# WORKSHOP
# An introduction to NEK5000

**Diego C. P. Blanco**
Advisor: **André V. G. Cavalieri**

Divisão de Engenharia Aeronáutica
Instituto Tecnológico de Aeronáutica

November 28th, 2023

## Table of Contents

## Introduction

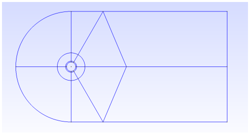
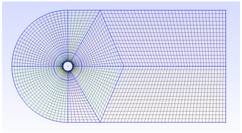
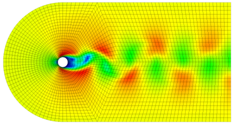
## Objectives

### Understand

- Main theoretical ideas behind the solver;
- How the code is organized.

### Demonstrate

- Case from zero: generate the appropriate files from a mesh grid, run the case and post-process the data;
- KTH framework tools using a 3D case.

## Overview

### Historical background

In the 1980's Paul Fischer, Lee Ho, and Einar Rønquist (M.I.T) developed the incompressible fluid flow solver NEKTON, that today is integrated to Ansys Fluent. In the 1990's, the code was branched off in a research focused version called NEK5000 under the BSD open-source license.

### Software support

- Main programming language is FORTRAN 77, but FORTRAN 90 syntax is allowed;
- Support for all Unix-like systems (Linux and Mac);
- Support for Windows 10/11 is provided via the Windows Subsystem for Linux (WSL v2).

## NEK5000

**Definition**

Nek5000 is a **fast** and **scalable** open source CFD solver.

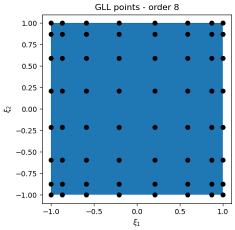
- **fast**: Efficient algorithms (pre-conditioners, matrix-free manipulation, etc.) written in a fast compiled language (Fortran)
- **scalable**: Runs on platforms ranging from laptops to the world's fastest computers, with efficient parallelization.

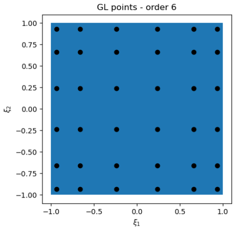
However, the solver has limitations: only solves incompressbile and low Mach-number Navier-Stokes and only accepts quadrilateral (2D) and hexahedral (3D) structured meshes.

## Theory

## Spatial discretization

- The NEK5000 solver implements the **continuous Galerkin** method to solve the incompressible Navier-Stokes equations;
- For the $\mathbb{P}_n - \mathbb{P}_{n-2}$ method, trial functions are constructed using
    - **Nodal basis** of Lagrange interpolation polynomials on **Gauss-Lobato-Legendre (GLL) points** for velocity;
    - **Modal basis** of Legendre polynomials on **Gauss-Legendre points** for pressure.





## Continuity

Since the GLL quadrature has points placed at the edges of the element, continuity for the velocity field is enforced by equalizing overlapping edge nodes.

### Consequence

- Velocity fields: $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$ are $C0$ continuous
- Gradient fields: $\frac{\partial \boldsymbol{u}}{\partial x}$, $\frac{\partial \boldsymbol{v}}{\partial y}$, $\frac{\partial \boldsymbol{w}}{\partial z}$, etc., are discontinuous (averaged over overlapping edge points)
- Pressure field: $p$ is discontinuous.

## Time integration

An implicit backwards-differenting scheme of order $k$ (BDF$k$) is employed for time integration. For instance

BDF2: $N_2'(t^{n+1}) = \frac{3u^{n+1} - 4u^{n} + u^{n+1}}{2h} = f^{n+1}$

where $f^{n+1}$ are the non-linear terms, which are explicitly marched in time.

**Consequence**

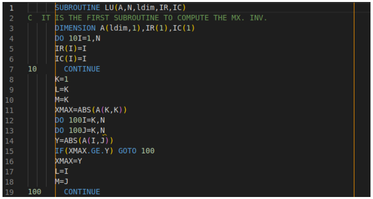
A proper restart file should contain more than one snapshot. If only one snapshot is present, the restart procedure will generate numerical errors.

## Implementation

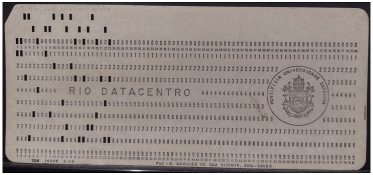
## The FORTRAN language

### FORTRAN 77 - standard fixed form: the legacy of punch cards

- There is no distinction between lowercase and UPPERCASE;
- The first 72 columns of each line are scanned;
- The first 5 columns must be blank, contain a numeric label or comment symbol;
- Long lines are truncated.



```
      SUBROUTINE LU(A,N,ldim,IR,IC)
C  IT IS THE FIRST SUBROUTINE TO COMPUTE THE MX. INV.
      DIMENSION A(ldim,1),IR(1),IC(1)
      DO 10I=1,N
      IR(I)=I
      IC(I)=I
10      CONTINUE
      K=1
      L=K
      M=K
      XMAX=ABS(A(K,K))
      DO 100I=K,N
      DO 100J=K,N
      Y=ABS(A(I,J))
      IF(XMAX.GE.Y) GOTO 100
      XMAX=Y
      L=I
      M=J
100     CONTINUE
```



## Common blocks I

Fortran 77 has no global variables. In order to be able to pass a large number of parameters between subroutines, common blocks are included in a subroutine before the declaration of variables.

Common blocks are contained in UPPERCASE files in the `Nek5000/core/` directory, or in the case directory.

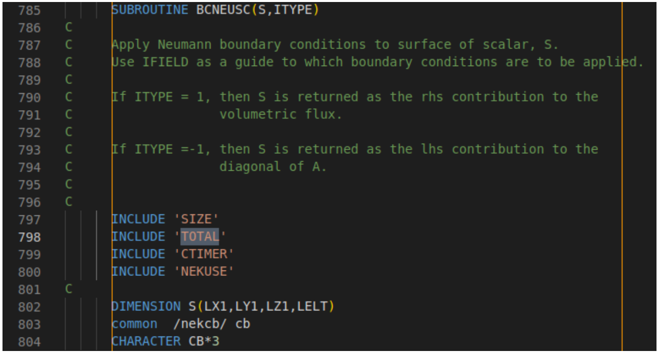
### Examples

- `SIZE`: Case specific. Controls the size of allocated arrays in memory;
- `INPUT`: Input parameters from pre-processors, including runtime flags and boundary conditions;
- `GEOM`: Geometry arrays for velocity and pressure meshes;
- `TOTAL`: Includes all important sets of common blocks;

## Common blocks II

**Cons**

Common blocks make it very difficult to track the variables being modified by specific subroutines! Specially the `TOTAL` one!



```
      SUBROUTINE BCNEUSC(S,ITYPE)
C
C     Apply Neumann boundary conditions to surface of scalar, S.
C     Use IFIELD as a guide to which boundary conditions are to be applied.
C
C     If ITYPE = 1, then S is returned as the rhs contribution to the
C                   volumetric flux.
C
C     If ITYPE =-1, then S is returned as the lhs contribution to the
C                   diagonal of A.
C
C
      INCLUDE 'SIZE'
      INCLUDE 'TOTAL'
      INCLUDE 'CTIMER'
      INCLUDE 'NEKUSE'
C
      DIMENSION S(LX1,LY1,LZ1,LELT)
      common  /nekcb/ cb
      CHARACTER CB*3
```

## Core functions

The `Nek5000/core/` directory also contains the functions responsible for the main features of the solver, such as spatial discretization, time-stepping, input-output, linear solvers, etc.

Since the Nek5000 code is open source, one can edit every function in the core structure.

The main subroutine is `nekton()` in the `drive.f` file.

**Attention**

IF YOU DO NOT KNOW EXACTLY WHAT YOU ARE DOING, YOU SHOULD NOT EDIT THE FILES INSIDE `Nek5000/core/`!

## Case files I

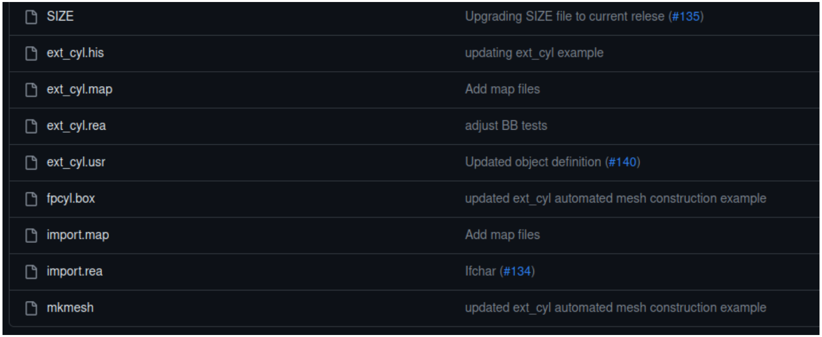
The necessary files to run a Nek5000 case are displayed below:

- `<casename>.par`: Contains the runtime parameters for the simulation.
- `<casename>.re2`: Binary file. Stores the mesh and boundary groups;
- `<casename>.usr`: User defined functions. This is the main interface the the Nek5000 code;
- `<casename>.ma2`: Binary file. Stores the mesh partitioning;
- `SIZE`: Common blocks already described before;
- `SESSION.NAME`: Lists the path to relevant files. Automatically generated by the command `nekmpi`.

## Case files II

**Observation**

Most examples present in the `Nek5000/examples/` directory do not follow this file structure, but rather an older arrangement with `.rea` and `.map` files.



| | |
|---|---|
| SIZE | Upgrading SIZE file to current relese (#135) |
| ext_cyl.his | updating ext_cyl example |
| ext_cyl.map | Add map files |
| ext_cyl.rea | adjust BB tests |
| ext_cyl.usr | Updated object definition (#140) |
| fpcyl.box | updated ext_cyl automated mesh construction example |
| import.map | Add map files |
| import.rea | lfchar (#134) |
| mkmesh | updated ext_cyl automated mesh construction example |

## Repository

## Case setup

**Step 1**: Clone the git repository

```
$ git clone
↪   https://github.com/DiegoCPB/Workshop-NEK5000-ITA2023.gi
```

**Step 2**: Change to workshop directory

```
$ cd Workshop-NEK5000-ITA2023
```

**Step 3**: Execute the setup script. It will install the required dependencies (Ubuntu), clone the KTH framework repository and install the visualization software VisIt. Also works for the clusters at ITA (CentOS 7).

```
$ ./setup.sh
```

**Step 4**: Close and reopen the terminal

**Step 5**: Check if VisIt is working

```
$ visit
```

## Overview

The repository contains multiple directories:

- `cyl2d` : 2D cylinder case ready to be executed.
- `cyl2d_ZERO` : Directory containing only the 2D cylinder mesh script.
- `cyl3d_STAT` : 3D cylinder case ready to be executed. Computes 2D statistics, averaging over time and span using the KTH framework. Needs a computer with at least 16 CPUs.
- `cyl3d_PSTAT2D` : Files to post-process the statistics of the 3D cylinder using the KTH framework.
- `Matlab` : Extra files to read Nek5000 data into Matlab.

## 2D cylinder case

## Mesh files I

Change to the `cyl2d_ZERO` directory

```
$ cd cyl2d_ZERO
```

From the gmsh script `cyl2d.geo` that defines the mesh, we need to generate a `.msh` file. The command

```
$ gmsh -2 cyl2d.geo
```

outputs a `cyl2d.msh` file. The argument `"-2"` means that a 2D mesh is generated. NEK5000 only accepts elements of order 2 and `.msh` files version 2. These parameters are set in the last lines of the `.geo` script.

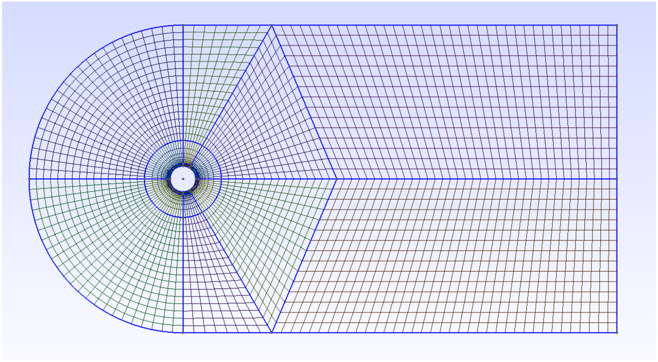
**Observation**

It is important to properly set the physical curves and surfaces on which the boundary conditions will be applied.

## Mesh files II

```
$ gmsh cyl2d.geo
```

```
Modules -> Mesh -> 2D
```



Mesh script adapted from file kindly provided by Prof. Rodrigo Moura.

## .re2 from .msh file I

The solver provides tools to convert the files from GMSH to Nek5000. These need to be compiled.

**Step 1**: Change to the tools directory

```
$ cd ~/Apps/KTH_Framework/Nek5000/tools
```

**Step 2**: Compile the gmsh2nek tool. This needs to be done only once.

```
$ ./maketools gmsh2nek
```

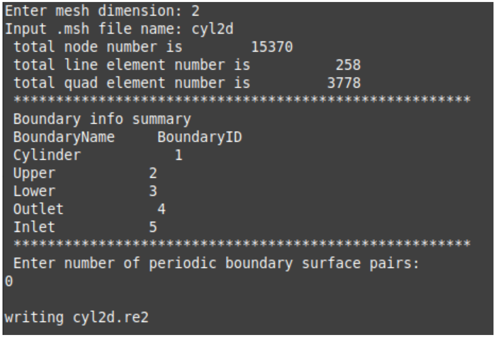
**Step 3**: Change back to the cyl2d_ZERO directory

```
$ cd ~/Workshop-NEK5000-ITA2023/cyl2d_ZERO (may vary)
```

## .re2 from .msh file II

**Step 4**: Execute `gmsh2nek`

```
$ gmsh2nek
```



```
Enter mesh dimension: 2
Input .msh file name: cyl2d
 total node number is        15370
 total line element number is          258
 total quad element number is         3778
 ******************************************************
 Boundary info summary
 BoundaryName     BoundaryID
 Cylinder            1
 Upper            2
 Lower            3
 Outlet            4
 Inlet            5
 ******************************************************
 Enter number of periodic boundary surface pairs:
0

writing cyl2d.re2
```

The information about total number of elements and Boundary IDs will be important.

## .ma2 from .re2 file

Once the mesh file is ready, we need to generate the domain partitioning.

**Step 1**: Compile the `genmap` tool, exactly like the `gmsh2nek` tool. This needs to be done only once.

**Step 2**: Execute `genmap` inside the `cyl2d_ZERO` directory to generate the `cyl2d.ma2` file.

```
$ genmap
```

## History points I

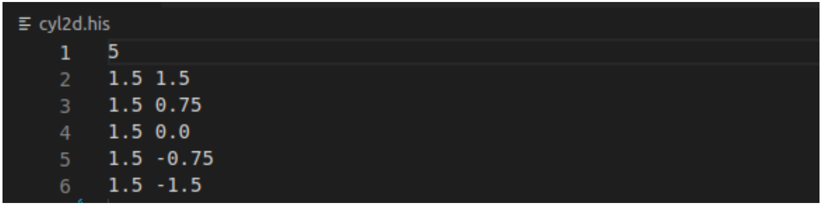
A tool to probe points at any coordinate within the domain is implemented in the solver. Nek5000 will automatically interpolate values to the desired point with spectral precision.

To use it, a `.his` file needs to be provided. Its first line should contain the number of probe points, with subsequent lines listing the coordinates separated by spaces: X Y (Z).

The values at the probe's positions will be appended to the `.his` file with the columns: Time U V (W) P Scalars.

## History points II

Example `cyl2d.his` file:


cyl2d.his

```
5
1.5 1.5
1.5 0.75
1.5 0.0
1.5 -0.75
1.5 -1.5
```

The `.his` file can be generated using scripts. The number of history points can range from a couple to millions.

## SIZE file

Controls the size of pre-allocated arrays in memory. A template is provided inside the `Nek5000/core/` directory.

**Step 1**: Copy the template to from `Nek5000/core/` directory.

```
$ cp ~/Apps/KTH_Framework/Nek5000/core/SIZE.template SIZE
```

**Step 2**: Modify variables according to the case. Example:

- `ldim = 2`
- `lx1 = 4`
- `lxd = 6`
- `lelg = 4000` (bigger than the actual number of elements)
- `lhis = 10` (bigger than the actual number of history points)
- `lorder = 2` (use BDF2 scheme)

## .par file

Simulation runtime parameters. In practice, .par files are usually copied from example cases. A list of all parameters is available here.

**Step 1**: Copy the .par file from the complete cyl2d case.

```
$ cp ../cyl2d/cyl2d.par .
```

## .usr file

Collection of subroutines accessible to the user. Among them, the subroutine `userchk()` is called at every time iteration and is used to interface with the solver. A template is provided inside the `Nek5000/core/` directory.

**Step 1**: Copy the template to from `Nek5000/core/` directory.

```
$ cp ~/Apps/KTH_Framework/Nek5000/core/zero.usr cyl2d.usr
```

**Step 2**: Modify the following subroutines:

- `userdat` to set the edges where boundary conditions are applied;
- `userbc` to set the velocity and the INLET boundary;
- `useric` to set initial conditions;
- `userchk` to create the objects where forces are integrated, compute lift and drag coefficients and output history points.

## Compiling and executing

In order to run the simulation, we need to compile the code using the parameters from the multiple files we defined up to this point.

**Step 1**: Copy the compilation recipe to the current folder.

```
$ cp ~/Apps/KTH_Framework/Nek5000/bin/makenek .
```

**Step 2**: Uncomment and edit the NEK_SOURCE_ROOT variable to the actual path to NEK5000.

```
$ nano makenek (or any other editor)
$ NEK_SOURCE_ROOT="$HOME/Apps/KTH_Framework/Nek5000"
```

**Step 3**: Compile the case.

```
$ ./makenek
```

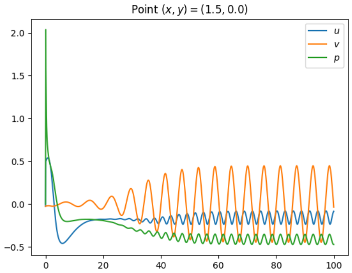
**Step 4**: If the compilation is successful, run the case. An example `cyl2d/run_nek.sh` is provided for cluster execution.

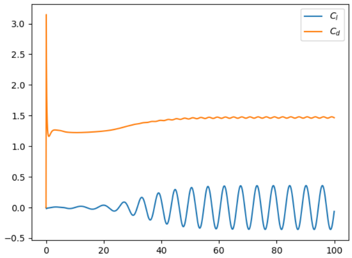
```
$ nekmpi cyl2d 4 >> cyl2d.out
$ tail -f cyl2d.out (In another terminal)
```

## Post-processing I

Inside the cyl2d_SCRIPTS/ directory are two scripts to read the history points data appended to the cyl2d.his file and read the forces computed and written in the cyl2d.out file.

```
$ python3 plot_his_data.py
$ python3 plot_ClCd.py
```



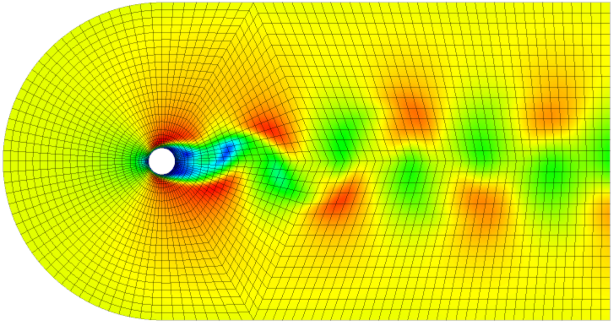


## Post-processing II

To visualize the snapshots saved as `cyl2d0.f*` files, run

```
$ visnek
$ visit -o cyl2d.nek5000
```

to open Visit.



## 3D cylinder case

## KTH framework

The KTH framework for Nek5000 is a set of subroutines that extends the functionalities of the solver. Among the new features, are:

- Selective frequency damping (SFD) to compute base flows;
- 2D and 3D statistics;
- Multi-step checkpoints;
- Improved history points interface;
- Arnoldi and power iteration routines;
- Implementation of sponge and tripping zones.

The framework is well documented here and examples are provided in the `KTH_framework/examples/` directory.

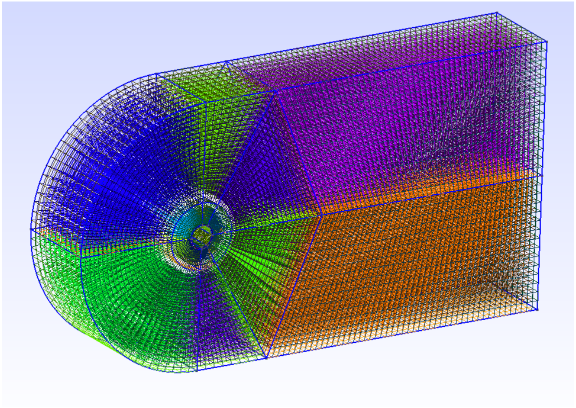
## Toolbox application

In the presented case, the statistics and checkpoint toolboxes are applied. Three main directories are present:

- `cyl3d_STAT`: Computes the solution and statistics of the 3D case
- `cyl3d_PSTAT2D/pp_Nek`: Post processes the output fiels of the 3D case and computed 2D statistics averaged over time and a homogeneous direction (span);
- `cyl3d_PSTAT2D/pp_Python`: Writes the list of points where 2D statistics are computed and plots profiles.

## cyl3d_STAT mesh files I

Mesh files (`.msh`, `.re2`, `.ma2`) are generated in a similar way of the 2D case. The mesh is essentially the same, but extruded in the $z$ direction.



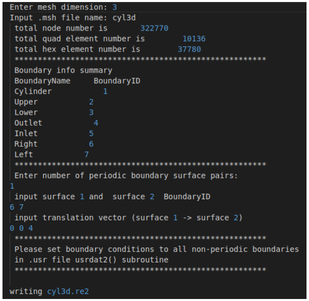
## cyl3d_STAT mesh files II

- .msh file generation, where "-3" means 3D mesh

  `$ gmsh -3 cyl3d.geo`

- .re2 file generation. Setup of periodic boundaries.

  `$ gmsh2nek`



```
Enter mesh dimension: 3
Input .msh file name: cyl3d
 total node number is       322770
 total quad element number is        10136
 total hex element number is        37780
 ******************************************************
 Boundary info summary
 BoundaryName     BoundaryID
 Cylinder            1
 Upper           2
 Lower           3
 Outlet           4
 Inlet           5
 Right           6
 Left           7
 ******************************************************
 Enter number of periodic boundary surface pairs:
1
 input surface 1 and  surface 2  BoundaryID
6 7
 input translation vector (surface 1 -> surface 2)
0 0 4
 ******************************************************
 Please set boundary conditions to all non-periodic boundaries
 in .usr file usrdat2() subroutine
 ******************************************************

writing cyl3d.re2
```

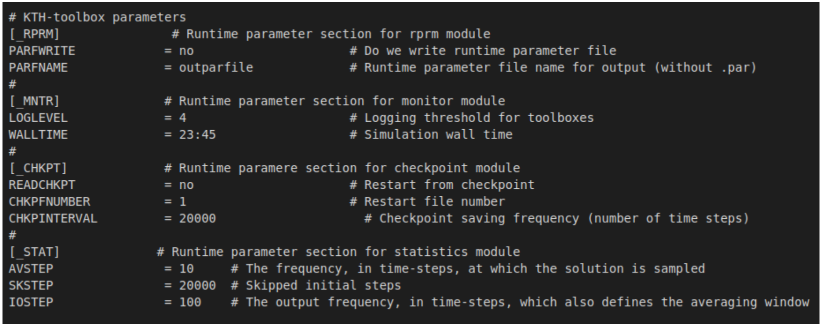
- .ma2 file generation.

  `$ genmap`

## cyl3d_STAT case files I

The SIZE file is setup exactly the same way as in the 2D case. Only the number of elements `lelg` is scaled accordingly.

The `.par` file is similar, but extra parameters are added:



```
# KTH-toolbox parameters
[_RPRM]                # Runtime parameter section for rprm module
PARFWRITE             = no                        # Do we write runtime parameter file
PARFNAME              = outparfile                # Runtime parameter file name for output (without .par)
#
[_MNTR]               # Runtime parameter section for monitor module
LOGLEVEL              = 4                         # Logging threshold for toolboxes
WALLTIME              = 23:45                     # Simulation wall time
#
[_CHKPT]              # Runtime paramere section for checkpoint module
READCHKPT             = no                        # Restart from checkpoint
CHKPFNUMBER           = 1                         # Restart file number
CHKPINTERVAL          = 20000                       # Checkpoint saving frequency (number of time steps)
#
[_STAT]              # Runtime parameter section for statistics module
AVSTEP                = 10       # The frequency, in time-steps, at which the solution is sampled
SKSTEP                = 20000    # Skipped initial steps
IOSTEP                = 100      # The output frequency, in time-steps, which also defines the averaging window
```

## cyl3d_STAT case files II

An extra common block file STATD is included, the only necessary change concerns the variable `stat_rdim = 1`, which set the average to time and homogeneous direction.

In the `.usr` file, the boundary and initial conditions subroutines stay untouched. The `userchk` now calls specific subroutines to initialize the framework, monitor the solution, write multistep checkpoints and perform averages. Extra subroutines are included to register, initialize and finalize the toolboxes (`frame_usr_*`).

In the subroutine `user_map2d_get` only the variable `idir` needs to be modified to set the homogeneous direction. In `user_stat_trnsv`, only the pressure normalization might need to be changed.

## Compiling and executing

The KTH framework is compiled using a different recipe. The paths to NEk5000 and KTH_toolbox needs to be set in the `compile_script` file over variables `TOOLBOX_SRC` and `NEK_SOURCE_ROOT`. The variable `CASE also needs to be set.` An additional `makefile_usr.inc` is necessary to tell the compiler which objects to link (see examples).

**Step 1**: Compile the case.

```
$ ./compile_script --all
```

**Step 2**: If the compilation is successful, run the case. An example `cyl2d/run_nek.sh` is provided for cluster execution. The solver will not execute in a computer with less than 16 CPUs.

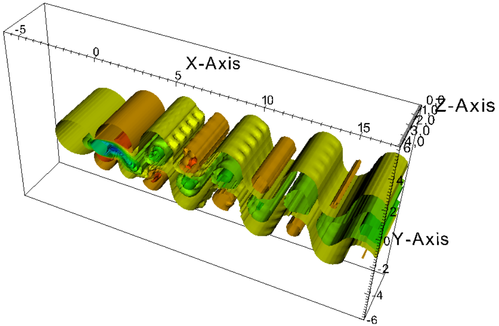
```
$ nekmpi cyl3d 24 >> cyl3d.out
$ tail -f cyl3d.out (In another terminal)
```

## Post-processing I

The case generates three types of files:

- Snapshots: `cyl3d0.f*` that can be read using ViSit

```
$ visnek
$ visit -o cyl3d.nek5000
```



## Post-processing II

- Statistics: `c2Dcyl3d0.f*` and `stscyl3d0.f*` files that are moved to the `cyl3d_PSTATD/pp_Nek/DATA` directory to be post-processed

```
$ mv c2Dcyl3d0.f* ../cyl3D_PSTAT2D/pp_Nek/DATA
$ mv stscyl3d0.f* ../cyl3D_PSTAT2D/pp_Nek/DATA
```

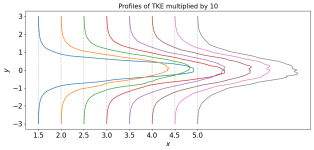
- Multi-step heckpoints: `rs6cyl3d0.f*` files able to restart the solution without numerical errors.

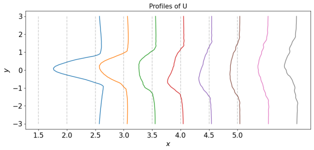
## cyl3d_PSTA2D directory I

The case generates three types of files:

- pp_Nek: Post-processing run of the cyl3D_STAT case. No time step iteration is accomplished, only reads and writes of files. The mesh must be a 2D slice of the 3D mesh.
- pp_Python: Generates the list of points for which the statistics are calculated and plots profiles.

For details, check the PSTAT2D module documentation.





## References

## For theory details:

- Deville, M.O. and P.F. Fischer and E.H. Mund. High-order methods for incompressible fluid flow. Cambridge University Press, 2002. doi.org/10.1017/CBO9780511546792
- KTH spring term Nek5000 course 2021

## For implementation details:

- Nek5000 documentation
- KTH framework for Nek5000 toolboxes documentation

## Other interface and post-processing tools:

- PyMech: A Python software suite for Nek5000 and SIMSON.
- NekMatLab: NEK utilities for MATLAB. A better version is provided in the repository.
- Snek5000: a Nek5000 interface in Python, requires a custom Nek5000 version.

## Acknowledgements

This work was funded by the São Paulo Research Foundation (FAPESP), grants nos. 2020/14200-5 and 2022/01424-8.

FAPESP
FUNDAÇÃO DE AMPARO À PESQUISA
DO ESTADO DE SÃO PAULO