PyClimate: A Python Based Toolkit for the Analysis of Large Atmospheric and Oceanic Data Sets.

Jon Sáenz, Jesús Fernández and Juan Zubillaga

University of the Basque Country

28th February 2000

PYTHON Powered

eman ta zabal zazu

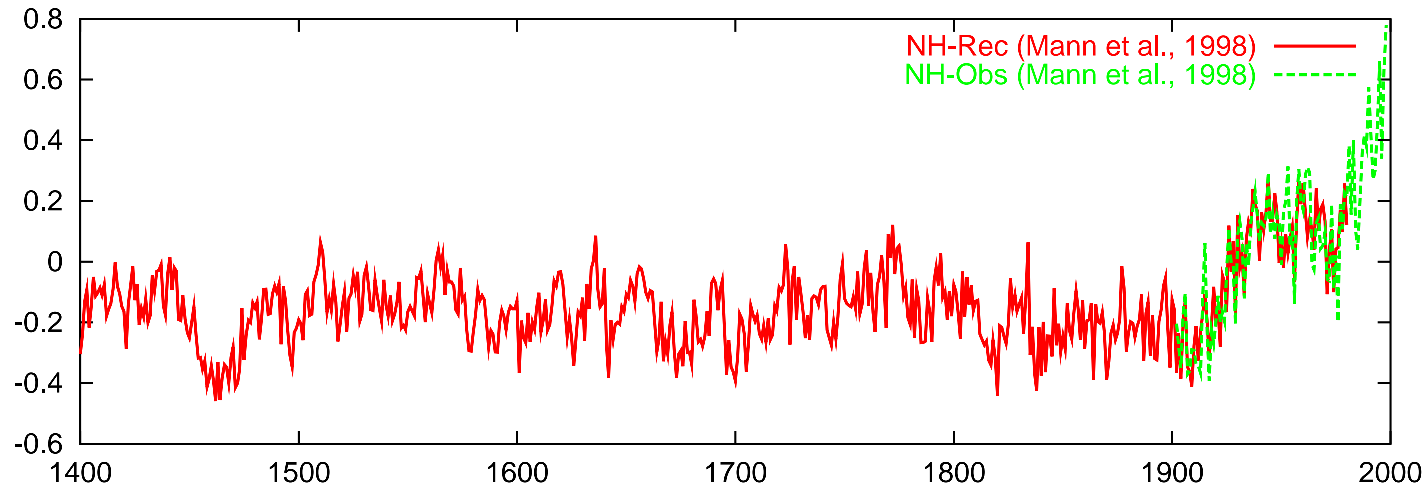
Contents:

- Detection and attribution of human–induced climate change
- A variety of large instrumental or modelling data sets
- A Python–based approach to data analysis
  1. Handling data sets under heterogeneous formats
  2. Heavy use of eigenvalue techniques
- Overview of PyClimate
- Examples of actual uses of PyClimate
- New challenging tasks

Global warming during the last Century

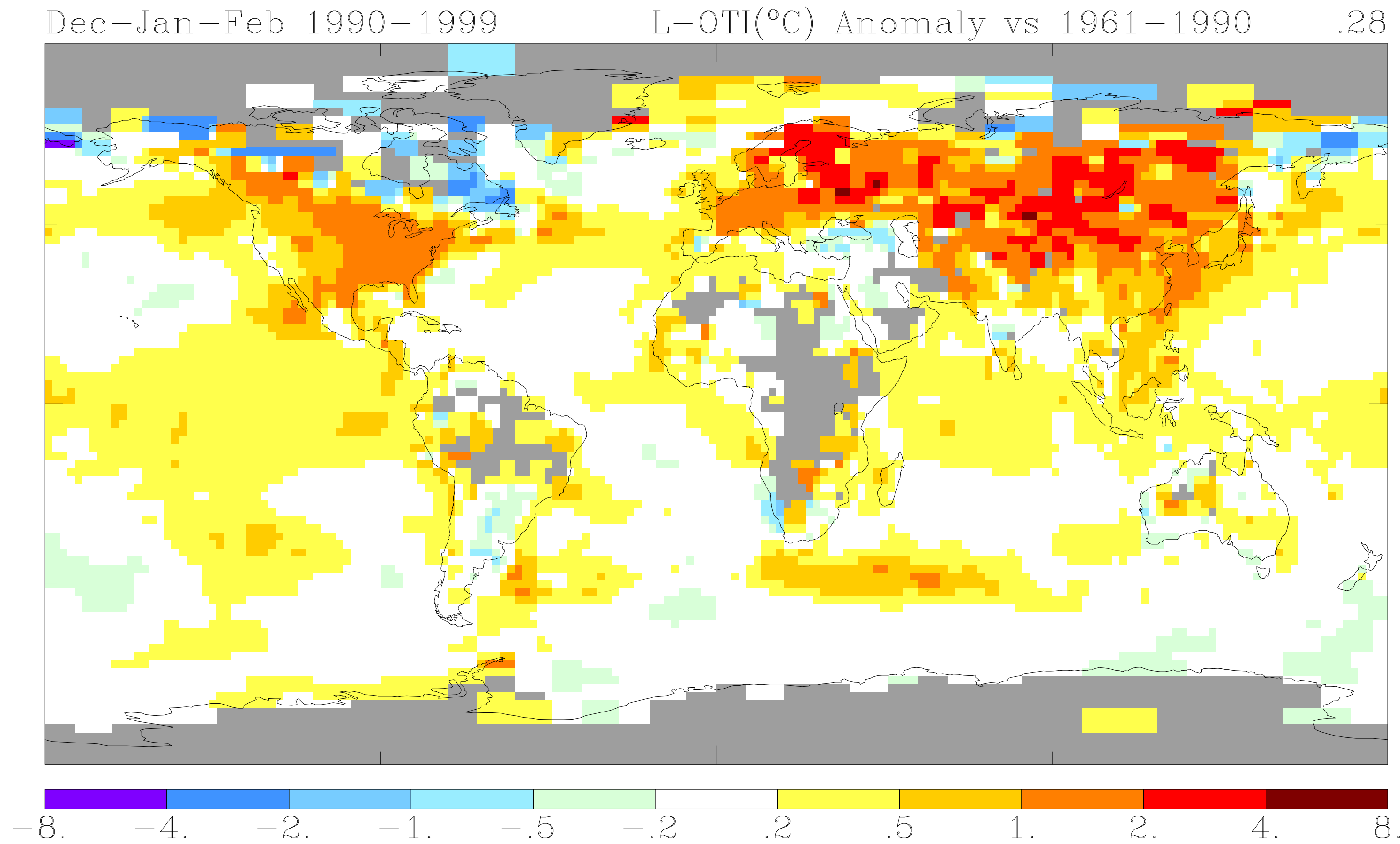
Multi–proxy approach to temperature change (Mann et al., 1998):

- Sediment cores
- Borehole temperatures
- Isotopic concentrations (^{18}O)
- Dendrocronology



Spatial and temporal variability of climate

Variations are not equal over the whole Earth (i.e. GISS temp). System is 3–D.



Coupled variability of different subsystems

Mutual influences between different subsystems of the climate system.

- Atmosphere – Ocean
- Ocean – Cryosphere
- Cryosphere – Atmosphere

Different scales of motion

Different subsystems show different scales of motion. Inside each of the subsystems, non–linear behaviour means a basically continuous spectrum. Different physical processes show different frequencies.

- Baroclinic perturbations
- Madden–Julian Oscillation
- Internal gravity waves versus Rossby waves
- Low–frequency variability
  - ENSO
  - NAO

Need to filter out spurious frequencies.

Variety of data sets

Different formats are very common:

- ASCII
- *Native* (low–endian or big–endian) binary
- netCDF
- GRIB
- HDF, . . .

It is difficult to access all of them from a low level programming language.

Huge data sets

Data sets are huge:

- NCEP–Reanalysis, 52 years, 6-hourly data, 1 vertical level, 1 variable (1.6Gb)
- Sea Surface Temperature, monthly data, 1860– (200 Mb)
- High resolution global model output, 10–ensemble winter runs, 9 vertical levels, 12 hourly data, 2 selected variables (2 Gb).

It is necessary to use a proper strategy to be able to handle this amount of data.

What is Python?

- Free and multiplatform *interpreted* object–oriented high level language with dynamic semantics, namespaces, named exceptions, ...
- Allows complex (built–in and user–defined) data types and supports Object Oriented Programming (polymorphism, operator overloading and multiple inheritance).
- Programs are easy to read, write and debug. Fast development cycle.
- Easy to learn. Very interesting for teaching students how to program.
- Easily extensible and embeddable. Full access to almost any external device, library and system call.

Python and scientific computing

- **NumPy** (Numerical Python) is an efficient C implementation of the tasks needed for scientific computing accesible from the high level **Python** language.
- **Python** is *interpreted*, but **NumPy** is compiled and performs fast for numerical tasks. Support of different numerical precisions (`Float32`, `Int8`, . . .).
- Support of general purpose libraries (LA, FFT, RNG, special functions).
- **Python** is extensible. New libraries can be created in C, C++, Fortran, compiled as shared libraries and integrated as packages into the interpreter.
- **Python** can be embedded into scientific applications to make them programmable.

Python versus C, C++, Fortran

- Execution time is somewhat shorter using F77, C, C++
- Development time is much shorter using Python
- Maintaining and upgrading Python software is easier

Therefore, there is a trade–off between execution time and developing time. Both times should be considered as a whole part of the life cycle of an application.

Python can be extended. Critical parts of the code can be written using F77, C,...

NumPy provides a complete framework for array–based operations.

Accessing data under GRIB format

```
def getfield(ifile,shape):
    datastr=ifile.read(shape[0]*shape[1]*4)
    return fromstring(datastr,Float32)

def conversion(file,avevar,varvar):
    os.system("wgrib %s|wgrib -i %s -bin>/dev/null"%(file,file))
    ifile=open("dump","r")
    for imon in xrange(12):
        avevar[orec]=getfield(ifile,avevar.shape[1:])
        varvar[orec]=getfield(ifile,varvar.shape[1:])

files=glob.glob("19*hgt500")
for f in files:
    conversion(f,avevar,varvar)
```

Replicating netCDF files (I)

```
# dimensions:
#   time = UNLIMITED ; // (624 currently)
#   lat = 73 ;
#   lon = 144 ;
# variables:
#   double time(time) ;
#       time:units ="hours since 1948-01-15";
#   float lat(lat) ;
#       lat:units = "degrees_North" ;
#   float lon(lon) ;
#       lon:units = "degrees_East" ;
#   float HGT(time, lat, lon) ;
#       HGT:missing_value = 9.999e+20f ;
# // global attributes:
#       :Conventions = "COARDS" ;
```

eman ta zabal zazu

Replicating netCDF files (II)

```
from pyclimate.ncstruct import *
from Scientific.IO.NetCDF import *
dims=("time","lat","lon") # Copy dimensions
var_str=dims+("HGT",) # Copy dims, attrs
var_all=dims[1:] # Copy dims, attrs, data
inc=NetCDFFile("ncepHGT500monthly.nc")
onc=nccopystruct("hgtDJF.nc",inc,
    dims,var_str,var_all)
```

Spatial and temporal variability

Let $T(\mathbf{r},t)$ be a scalar field. We are interested in its decomposition as $T(\mathbf{r},t) = \overline{T(\mathbf{r})} + \sum_{i=1}^{L} \alpha_l(t) T_l(\mathbf{r})$.

Karhunen–Loève expansion of T + discretization + truncation give the eigenvalue problem:

$$\sum_{l=1}^{L} \Gamma_{il} \Phi_k(\mathbf{r}_l) = \lambda_k \Phi_k(\mathbf{r}_i),$$

with $\Phi_k(\mathbf{r}_i) = T_k(\mathbf{r}_i)(\Delta A_i/A)^{\frac{1}{2}}$.

The time series α_l (PCs) and the eigenvalues $T_l(\mathbf{r})$ (EOFs) allow us to describe the temporal and spatial variability of the most important modes of the field.

EOF analysis by means of SVD (singular covariance matrix)

Let X be a (centered) $N \times P$ matrix of data ($P \gg N$) and $C = \frac{1}{N} X^T X$ the corresponding covariance matrix.

Let E be the matrix with orthogonal eigenvectors of the covariance matrix as columns; $CE = E\Lambda$, $\alpha = XE$, $X = \alpha E^T$, and Λ a diagonal matrix.

$$\text{Var}(\alpha) = \frac{1}{N}\alpha^T\alpha = \frac{1}{N}E^T X^T X E = E^T C E = E^T E \Lambda = \Lambda$$

Using the SVD decomposition of the data matrix $X = L\Sigma R^T$, then $C = \frac{1}{N} R\Sigma L^T L \Sigma R^T = \frac{1}{N} R\Sigma^2 R^T$. Thus, $CR = R\Lambda$ if $\Lambda = \frac{\Sigma^2}{N}$ and $R = E$. We also get $\alpha = XE = L\Sigma R^T E = L\Sigma$.

Linearly coupled variability of subsystems. SVD and CCA analysis

Let $X = [x_{i,j}]$ and $Y = [y_{i,k}]$ be centered data matrices corresponding to different subsytems of the climate system (e.g. SST and geopotential height) with $i = 1 \ldots N$ the samples, $j = 1 \ldots J$ the number of ocean grid points and $k = 1 \ldots K$ the number of atmospheric grid points.

$C_{xy} = \frac{X^T Y}{N} = U \sigma V$ gives left (U) and right (V) singular vectors which explain the maximum *covariance* between both fields.

Similarly, we can transform X and Y into their EOF space, scale them using the eigenvalues, so that $X = \alpha E^T = a e^T$ and $Y = \beta F^T = b f^T$, with $a_l = \alpha_l / \lambda_l$ and so on. Using the SVD of $C_{ab} = W \sigma^* Z$, we can get W and Z to obtain patterns which explain maximum *correlation* between both fields.

Overview of PyClimate

pyclimate is a **Python** package which performs several tasks useful for the analysis of climate variability. Currently, it offers:

- Functions to read ASCII data as Numerical arrays.
- Functions to replicate the structure of a NetCDF file.
- Functions and classes to handle monthly time steps.
- A **Python** interface to the library DCDFLIB.C
- EOF and SVD analysis.
- Multivariate digital filters. Kolmogorov–Zurbenko and Lanczos.
- Differential operators in spherical coordinates.
- Kernel–based probability density function estimation (univariate and multivariate).

Interface to DCDFLIB.C

pyclimate includes an interface to DCDFLIB.C, a C library for direct and inverse computations of parameters of several statistical distribution functions (β, binomial, negative binomial, χ^2, non–central χ^2, F, non–central F, Γ, Normal, Poisson, t, non–central t).
Original C prototype in DCDFLIB.C:

```
void cdfchi(int *which,double *p,double *q,double *x,double *df,
            int *status,double *bound);
```

Access to DCDFLIB.C from Python (I)

```
/*** Wrapper for cdfchi() ***/
typedef struct {
  int which;
  double p;
  double q;
  double x;
  double df;
  int status;
  double bound;
} CDFChi ;
extern int pycdfchi( CDFChi *sptr );
```

Access to DCDFLIB.C from Python (II)

```
from pyclimate.pydcdflib import *
chtab=[]
for i in xrange(3):
    chi2=CDFChi()
    chi2.which,chi2.p=2,0.9+i*0.025
    chtab.append(chi2)
for df in xrange(5,30):
    print "%6d"%(df,),
    for ch in chtab:
        ch.df=df
        pycdfchi(ch)
        print "%9.5f(%1d)"%(ch.x,ch.status),
    print
```

Eigenvalue techniques: EOF, SVD

pyclimate provides modules to perform:

- EOF analysis, getting the eigenvalues/eigenvectors and giving numerical estimations for truncation rules (Bartlett test and Monte Carlo analysis based on temporal subsampling). It is based on the SVD decomposition of the data matrix.
- SVD decomposition of the covariance matrix, obtaining the left and right singular vectors, homogeneous and heterogeneous correlation maps, the singular values, the covariance and squared covariance fractions and Monte Carlo test on the reliability of the singular vectors based on temporal subsampling.

Computation of EOFs (I)

```
import math
from Scientific.IO.NetCDF import *
from pyclimate.svdeofs import *
from pyclimate.ncstruct import *

inc = NetCDFFile("ncepslp.djf.nc")
slp = inc.variables["djfslp"]
lats = math.pi/180.*inc.variables["lat"][:]
areafactor = sqrt(cos(lats))
slpdata = (slp[:,:,:,:] *
           areafactor[NewAxis,NewAxis,:,NewAxis])
oldshape = slpdata.shape
slpdata.shape = (oldshape[0],
               oldshape[1]*oldshape[2]*oldshape[3])
```

Computation of EOFs (II)

```
pcs, lambdas, eofs = svdeofs(slpdata)
ds = ("lat", "lon")
onc = nccopystruct("firstEOF.nc", inc, ds, ds, ds)
eof1 = onc.createVariable("eof1", Float32, ds)
eof1.long_name = "First SLP EOF"
eof1[:,:] = ( reshape(eofs[:,0], oldshape[2:]) /
            areafactor[:,NewAxis] ).astype(Float32)
onc.close()
```

Input file size: 1 Mb
Alpha Workstation 700MHz 190MbRAM: 0.06 sec.
Laptop AMD K6 380MHz 64MbRAM: 0.55 sec.
Pentium III 450MHz 512MbRAM: 0.19 sec.

Differential operators over the sphere

They work over regular longitude×latitude non–periodic grids arranged as in the `COARDS` conventions.

- The horizontal component of the gradient of a scalar (i.e. geostrophic wind)
- The divergence of a two–dimensional vector field (i.e. the balance Evaporation minus Precipitation from the vertically integrated moisture transport)
- The vertical component of the curl of a vector field (i.e. vertical velocity at the bottom of the oceanic Ekman layer due to the wind stress)

Use of differential operators (I)

```
from pyclimate.diffoperators import *
from Scientific.IO.NetCDF import *
from pyclimate.ncstruct import *
inc = NetCDFFile("NCARreanal200.nc")
hgt = inc.variables["hgt"]
lats = inc.variables["lat"]
lons = inc.variables["lon"]
hgtdata = hgt[0,:,:]
grad = HGRADIENT(lats[:], lons[:])
geosfact = 1.486e-5 * sin(lats[:] * 0.017453)
geosfact = geosfact[:,NewAxis]*ones(hgtdata.shape)
geosfact = geosfact + equal(geosfact, 0.0)
hg = grad.hgradient(hgtdata)
wind = (-hg[1]/geosfact, hg[0]/geosfact)
```

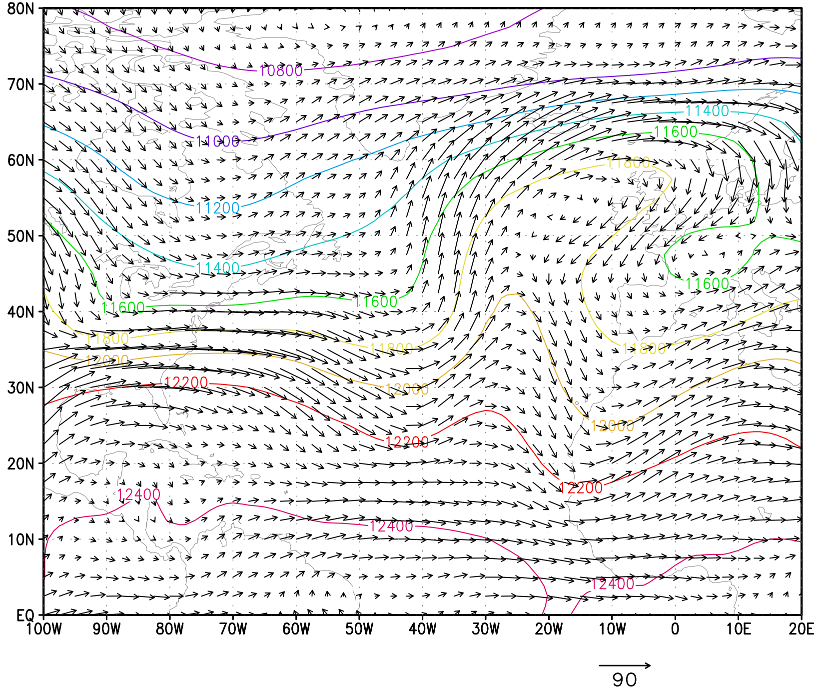
Use of differential operators (II)

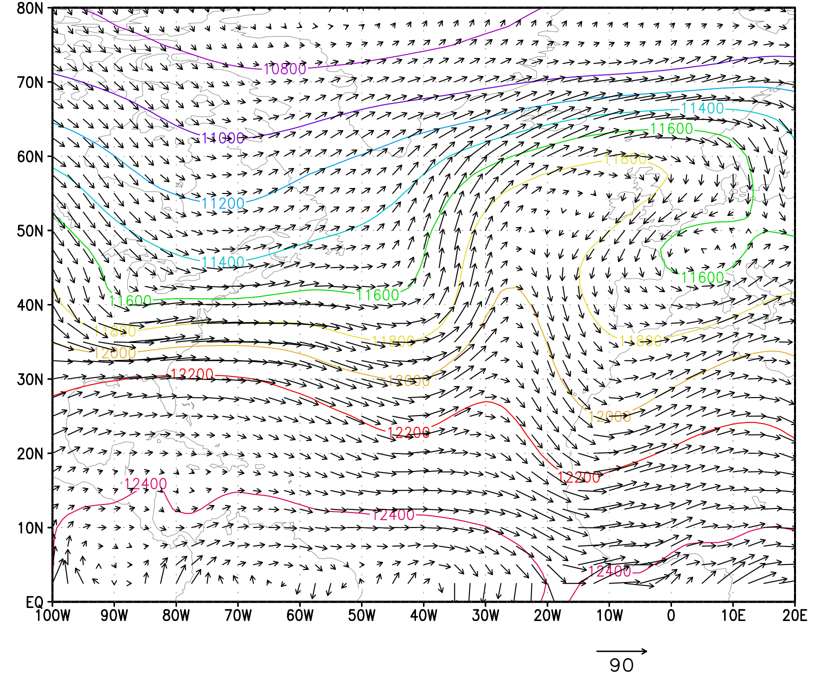
```
ds = hgt.dimensions
onc = nccopystruct("gwind200.nc", inc, ds, ds, ds)
onc.title = "Geostrophic wind"
gw_u = onc.createVariable("gw_u", Float32, ds)
gw_u.units = "m/s"
gw_v = onc.createVariable("gw_v", Float32, ds)
gw_v.units = "m/s"
gw_u[0,:,:] = wind[0].astype(Float32)
gw_v[0,:,:] = wind[1].astype(Float32)
onc.close()
```

Input file size: 20Kb	40Mb
Alpha Workstat. 700MHz 190MbRAM: 0.02 sec.	2’30”
Laptop AMD K6 380MHz 64MbRAM: 0.3 sec.	3’18”
Pentium III 450MHz 512MbRAM: 0.25 sec.	2’23”

http://www.pyclimate.org

Observed versus computed winds



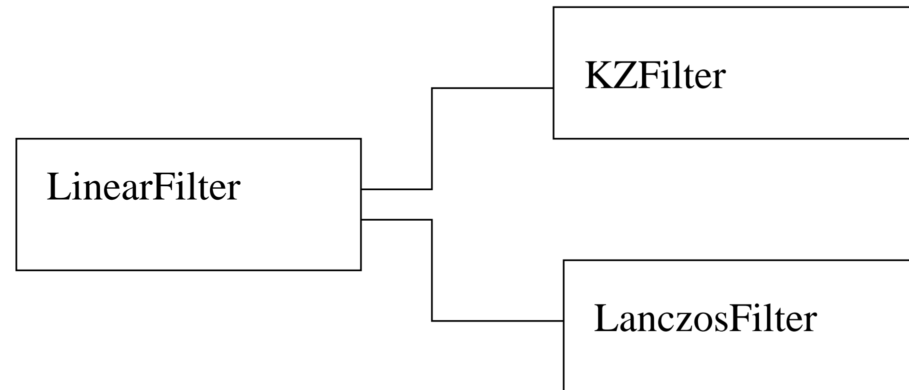


Linear filters for huge data sets

Given an input data set X_t, a general linear filter involves the operation:

$$Y_t = \sum_{i=-n}^{n} a_i X_{t+i}.$$

This basic operation has been implemented as a base class which derives different kinds of filters by means of the constructors (different a_k coefficients). Thus, the basic filter can be easily extended by redefining these coefficients.



Use of linear filters (I)

```
import math
from pyclimate.LanczosFilter import *
twopi = 2. * math.pi
def f(x):
  return sin(twopi * 0.01 * x)+0.75 * sin(twopi * 0.05 * x) +\
           0.50 * sin(twopi * 0.20 * x)
a = f(arrayrange(1000))  # [f(0) f(1) ... f(999)]
a.shape = (1000,1)
npoints = 50
lp = LanczosFilter('lp', 0.025, 0.025, npoints)
bp = LanczosFilter('bp', 0.025, 0.08, npoints)
hp = LanczosFilter('hp', 0.18, 0.18, npoints)
```

Use of linear filters

```
for i in range(len(a)):
  lfa = lp.getfiltered(a[i])
  bfa = bp.getfiltered(a[i])
  hfa = hp.getfiltered(a[i])
  if lfa:
    print "%d %f %f %f"%(
            i-npoints, lfa[0], bfa[0], hfa[0])
```

Filtering of 1000 points:

Alpha Workstation 700MHz 190MbRAM: 0.5 sec.

Laptop AMD K6 380MHz 64MbRAM: 2.8 sec.

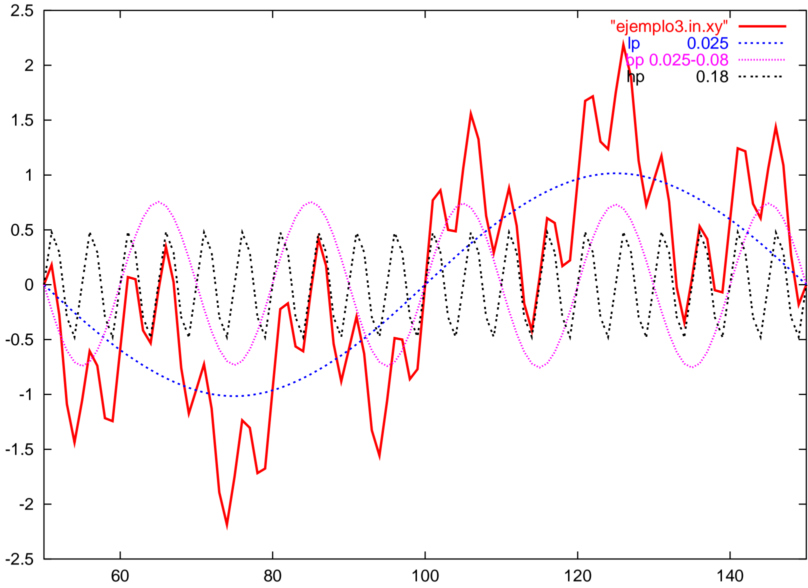
Pentium III 450MHz 512MbRAM: 0.72 sec.

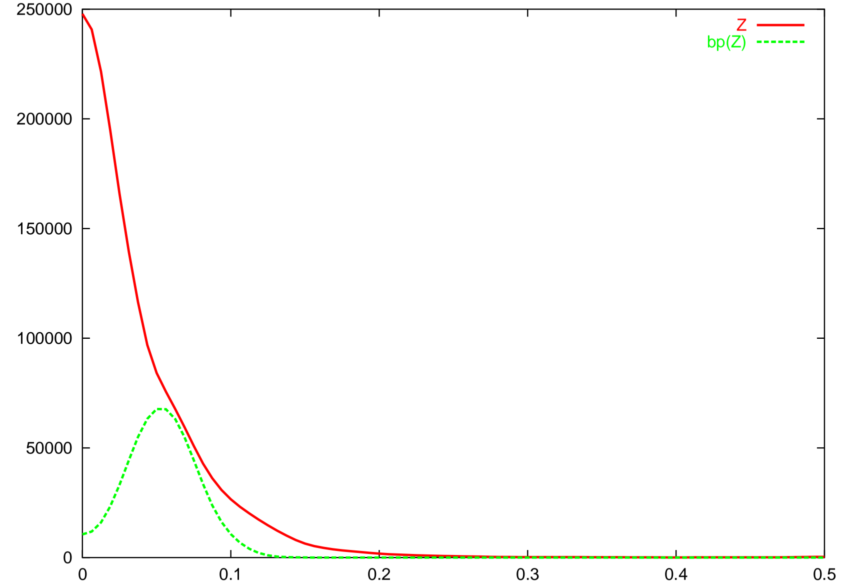
PYTHON Powered

http://www.pyclimate.org

eman ta zabal zazu

Synthetic and observed Z_{500} filtered data (2.5–10 days)





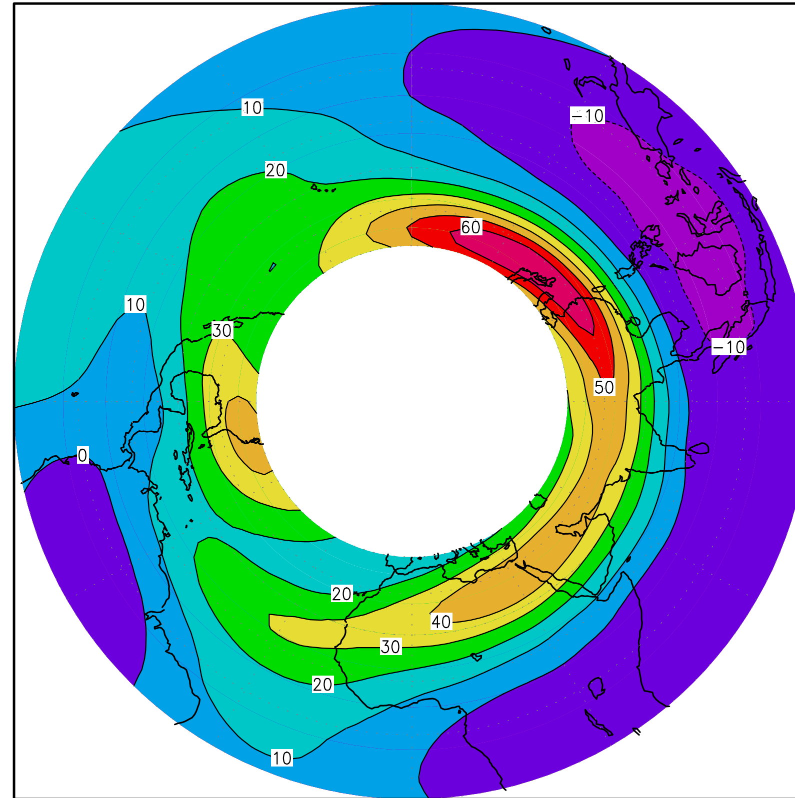
Outgoing Longwave Radiation and the NAO

- Is tropical forcing in the Atlantic to trigger an atmospheric response? (Dong et al., 2000)
- In the Atlantic basin, nonlinear wave interactions are more important than tropical forcing (Bladé, 1999; Hansen et al., 1997)
- Theoretical basis

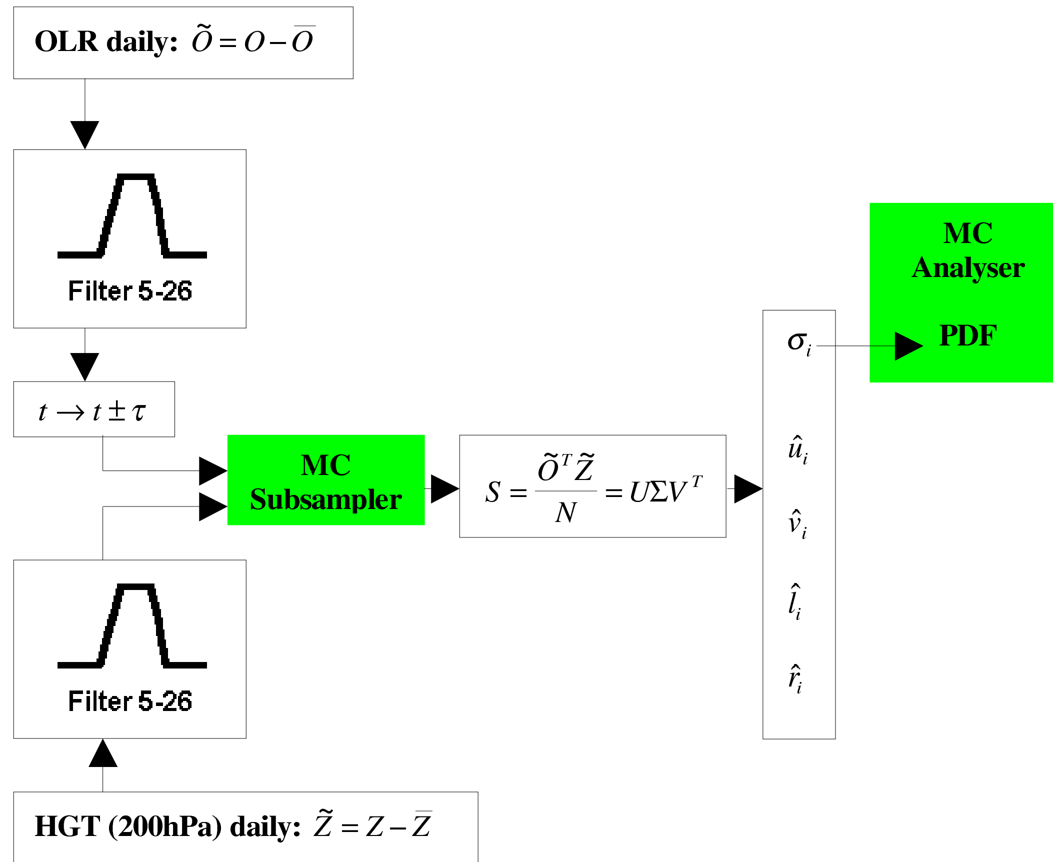
1. Rossby source term: $S = -\nabla \cdot (\mathbf{v}_\chi \cdot \zeta)$ (SH, 1988)
2. Convective anomalies over the Caribbean are able to produce euroatlantic blockings (FMP, 1994)
3. Rossby wave propagation depends on the zonal speed at high levels (Hendon et al., 2000)
4. OLR is a good proxy for tropical convection (divergence) and it is difficult to evaluate properly S in the Tropics (HZG98; LMGKWM98)

OLR–NAO

DJ Zonal wind speed at 200 hPa

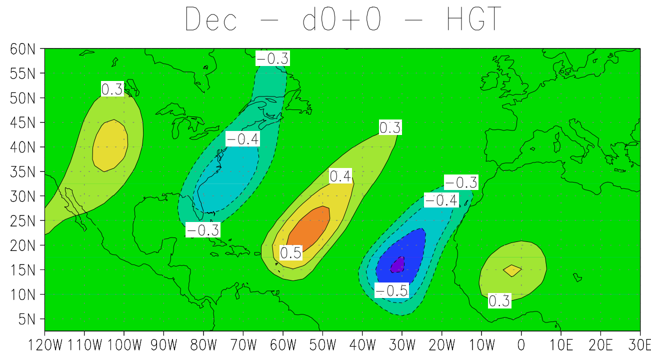


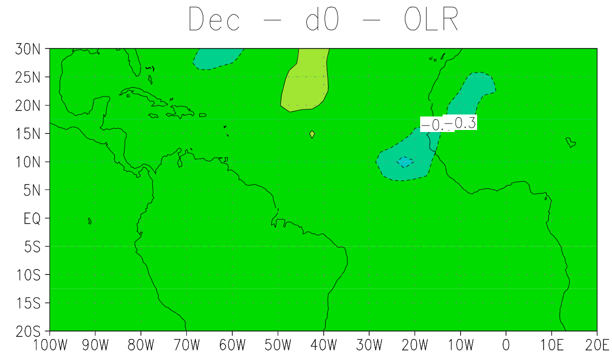
Detection method

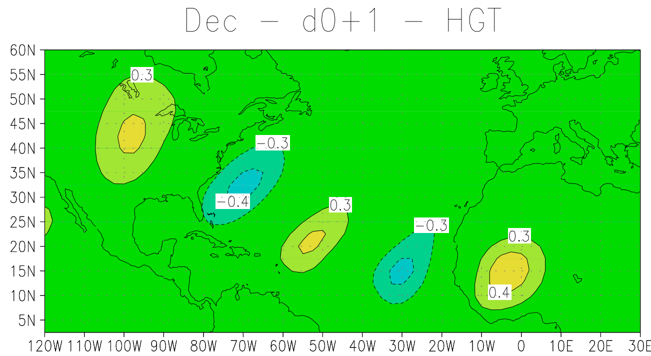


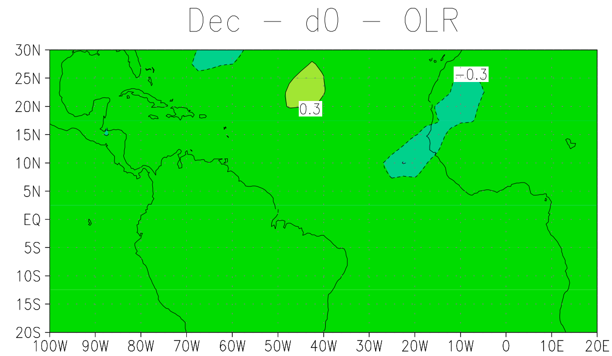
OLR–NAO

Rossby waves propagate towards the Tropics from the extratropics

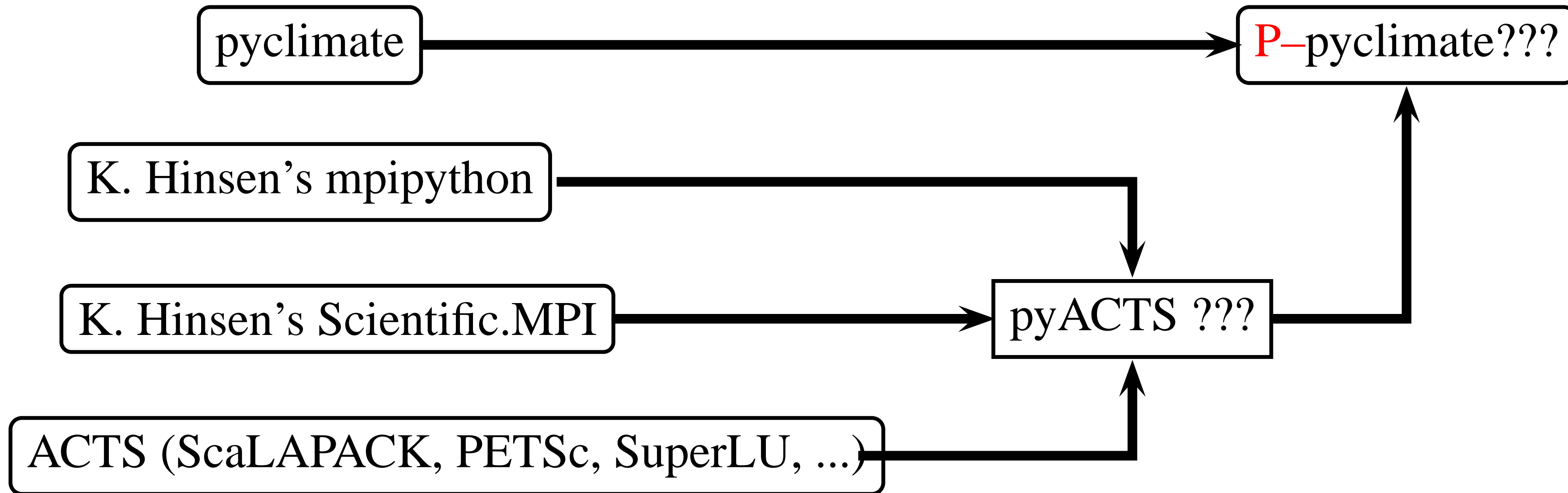








What next?



Conclusions

- **Python** is a powerful free multiplatform high level language.
- **Python** is extensible and has a fast Numerical Array extension.
- Those features allow fast development of code for data analysis.
- A set of routines has been developed to perform some frequent tasks during the analysis of climate data sets
- Those routines make easier fast data analysis of big data sets.
- It is interesting to get this computing paradigm working under parallel architectures, as well, to achieve better performance..

LINKS

- **http://www.python.org**
- **http://numpy.sourceforge.org**
- **http://www.pyclimate.org**