

CLIM0024

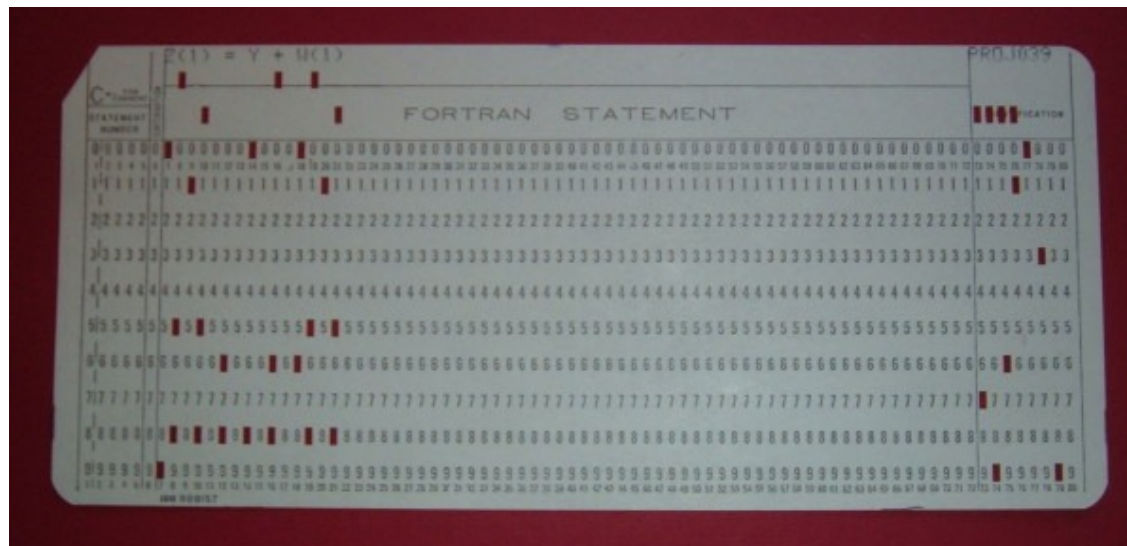
**Introduction to modelling in
physical geography
Partim 3 : Advanced climatology**

Fortran 2, Linux, Python

1.1 Introduction

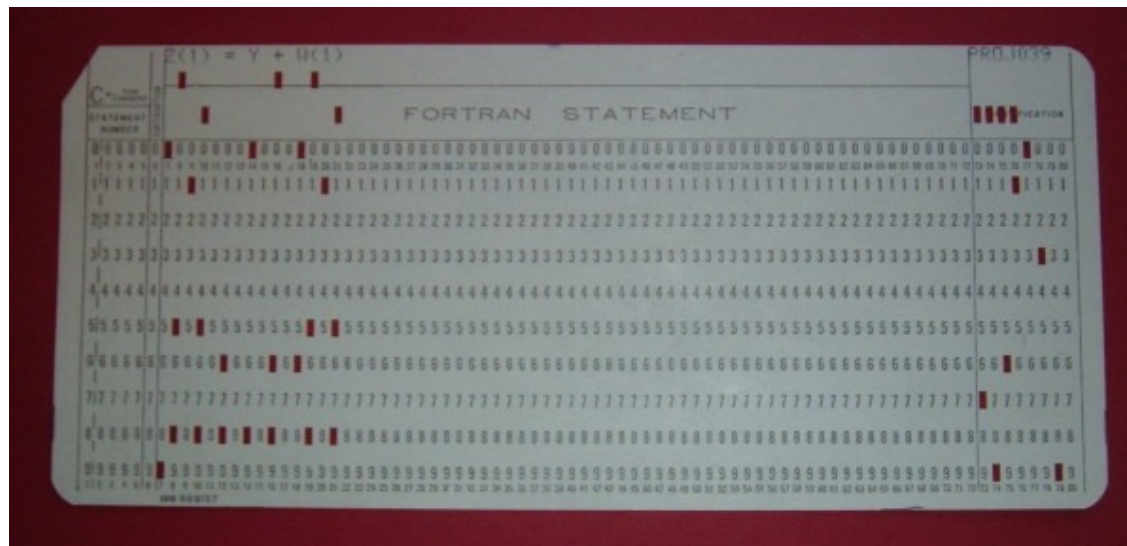
The aim of this lecture is teaching all the tools useful to be able to run the MAR model and afterwards to analyse/plot its NetCDF outputs:

- main UNIX commands and basics of the BASH language
- reading and writing Netcdf files with Fortran
- parallelisation with OpenMP in Fortran
- modules in Fortran
- introduction to Python used to build figures



1.2 Introduction

Evaluation: oral exam consisting of presenting a parallelised FORTRAN/BASH program comparing MAR model outputs (produced by the student) to gridded observations (e.g. [ECA&D](#)) + a discussion about the MAR ability to simulate the observed climate.



2.1 Main UNIX commands

pwd = *print working directory*
cd = *change directory*

ls = list
ls -l = list with more details (type `man ls` to have an explanation of options)
ls -oh = idem
ls -a = list the hidden files

mkdir = make a directory (ex: `mkdir dir1`)

. = current directory (ex: `ls .`)
.. = father directory (ex: `cd ../dir2`)

cp = copy a file (ex: `cp file.pdf file2.pdf`)
`cp file.pdf dir1`

cp -r = copy a directory (ex: `cp -r dir1 ../dir2`)
`cp -r dir1 ..`

mv = mv or rename a file/dir (ex: `mv file.pdf file2.pdf`)

rm (-f) = remove a file
rm -r = remove a directory (ex: `rm -rf /home/fettweis/dir1`)

ln -s = make a symbolic link (ex: `ln -s /home/fettweis/fichier.nc fichier2.nc`)

2.2 Main UNIX commands

d=directory
r=read
w=write
x=execute

date

```
fettweis@climato 15:50:27 ~ 1 >id
```

```
uid=1000(fettweis) gid=100(users) groups=100(users)
```

```
fettweis@climato 15:50:28 ~ 2 >ls -l
```

```
total 93692
```

size

drwxr-xr-x	2	fettweis	climato	4096	2023-09-12	22:27	BACKUP/
-rw-r--r--	1	fettweis	users	6777589	2023-09-07	13:48	Basins_1km.nc
drwx-----	2	fettweis	climato	4096	2020-06-19	08:35	Desktop/
lrwxrwxrwx	1	fettweis	climato	29	2021-01-03	10:49	Documents -> /srv8..
drwx-----	2	fettweis	climato	4096	2022-09-14	17:10	Downloads/
-rwxr-xr-x	1	fettweis	users	204	2023-05-09	10:29	ICE_check_all.bash*
-rwxr-xr-x	1	fettweis	users	1581	2023-05-09	10:29	ICE_check.bash*

group

user

user (u) – group (g) – other (o)

chmod = change mode

ex: `chmod go-rwx file.f90`

2.3 Main UNIX commands

echo	= print at the screen (ex: <code>echo "hello word"</code>)
du	= size of a dire/file (ex: <code>du -sh \$HOME</code>)
date	= date (ex: <code>date '+%Y %m %d %H:%M:%S'</code>)
cat	= print a file at the screen (ex: <code>cat file.f90</code>)
less	= print a file at the screen but “screen” by “screen” (ex: <code>less file.f90</code>)
head	= print the n 1 st lines (ex: <code>head -n 10 file.f90</code>)
tail	= print the n last line (ex: <code>tail -n 5 file.f90</code>)
grep	= print lines matching a pattern (ex: <code>grep -i "open" *.f90</code>)
find	= search for files (ex. <code>find . -name "*.f90"</code>)
wc	= word count (ex: <code>wc -l file.f90</code>)
sort	= sort a file (ex: <code>sort -n file.dat</code>)
sed	= “word” editor (ex: <code>sed "s/TOTO/TATA/g" file1.txt > file2.txt</code>)
awk	= pattern scanning and processing language (ex: <code>ls -l awk '{print \$5}'</code>)

2.4 Main UNIX commands

stdout/stderr



Exemple:

```
fettweis@climato 15:56:52 ~ 1 >ls MAR
bin/ doc/ frc/ git.bash* ice/ in@ lib/ msg/ out@ run/ sim/ src/ t
fettweis@climato 15:56:55 ~ 2 >ls WRF
/bin/ls: cannot access 'WRF': No such file or directory
```

2.5 Main UNIX commands

Pipes

Stdout from one command is used as stdin for the next one

Exemple:

```
ls -l | more
ls -l | tail -1
ls -l | wc -l
ls -l | awk '{print $5}' | sort -n
```

Diagram annotations:

- pipe (points to the first pipe symbol)
- Nbr of lines (points to the `wc -l` command)
- Print column #5 (points to the `{print $5}` command)
- Sort the numbers (points to the `sort -n` command)

Redirection into a file

```
ls -l > file.txt : stdout in file.txt
ls -l >> file.txt : stdout in file.txt (via an append)
ls -l 2> file.txt : stderr in file.txt
mkdir dir1 &>/dev/null : both in /dev/null (TRASH in Linux)
```

Kill a process

```
fettweis@climato 16:02:03 ~ 4 >ps ux | grep nedit
fettweis 18986  0.8  0.0  62920 10956 pts/21  SN   16:02   0:00  \_
nedit
fettweis@climato 16:02:09 ~ 5 >kill (-9) 18986 (or killall nedit)
```

3.1 BASH shell language

To build a BASH script:

```
nedit my_script.bash &
```

1. you need to write this at the first line

```
#!/bin/bash
```

...

2. make executable your script

```
chmod +x my_script.bash
```

To run your script (we don't need to compile BASH)

```
./my_script.bash arg1 arg2 arg3
```



```
#!/bin/bash
```

```
echo "$1" # print arg1  
echo "$3" # print arg3  
echo "$*" # print all arguments  
echo "$#" # print the nbr of arguments
```

3.2 BASH shell language

Variables in BASH

```
#!/bin/bash

var1=hello"
var1="${var1} word"
echo "${var1}"

var2=`uname -a`
echo "${var2}"
echo ${var2:20:9}    # print characters #20 to #29
echo ${#var2}        # print the nbr of characters
echo ${var2%SMP*}    # remove the characters after SMP
echo ${var2##*2008}  # remove the characters before 2023

var3=`pwd`
echo ${var3#*\ /}
echo ${var3##*\ /}
```

Variable declaration (without space before/after the =)

Print the variable \$var1 can be used but \${var1} is better

the ` put stdout from a command into a variable

for "special" characters like a space here



\$HOME, \$USER, \$PATH, ...
are used by Linux (type env)

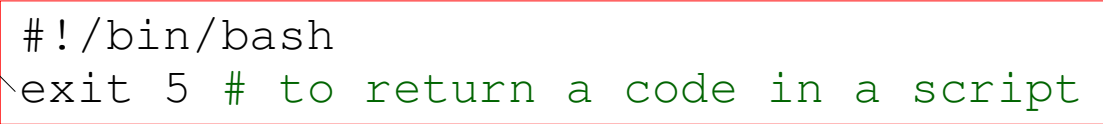
3.3 BASH shell language

Return code

The return code (**\$?**) can be used to check if a command has worked like:

```
ls -l file.f90 &>/dev/null
```

```
echo $?
```



```
#!/bin/bash
```

```
exit 5 # to return a code in a script
```

Make computation in integer

```
i=10 ; echo $(( $i + 1 ))  
j=$(( $i -6 ))  
echo $(( $i / $j )) # 2 in integer  
echo $(( 2013 % 4 )) # leap year ?
```

Read a File Line By Line

```
while read -r line  
do  
    echo "$line"  
done < "$input"
```

3.4 BASH shell language

If then else fi

```
file="programme.f90"
```

```
if [ -f "${file}" ] && [ ${#file} -gt 0 ] ; then
```

```
    echo "${file} is there"
```

```
else
```

```
    echo "${file} is not there"
```

```
fi
```

&& = AND ; **||** = OR

-f file : file existing?
! -d dir : dir non existing?
"\${var1}" == "\${var2}" : var1 = var2
"\${var1}" != "\${var2}" : var1 != var2
\${var1} -gt \${var2} : var1 > var2 (integer)
\${var1} -le \${var2} : <= (idem)

Loop

```
for j in `seq 2 9` ; do
```

```
    echo $(( $i + $j ))
```

```
done
```

```
for file in *.* ; do
```

```
    echo "${file}"
```

```
done
```

```
var="Les sanglots longs des violons de l'automne blessent ..."
```

```
for mot in ${var} ; do
```

```
    echo "${mot}"
```

```
done
```

3.5 BASH shell language

BASH and ferret:

```
#!/bin/bash
```

```
rm -f toto.dat
```

```
for file in ICE*.nc ; do
```

```
ferret<<EOF
```

```
use $file
```

```
set reg/i=10/j=10
```

```
list/nohead/file=toto.dat/append/format=(f8.2) TT[l=@ave,k=1]
```

```
quit
```

```
EOF
```

```
done
```

3.6 BASH shell language

1. Build a calendar with BASH:

set -x # to debug a script

DD=1 ; MM=1

while [\$MM -le 12] || [\$DD -le 31] ; do

DD=\$((\$DD + 1))

...

Done

2. With the help of generate.bash,

- rename the file to have file_099.dat instead of file_99.dat, file_9.dat
- compute the number of files
- compute the sum of positive integers in each file
- print the file name (its numbers) without negative numbers into
- find the largest files
- find the smallest files

4.1 Read/write NetCDF files

NetCDF (network Common Data Form)

In command line:

```
ncdump -h uwnd.sig995.2005.nc
```

```
netcdf uwnd.sig995.2005 {  
  dimensions:  
    lon = 144 ;  
    lat = 73 ;  
    time = UNLIMITED ; // (1460 currently)
```

```
  variables:  
    float lat(lat) ;  
      lat:units = "degrees_north" ;  
      lat:actual_range = 90.f, -90.f ;  
      lat:long_name = "Latitude" ;  
  ...
```

```
ncdump uwnd.sig995.2005.nc | less
```

```
netcdf uwnd.sig995.2005 {  
  ...  
  data:  
    lat = 90, 87.5, 85, 82.5, 80, 77.5, 75, 72.5, 70, 67.5, 65, 62.5, 60, 57.5,  
          55, 52.5, 50, 47.5, 45, 42.5, 40, 37.5, 35, 32.5, 30, 27.5, 25, 22.5, 20,  
          17.5, 15, 12.5, 10, 7.5, 5, 2.5, 0, -2.5, -5, -7.5, -10, -12.5, -15,  
          -17.5, -20, -22.5, -25, -27.5, -30, -32.5, -35, -37.5, -40, -42.5, -45,  
          -47.5, -50, -52.5, -55, -57.5, -60, -62.5, -65, -67.5, -70, -72.5, -75,  
          -77.5, -80, -82.5, -85, -87.5, -90 ;
```

NetCDF type : *ncdump -k*

Conversion : *nccopy -k 3 file.nc4 file.nc3*

nccopy -k 4 -d 2 -s f.nc3 f.nc4

dimension

variable name (as read by Fortran)

values

4.2 Read/write NetCDF files

Compilation: ifort libUN.f ex6_1.f90 -lnetcdf_ifort
 gfortran libUN.f ex6_1.f90 -lnetcdf -lnetcdf

```
integer, parameter :: dx=144
integer, parameter :: dy=73
integer, parameter :: dz=17
integer, parameter :: dt=365
integer i,j,k,t

real hgt(dx,dy,dz,dt),slp(dx,dy,dt),var2D(dx,dy),var3D(dx,dy,dz)

do t=1,dt ! To read/write a NetCDF, we need a loop on time
  call CF_READ2D("slp.2005.nc",'slp',t,dx,dy,1,var2D)

  call CF_READ3D("hgt.2005.nc",'hgt',t,dx,dy,dz,var3D)
  do i=1,dx; do j=1,dy

    slp(i,j,t) = var2D(i,dy-j+1) ! Because the y axis is inverted
    do k=1,dz
      hgt(i,j,k,t)= var3D(i,dy-j+1,dz-k+1)
    enddo
  enddo ; enddo
enddo
```

4.3 Read/write NetCDF files

Axis reading of the NetCDF files (useful to build a new file with the same grid)

```
integer, parameter :: dy=73  
integer, parameter :: dz=17  
integer i,j,k,l  
real lat(dy),level(dz)
```

```
call CF_READ2D('slp.2005.nc','lat',1,dy,1,1,lat)
```

```
call CF_READ2D('hgt.2005.nc','level',1,dz,1,1,level)
```

float lat(lat) ;

lat:units = "degrees_north" ;
lat:actual_range = 90.f, -90.f ;
lat:long_name = "Latitude" ;

float level(level) ;

level:units = "millibar" ;
level:actual_range = 1000.f, 10.f ;
level:long_name = "Level" ;
level:positive = "down" ;

"x-axis"



WARNING:
axis is 1D and only
the 1st dimension is read

4.4 Read/write NetCDF files

Writing a NetCDF file (1/2)

```
call CF_INI_FILE('toto.nc','my NetCDF file') ! Name and title of the file
```

“time” axis is needed even for a 2D file

```
call CF_CREATE_DIM("time","-",1,1) ! Axis: name, units, dimension, value
call CF_CREATE_DIM("lon","degree",dx,lon)
call CF_CREATE_DIM("lat","degree",dy,lat)
```

no z axis => “-”

```
call CF_CREATE_VAR("mean","Mean SLP","hPa","-", "lon","lat","-" )
! Variable: name, description, units, axis name (T,X,Y,Z)
```

```
call CF_CREATE_FILE('toto.nc')! building of the file
```

```
CALL CF_write ('toto.nc','mean',1,dx,dy,1,ave)
! writing of the file with the value of the table ave(dx,dy)
```

time step (1 if no time dimension)

4.5 Read/write NetCDF files

Writing a NetCDF file (2/2)

with a time dimension (here: monthly to yearly)

```
integer, parameter :: dt=720

real time(dt/12), ave(dx,dy)

do t=1,dt/12
  time(t)= real ((351      +(t+1947  -1902) *365      &
                  +(t+1947  -1901) / 4 + 366./2.) * 24 )
enddo

...

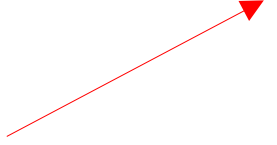
call CF_CREATE_DIM("time", "HOURS since 1901-01-15 00:00:00", dt/12, time)

...
do t=1,dt
  ave = xxx(t)
  CALL CF_write ('toto.nc', 'mean', t, dx, dy, 1, ave)
enddo
```

to be recognized by ferret



time step (a loop on time is needed !!)



5.1 Modules and allocate

Memory is limited to 2Go => allocate and modules

In f77 to share a variable between subroutines, we used common and include file.

Eg: MAR_DY.inc

```
...  
real          uairDY(mx,my,mz)  
...  
common/dynatm/ uairDY
```

static declaration

Common to share the variable

In f90, we can use module and allocate

Eg: mar_dy_mod.f90

```
module mar_dy  
  real, allocatable, save: uairDY(:, :, :)  
...  
subroutine/program  
  allocate(uairDY(mx, my, mz))  
...  
deallocate(uairDY)
```

save to share the variable

dynamics declaration



cleaning of memory

modules need to be compiled before (with ifort -c) or

```
module make_time_series_mod  
  implicit none  
  real, save, allocatable :: m_smb(:, :, :, :)  
end module
```

```
program make_time_series  
  use make_time_series_mod  
  allocate (m_smb...)
```

6.1 OpenMP and OpenMPI

- OpenMP :**
- allows parallel code to run on a single multi-core system
 - shared memory for each CPU
 - **non intrusive** in the code as most of the variable is common between CPUs
- OpenMPI**
- allows parallel code to run on multiple systems connected by a network
 - **allows to use more CPU than openMP**
 - each CPU has its own memory
 - very intrusive in the code as CPU needs to communicate variables

comment if not compiled with ifort -qopenmp

```
!$OMP PARALLEL DO default(shared)  
!$OMP private(i,j,...)  
!$OMP schedule(dynamic)  
  do i= ; do j =  
    ...  
  enddo ; enddo  
!$OMP END PARALLEL DO
```

variables are shared except the listed private ones

schedule could be dynamics if there is nothing to do on some pixels

firstprivate => variable can be initialized before the PARALLEL section

6.2 OpenMP and OpenMPI

```
sum=0
!$OMP PARALLEL DO reduction (+, sum)
!$OMP.private(i,j,..)
    do i= ; do j =
        sum=sum+..
    enddo ; enddo
!$OMP END PARALLEL DO
```

sum is "summed" after the PARALLEL section
reduction can be -,%,*,max and min

```
!$OMP threadprivate( /c_RADini/, my_var1,my_var2)
```

threadprivate can be used to put in private

- a common
- a list of variables from a module

Compilation: ifort -qopenmp

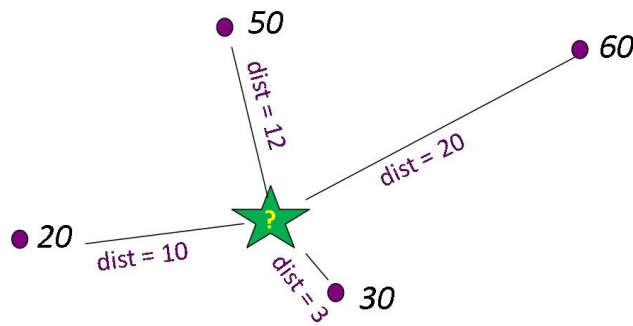
Execution: export OMP_NUM_THREADS=3 ; ulimit -s unlimited

7.1 Spatial interpolation

From **high** resolution to **low** resolution: average of “high” pixels which are in the “low” pixel

From **low** resolution to **high** resolution: sum of the 4 closest pixels weighted by the inverse of distance

Inverse Distance Weighting Interpolation



$$? = \sum Z_i / d_{ij}^n / \sum 1 / d_{ij}^n$$

$$? = (50/12) + (60/20) + (30/3) + (20/10) / (1/12 + 1/20 + 1/3 + 1/10)$$

$$? = 33.8$$

```
function distance(lon2t,lat2t,lon1t,lat1t)

implicit none

real,parameter :: R      = 6378.1370      ! km
real,parameter :: degrad = 3.141592/180. ! km
real :: lon1,lat1,dlat,dlon,a,c
real :: lon2,lat2,distance

real :: lon1t,lat1t
real :: lon2t,lat2t

lat2=lat2t*degrad
lon2=lon2t*degrad
lat1=lat1t*degrad
lon1=lon1t*degrad

dlat = (lat2-lat1)
dlon = (lon2-lon1)
a = sin(dlat/2.) * sin(dlat/2.) &
+ cos(lat1) * cos(lat2) * sin(dlon/2.) * sin(dlon/2.)
c = 2. * atan2(sqrt(a), sqrt(1.-a))
distance = max(1.0,R * c)

end function
```

8.1 Homework

1. On the basis of your MAR based daily outputs, evaluate the ability of MAR to simulate daily precipitation and temperature with respect to the [ERA5-Land](#).
 - A Temperature: build 2D maps of mean bias, daily correlation and RMSEc (divided by the daily ERA5-Land based daily variability) by considering all days (annual) and JJA only.
 - Precipitation: build 2D maps of mean bias (in %) of the cumulated precipitation at the annual and JJA scale.
2. On the basis of METAR observation, evaluate (bias, correlation, RMSEc) in at least 2 weather stations the daily temperature (at 2m), sea level pressure and wind speed (at 10m) from MAR and ERA5 with respect to <https://www.ogimet.com/gsynres.phtml.en/>