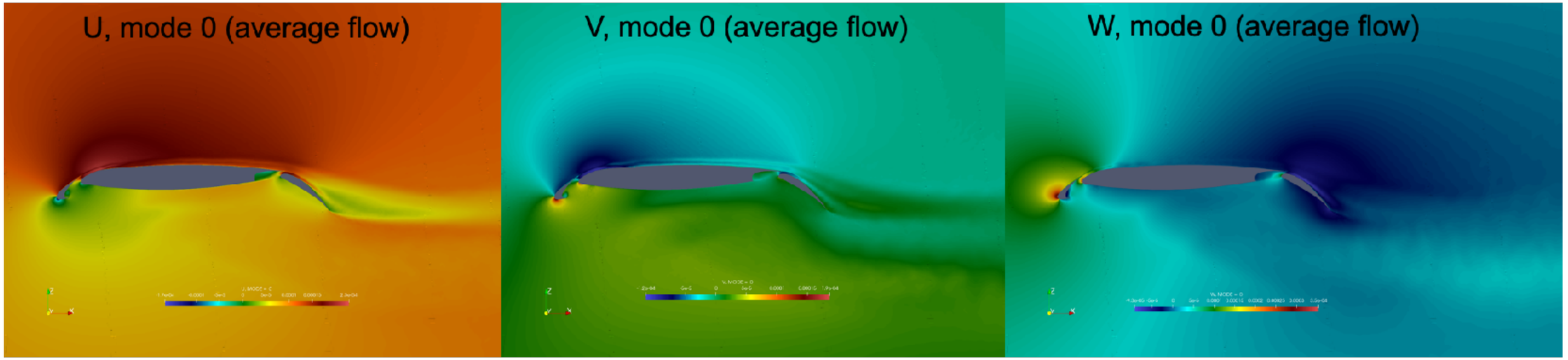
Parallel POD preliminary results

- Parallel SVD based on a “Tall and skinny” QR (TSQR) decomposition
- Implemented in Python/Cython with some C parts, directly linking with BLAS/LAPACK
- Allows us to handle large cases, for example 273M elements (image on the right)
- Has a good scalability for a large number of processors

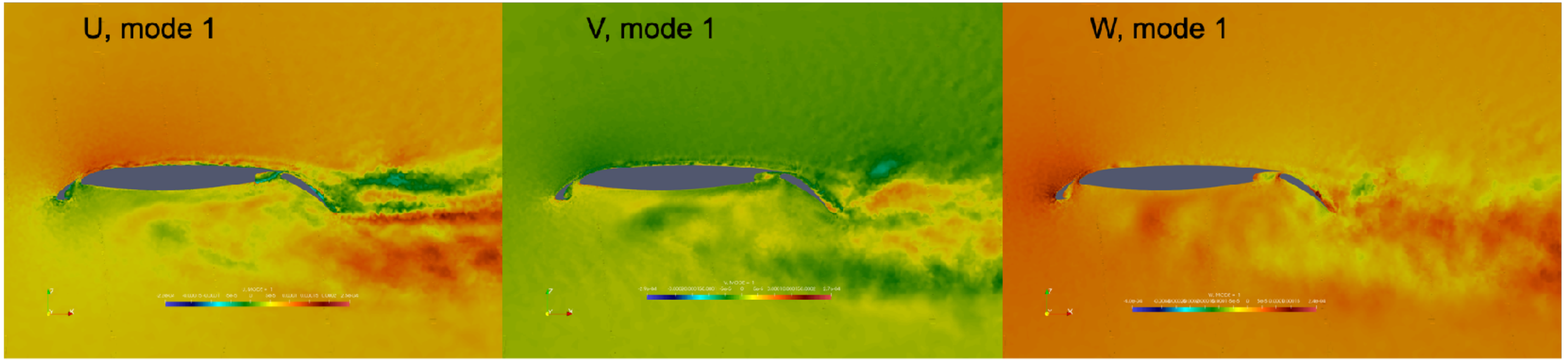


Lesson #5

Parallel POD preliminary results



First POD results of CRM plane, wing section in high lift configuration.



Lesson #5.1: understanding the dataset (physics, etc.) helps a lot when sampling the data

Lesson #5.2: ROM and VAE can be used to perform a smart exploration and sampling of the dataset

Some experiences with Multi Expression Programming (MEP)

Experiences with MEP

General overview

Multi Expression Programming (MEP) for regression problems allows using a set of data (training data) to regress and obtain mathematical expressions that follow a set of testing data (target data). MEP allows to encode multiple solutions in the same chromosome. In the simplest variant, MEP chromosomes are linear strings of instructions.

Pros and cons:

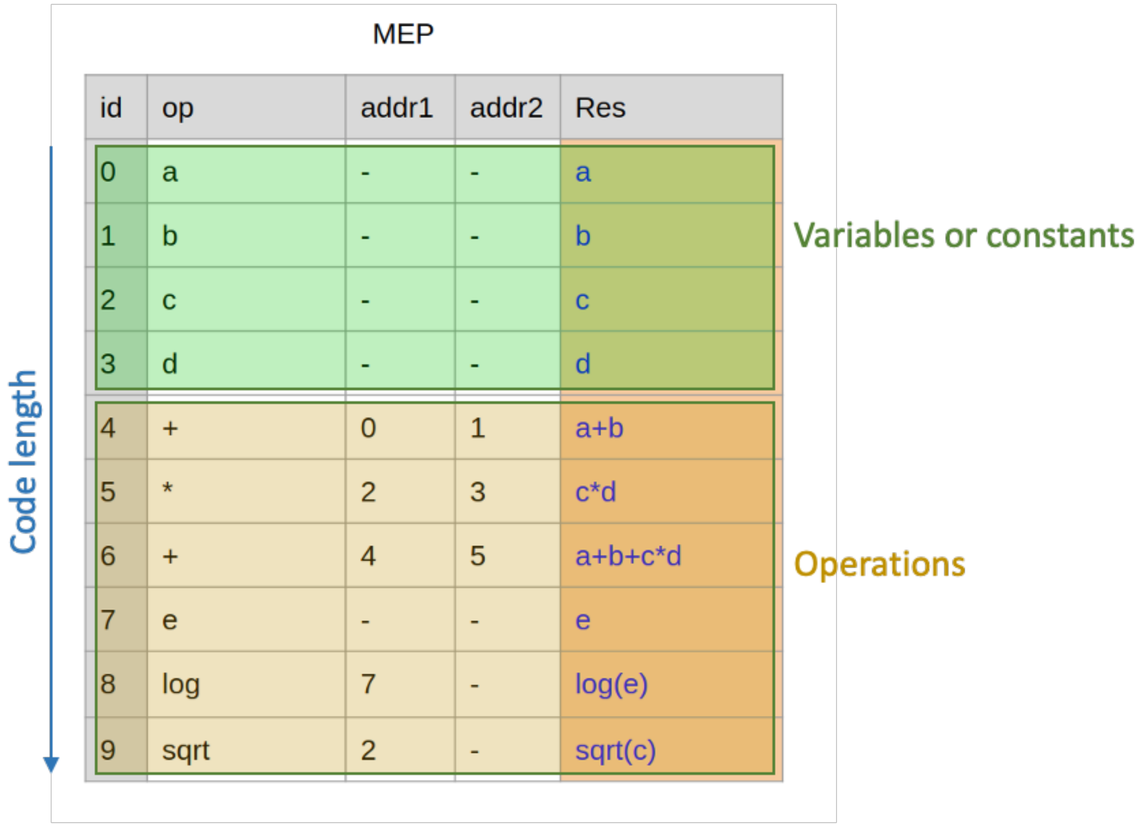
- Easier to code with respect to GEP: tables instead of trees, static structures instead of dynamic structures
- Higher “Density of information” thanks to the unexpressed genes allows to explore the search space more efficiently
- Easier to tune and adapt to special problems
- MEP approach is less general

Experiences with MEP

MEP Chromosome

A structure containing:

- MEP chromosome table
- Constants array
- When a chromosome is generated it is ensured that enough variables/ constants and operations are represented within the code length of the chromosome



Experiences with MEP

MEP Fitness

Each chromosome stores 3 fitness and their associated complexity:

- Overall fitness: best fitness within a given complexity
- Best fitness: best fitness regardless of the complexity
- Best complexity: best fitness with minimum complexity

The complexity associated with each operation and variable is set as an input parameter

Constants do not add complexity to the expression

Operations associated complexity:
+: 10
*: 5

MEP

id	op	addr1	addr2	Res	complexity
0	a	-	-	a	5
1	b	-	-	b	5
2	c (constant)	-	-	c	0
3	d	-	-	d	5
4	+	0	1	a+b	5+5+10 = 20
5	*	2	3	c*d	0+5+5 = 10
6	+	4	5	a+b+c*d	20+10+10 = 40
7	e	-	-	e	
8	log	7	-	log(e)	
9	sqrt	2	-	sqrt(c)	

Example of the computation of the complexity topdown from a chromosome (MEP table).

A similar procedure occurs with the fitness.

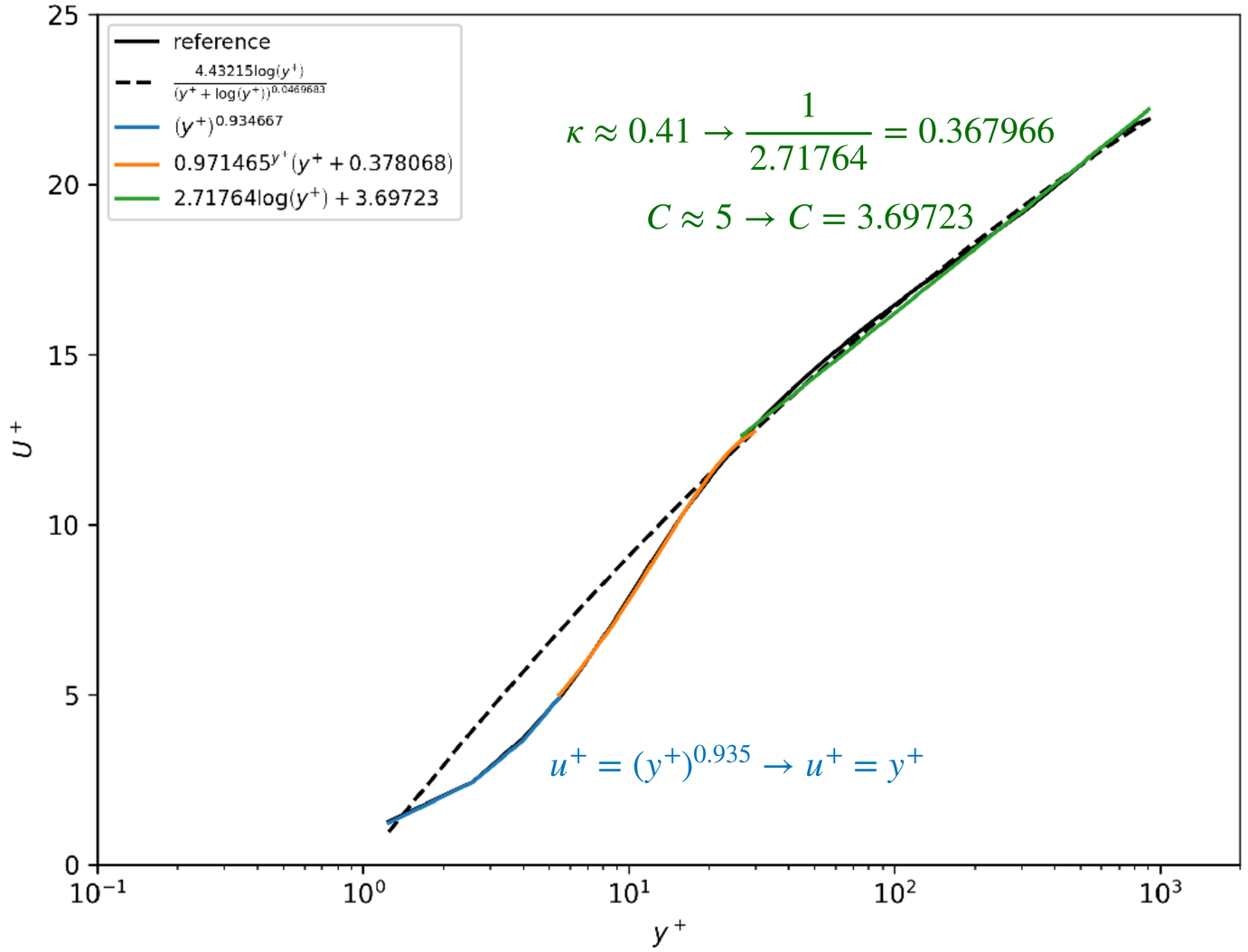
Experiences with MEP

MEP Main features

- Programmed in Python/Cython with an underlayer C API
- Symbolic representation of expressions by means of the SymPy package
- Addition of initial guess expressions
- Exploring population area that is far away from the current best
- Coefficients optimization using SciPy

Experiences with MEP

Some preliminary testing results - law of the wall



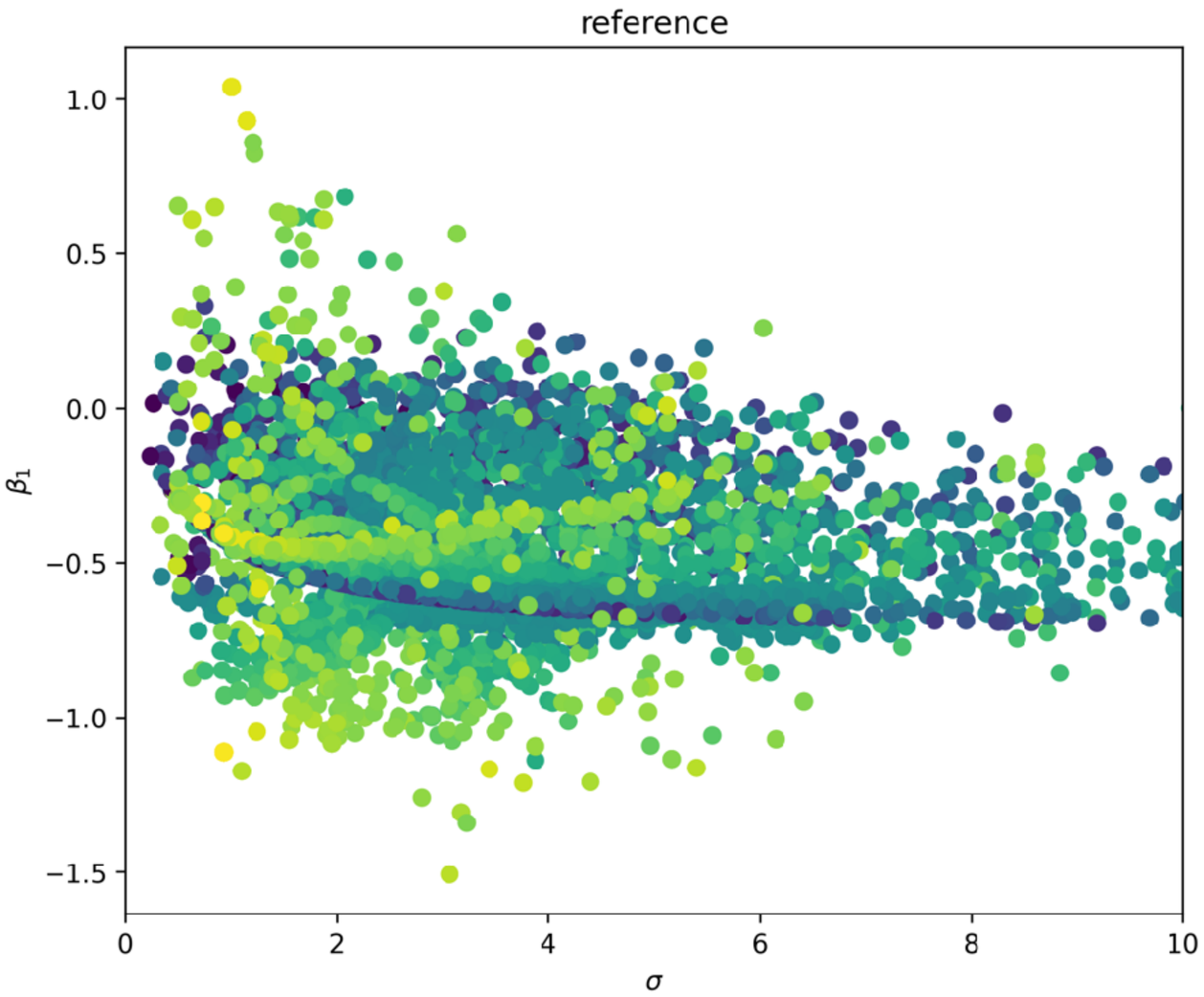
Obtained results are very close to these on the literature

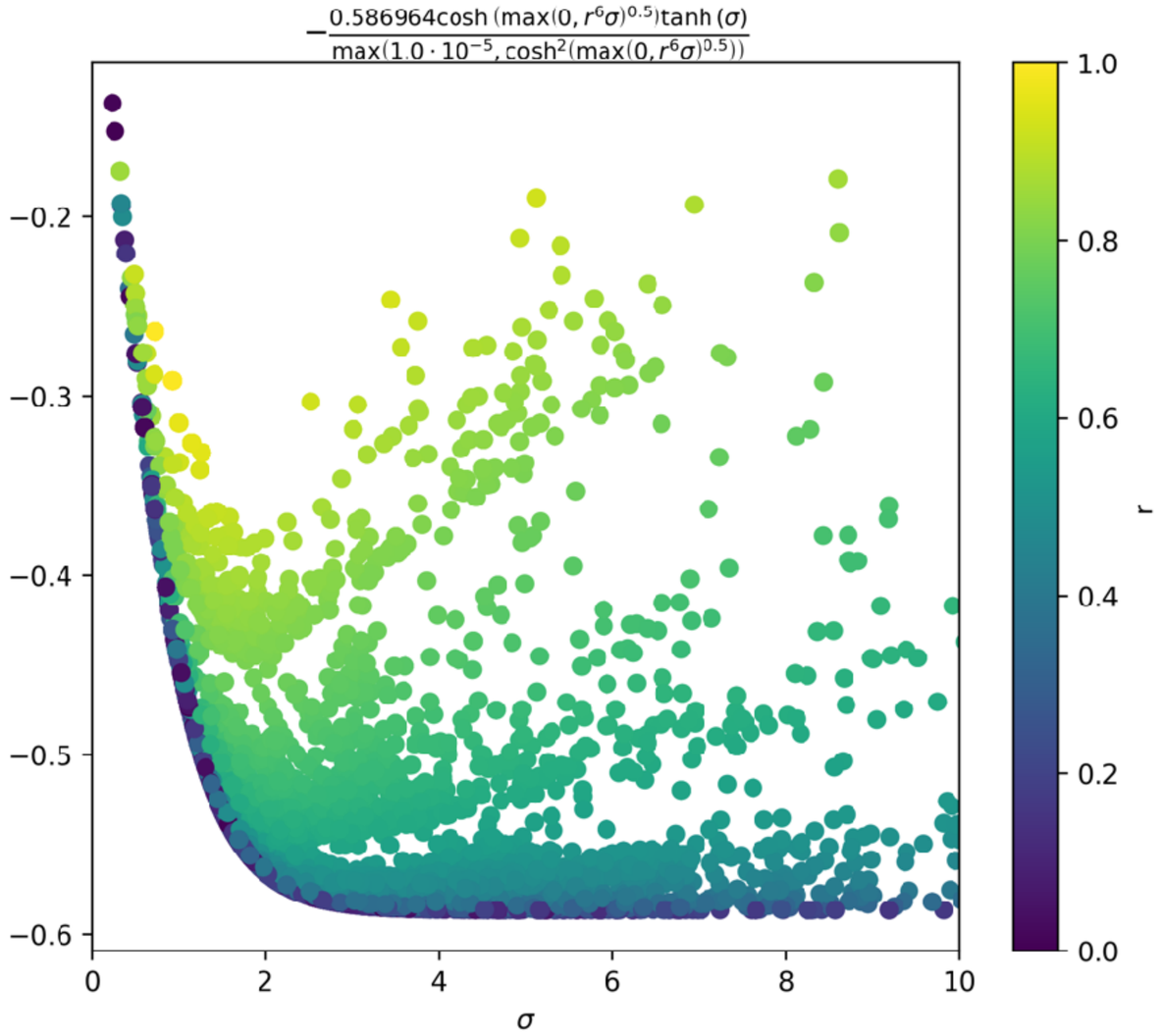
$$\frac{1}{\kappa}\log(y^+) + C$$

With $\kappa \approx 0.41$ and $C \approx 5$

Experiences with MEP

Some results from HiFiTurb

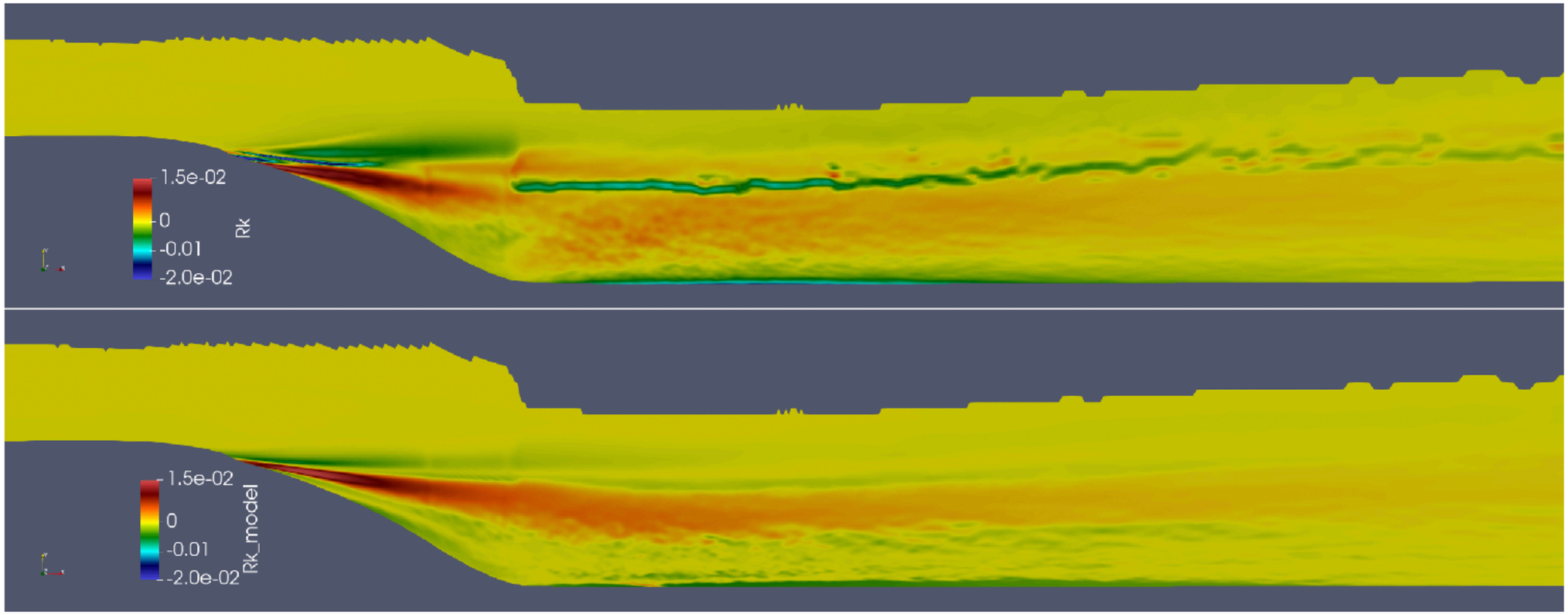




Experiences with MEP

Some results from HiFiTurb

R_K data vs model

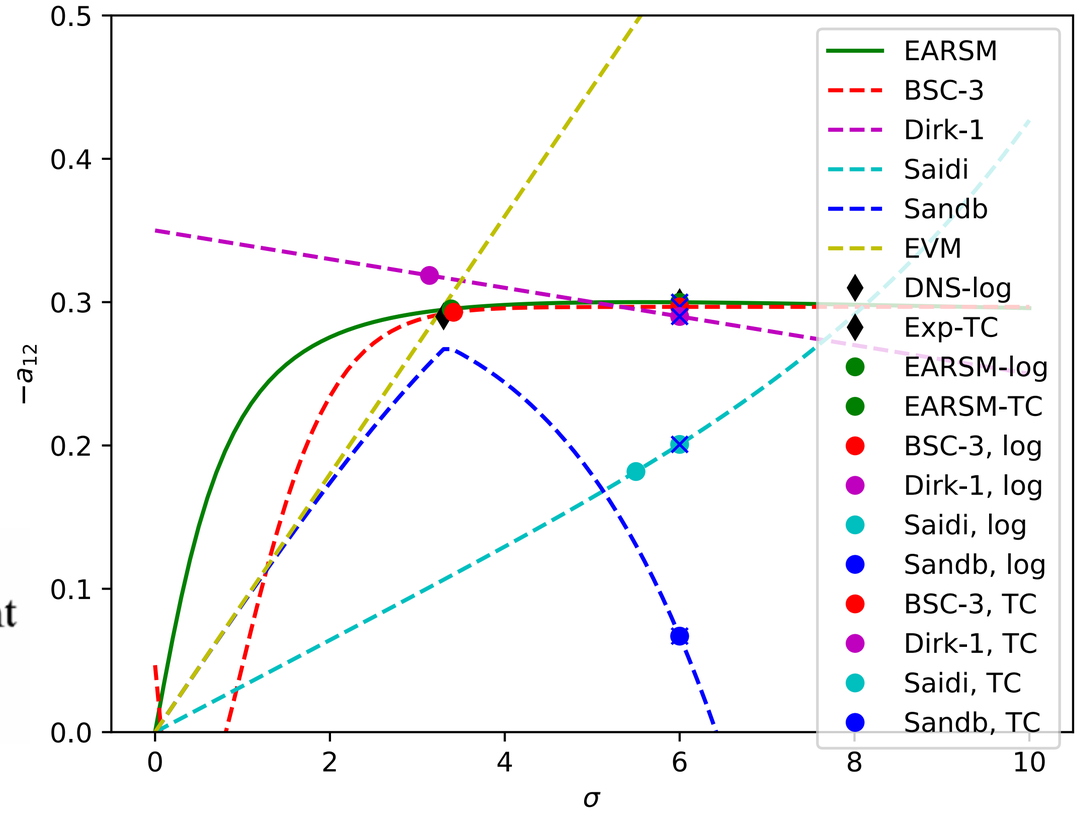


Experiences with MEP

Some results from HiFiTurb

- Comparison of the obtained expressions with the ones on the literature
- Obtaining a generic model vs obtaining a specific model
- On Philippe Spalart's words:

- Many times, the product is "not a turbulence model"
  - It uses invalid quantities, such as velocity magnitude or pressure gradient
  - It applies only to a small number of flows
  - It uses a very costly Neural Network, which is not even in the paper



Comparison of the performance of different models produced by BSC-Numeca and the literature (courtesy of S. Wallin)

Experiences with MEP

Some final conclusions

- Obtaining a general model is challenging
- The formulation proposed helps a lot to achieve the desired results
- Often the expression we are looking for is very complex, this presents:
  - A large number of generations is needed to achieve convergence
  - One solution can be represented by a family of expressions
- Optimising the coefficients on the loop helps achieve convergence faster
- Overall it depends on the branch of solutions MEP decides to follow

Final lesson: whatever the analysis and the method might be, having an understanding of the underlying physics is one of the keys to success