# Numerical Techniques 2021–2022

# Practicum 1: Linux and Fortran basics

Daan Degrauwe

daan.degrauwe@meteo.be

Postgraduate Studies in Weather and Climate Modeling

Ghent University

# The problem at hand: the oscillation equation

- The oscillation is a differential equation given by:

$$\frac{\partial \psi}{\partial t} = i\kappa\psi$$

- The exact solution is a harmonic function with frequency $\kappa$:

$$\psi(t) = e^{i\kappa t}\psi(t_0) = (\cos(\kappa t) + i\sin(\kappa t))\,\psi(t_0)$$

- A simple numerical scheme is the *forward scheme*:

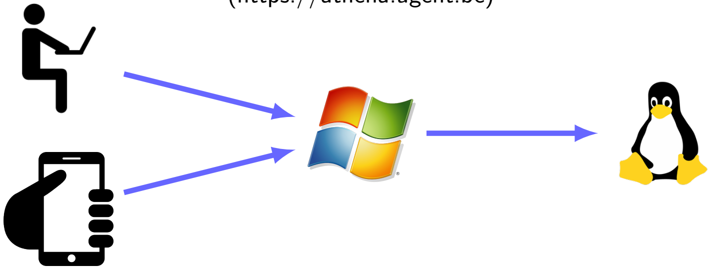
$$\frac{\phi^{t+\Delta t} - \phi^{t}}{\Delta t} = i\kappa\phi^{t}$$

or

$$\phi^{t+\Delta t} = (1 + i\kappa\Delta t)\phi^{t}$$

# Infrastructure: 3 machines

**Student** | **Athena** (https://athena.ugent.be) | **helios**



laptop, PC, smartphone, tablet:

- user input

Windows machine:

- Linux login (PuTTY)
- File editing (Notepad++)
- Postprocessing (RStudio)

Linux machine:

- compilation (gfortran)
- calculations (Fortran)

# Logging in

1. Create helios account on `https://helpdesk.ugent.be/account/en/helios.php`
2. Log in on Athena on `https://athena.ugent.be`
3. Open PuTTY on Athena
4. Set Host Name to *helios.ugent.be*
5. Click “Open” button

## Mounting H-drive

Do this every time you login on helios!

1. on helios, mount the H-drive with

   ```
   $ newns -i
   ```

2. go to the H-drive with

   ```
   $ cd /files/${USER}/home/
   ```

   (where you can substitute ${USER} by your user)

1. on helios, create an empty file with

```
$ mkdir /files/${USER}/home/numtech
$ cd /files/${USER}/home/numtech
$ touch test.txt
```

2. on Athena, open Notepad++
3. inside Notepad++, open the file `H:/numtech/test.txt`, and put some text in it
4. **Important**: set the end-of-line convention under
   Edit→EOL Conversion→Unix/OSX Format
   Do this always for files you will use in Linux!
5. Save the file in Notepad++
6. on helios, check the file content with

```
$ cat test.txt
```

# My first Fortran program

1. create a file osceq.F90:

```
$ mkdir /files/${USER}/home/numtech/osceq
$ cd /files/${USER}/home/numtech/osceq
$ touch osceq.F90
```

2. open the file in Notepad++, and put following code in it:

```
PROGRAM OSCEQ

! a program to solve the oscillation equation

WRITE (*,*) "Welcome to the Oscillation Equation Solver."

END PROGRAM OSCEQ
```

3. compile the program with

```
$ gfortran -o osceq osceq.F90
```

4. run the program with

```
$ ./osceq
```

# My first Linux script

1. create a file `run.sh`

   ```
   $ touch run.sh
   $ chmod +x run.sh
   ```

2. open the file in Notepad++, and put following code in it:

   ```
   # Script to compile and run the oscillation equation program

   # remove executable file
   rm osceq

   # Compile
   echo "Compiling"
   gfortran -o osceq osceq.F90

   # Run
   echo "Running"
   ./osceq
   ```

3. compiling and running are now done together with

   ```
   $ ./run.sh
   ```

# Fortran variables

1. It's good practice to take a copy before moving to the next assignment:

```
$ cp -r ../osceq/ ../osceq_v1
```

2. *Declare* variables with REAL, INTEGER, CHARACTER, . . . :

```
REAL ::  KAPPA
```

3. *Initialize* variables:

```
KAPPA = 0.5
```

4. *Use* variables in expressions:

```
WRITE (*,*) 'CFL = ',KAPPA*DT
```

1. Repetitive tasks (like a time integration) are implemented with Fortran loops:

```
DO IT=1,NT
    !...
ENDDO
```

2. Conditional branching is achieved with `IF`:

```
IF ( ABS(PHI) > 10.  )  THEN
    !...
ENDIF
```

Arrays contain a collection of numbers.

1. Declaration with ALLOCATABLE attribute and dimension indication:

```
COMPLEX, ALLOCATABLE ::  PSI(:)
```

2. Allocation (memory reservation) with ALLOCATE:

```
ALLOCATE(PSI(0:NT))
```

3. Reference to an element with ():

```
PSI(IT) = EXP(II*KAPPA*IT*DT)
```

4. Free memory with DEALLOCATE:

```
DEALLOCATE(PSI)
```

5. Be careful not to exceed the bounds of an array! Compile with bound checking for security:

```
$ gfortran -fbounds-check -o osceq osceq.F90
```

# Fortran subroutines

Use subroutines to organize your code.

1. Separate subroutines best go in separate files.
2. Arguments are declared with the INTENT attribute:

```
SUBROUTINE WRITE_RESULT(PSI,PHI)

REAL, INTENT(IN) ::  PSI(:)
REAL, INTENT(IN) ::  PHI(:)

!...

END SUBROUTINE WRITE_RESULT
```

3. Routines are called with CALL:

```
CALL WRITE_RESULTS(PSI,PHI)
```

Modules are useful to store global data (i.e. data not specific to a single subroutine).

1. Modules just contain the declarations:

```
MODULE CONSTANTS

INTEGER                :: NT       !  number of timesteps
REAL                   :: DT       !  timestep size

END MODULE CONSTANTS
```

2. Subroutines can access the variables from a module with the USE statement:

```
USE CONSTANTS
```

3. During compilation, make sure to put files containing modules first:

```
$ gfortran constants.F90 setup_constants.F90 timeloop.F90 \
        write_result.F90 osceq.F90 -o osceq
```

# Postprocessing in RStudio

1. On Athena, open RStudio
2. Load the results file `output.dat` with

   ```
   y=read.table('output.dat',header=TRUE)
   ```

3. Plot the exact solution with

   ```
   plot(y$time,y$exact,ylim=c(-3,3),type='b',pch=1,col=1)
   ```

4. add the numerical solution to the plot with

   ```
   points(y$time,y$numerical,type='b',pch=2,col=2)
   ```

5. put these commands in a file `show_results.R`, and run it in RStudio with

   ```
   source('show_results.R')
   ```

# Exercises on time schemes

1. Implement the backward and the trapezium schemes. What about their stability and phase error?

$$\phi^{n+1} = \frac{1}{1 - i\kappa\Delta t}\phi^n \qquad \text{(backward)}$$

$$\phi^{n+1} = \frac{1 + 0.5i\kappa\Delta t}{1 - 0.5i\kappa\Delta t}\phi^n \qquad \text{(trapezium)}$$

2. Implement the Matsuno scheme, and check the damping as a function of $\kappa\Delta t$. Is it accelerating or decelerating?

$$\tilde{\phi} = (1 + i\kappa\Delta t)\phi^n$$

$$\phi^{n+1} = \phi^n + i\kappa\Delta t\tilde{\phi}$$

3. Implement the Leapfrog scheme.

$$\phi^{n+1} = \phi^{n-1} + 2i\kappa\Delta t\phi^{n}$$

Use the forward scheme during the first timestep:

$$\phi^{1} = (1 + i\kappa\Delta t)\phi^{0}$$

4. Try to excite the computational mode in the Leapfrog scheme (by sabotaging the first timestep).

5. Implement the Robert-Asselin filter to damp the computational mode.

$$\begin{aligned} \phi^{n+1} &= \overline{\phi^{n-1}} + 2i\kappa\Delta t\phi^{n} \\ \overline{\phi^{n}} &= \phi^{n} + \gamma\left(\overline{\phi^{n-1}} - 2\phi^{n} + \phi^{n+1}\right) \end{aligned}$$