TD: analyse BLUE

Script à télécharger: ana2d.py

- analyse en 1D ou 2D, données synthétiques
- algorithme Cressman ou 3DVar
- code non optimisé, facile à lire séquentiellement
- plots dans fichiers images .png

Installation: (logiciels gratuits portables)

- python v2.7
- modules python:
  - matplotlib : pour les graphiques
  - scipy : pour l'optimiseur variationnel
- si pas installés:
  - installer la distribution Anaconda de python
  - ou utiliser easy_install, pip, etc pour installer à la main
- testé sur MacOSX et Linux

```
% ./ana2d.py
```

ana2d.py : paramètres (1)

```
#!/usr/bin/python
#   Simple 2D univariate analysis - F.Bouttier Dec 2015

import random,math,sys,os
import numpy as np
import matplotlib.pyplot as plt
import subprocess
from scipy.optimize import minimize
epsilon=1.e-5

######### user parameters ##############
### define model state
ni=1 ; nj=51                          # model grid size
imin=0. ; imax=0. ; jmin=0. ; jmax=10.    # geographical model boundaries

### define obs network
#obsinit='onemiddle'        # obs initialization method
obsinit='twomiddle'        # obs initialization method
```

ana2d.py : paramètres(2)

```
### define background
bgtype='zero'           # background type: zeroes everywhere
### define background error model
sigmab=1.             # B uniform background standard error
bdist=2.5             # characteristic distance for bg error correlations
#bcortype='gaussian'       # gaussian error correlation
bcortype='triangle'       # triangular error correlation

### define analysis method
#anamethod='cressman' ; minweight=0.1       # Cressman method
anamethod='3dvar'    ; niter=2             # variational

### plotting params
PLOTFILE='ana1dplot.png'
PLOT=1                    # 1 to generate plot
VIEW=1                    # 1 to view plotfile on screen
VIEWER='gqview 2>/dev/null'  # shell command to display png file on screen
# Mac OSX compatibility
osname=subprocess.Popen('uname',stdout=subprocess.PIPE).stdout.read()
if osname=='Darwin\n' : VIEWER='open'

### technical params
verb=1                    # 1 for verbose output
```

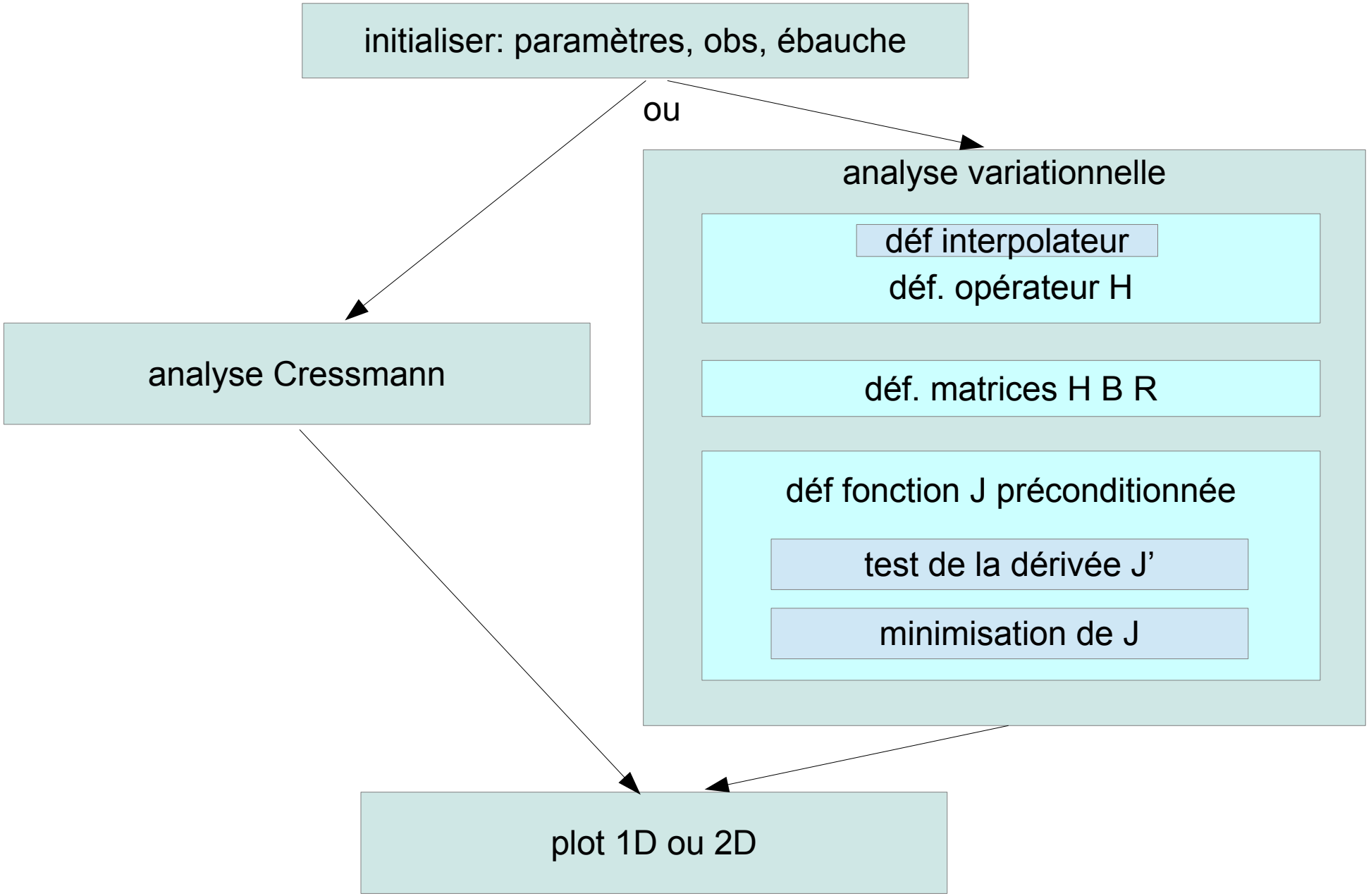
ana2d.py : def observations

```
########################## setup obs ######################

if obsinit=='onemiddle':        # obs = 1 in middle of domain
  nobs=1
  y =[ 1. ]
  yi=[ (imin+imax)/2. ]
  yj=[ (jmin+jmax)/2. ]

if obsinit=='twomiddle':        # obs = 1 in middle of domain
  nobs=2
  y =[ 0.6 , 1. ]
  yi=[ 0.6*imin + 0.4*imax , 0.4*imin + 0.6*imax  ]
  yj=[ 0.6*jmin + 0.4*jmax , 0.4*jmin + 0.6*jmax  ]

errobs=1.                  # R obs standard error
```

ana2d.py : def ébauche

```
### setup background state #####
if bgtype=='zero':
  print 'xb=zero everywhere'
  xb = np.zeros( (ni,nj) )

### setup background error covs ###
def bcov(ia,ja,ib,jb):   # compute bg error cov between points (ia,ja) and (ib,jb)
  d=math.sqrt(  (ia-ib)**2 + (ja-jb)**2 )

  if bcortype=='triangle':
    c = max( 0. , 1. - d / bdist )          # linear triangular correl

  elif bcortype=='gaussian':
    c = math.exp( - d**2 / ( 2. * bdist**2 ) )   # gaussian correl

  c = c * sigmab * sigmab        # convert correlation to covariance
  return(c)
```

ana2d.py : structure du code



ana2d.py : analyse Cressman

```
inc = np.zeros( (ni,nj) )

if anamethod=='cressman':
  di=0. ; dj=0.
  if ni>1 : di=(imax-imin)/float(ni-1)      # precompute model resolution
  if nj>1 : dj=(jmax-jmin)/float(nj-1)
  for i in range(ni):
    for j in range(nj):
      zi=imin+i*di       # real coords of gridpoint(i,j)
      zj=jmin+j*dj

      ## do the following for all model gridpoints ##
      zinc=0. ; zsum=0.
      for k in range(nobs):
        if verb: print '    obs idx=',k
        zw=bcov( zi,zj , yi[k],yj[k] )    # weight between model & obs points
        zinc = zinc + zw * dep[k]
        zsum = zsum + zw

      if zsum>minweight :
        inc[i,j] = zinc / zsum
        if verb: print 'Cressman ana at i,j=',i,j,' zi,zj=',zi,zj,' zsum,inc=',zsum,inc[i,j]

xa = xb + inc
```

ana2d.py : opérateur d'obs

```
def nearestgp(i,j):      # compute nearest gridpoint from real coords (i,j)
  kio=int( float(ni)*(i-imin)/(imax-imin+epsilon) + 0.5 )
  kjo=int( float(nj)*(j-jmin)/(jmax-jmin+epsilon) + 0.5 )
  assert( (kio>=0) and (kio<=ni) ),'out-of-domain obs i-coord kio='+repr(kio)
  assert( (kjo>=0) and (kjo<=nj) ),'out-of-domain obs j-coord kjo='+repr(kjo)
  return(kio,kjo)

def hobsop(x,i,j):    # interpolate state x at real coords (i,j)
  if verb: print '  obsop: i,imin,imax=',i,imin,imax
  if verb: print '  obsop: j,jmin,jmax=',j,jmin,jmax
  (kio,kjo)=nearestgp(i,j)
  if verb: print '  obsop: nearest gridpoint (i,j)=',kio,kjo
  hx=x[kio,kjo]
  if verb: print '  obsop: kio,kjo,h(x)=',kio,kjo,hx
  return( hx )        # interpolate using nearest-neighbour method

#### compute innovations i.e. obs-model departures ###
dep=[None]*nobs
for k in range(nobs):
  hxb=hobsop( xb , yi[k] , yj[k] )
  dep[k] = y[k] - hxb
```

ana2d.py : plot 1D

```
### plot the analysis (1D only) ###
if PLOT and ni==1 :

  # matplotlib y=f(x) plot
  fig,ax = plt.subplots(1)
  ax.set_ylim([-0.1,1.1] )
  ax.set_title('1D analysis')
  ax.set_xlabel('space coord')
  ax.set_ylabel('field value')
  ax.grid(True)
  jcoords=np.linspace(jmin,jmax,nj)   # list of i-coordinates in x

  # plot xa(j) as curve
  plt.plot( jcoords,xa[0,:] ,'o-',linewidth=5,color='cyan')

  # plot observations
  plt.plot( yj,y ,marker=r'$\asterisk$',linestyle='', \
            color='green',markeredgecolor=None,markersize=40)

  # plot backend: make PNG file
  fig.set_size_inches(20, 10)
  fig.savefig(PLOTFILE,bbox_inches='tight')
  os.system('ls -l '+PLOTFILE)
  if VIEW: os.system(VIEWER+' '+PLOTFILE)
```

ana2d.py : BLUE variationnel(1)

```
### define mapping between 2D (i,j) fields and control variables ###
  ncv=ni*nj
  print '3DVar: size of control variable=',ncv

  def idx_ij2cv(i,j):    # convert indices: 2D -> CV
    return( i*nj + j )

  def idx_cv2ij(k):      # convert indices: CV -> 2D
    i = int( float(k)/float(nj) )
    j = k - i*nj
    return(i,j)

  def vec_cv2ij(xcv):    # convert vector: CV -> 2D
    x2d = np.zeros( (ni,nj) )
    for k in range(ncv):
      (i,j) = idx_cv2ij(k)
      x2d[i,j] = xcv[k]
    return(x2d)

  def vec_ij2cv(x2d):     # convert vector: 2D -> CV
    xcv=np.zeros(ncv)
    for k in range(ncv):
      (i,j) = idx_cv2ij(k)
      xcv[k] = x2d[i,j]
    return(xcv)
```

ana2d.py :variationnel: def H, B, R, y-H(x)

```
#### fill explicit H ###
hmatrix = np.zeros( (nobs,ncv) )
for k in range(nobs):
  (io,jo)=nearestgp( yi[k],yj[k] )
  hmatrix[ k , idx_ij2cv(io,jo) ] = 1.          # interpolation coeff for obs k

# fill explicit B
bmatrix = np.zeros( (ncv,ncv) )
for k1 in range(ncv):
  for k2 in range(ncv):
    (i1,j1)=idx_cv2ij(k1)
    (i2,j2)=idx_cv2ij(k2)
    bmatrix[k1,k2] = bcov(i1,j1,i2,j2)

# define inverse of R (diagonal)
ri = 1./(errobs*errobs)

# departure computation from non-preconditioned state vector
def calcdep(xcv):
  x2d=vec_cv2ij(xcv)
  dep=np.zeros( (nobs) )
  for k in range(nobs):
    hx=hobsop( x2d , yi[k] , yj[k] )
    dep[k] = y[k] - hx
  return(dep)
```

ana2d.py : explication du préconditionnement

Pour éviter d'inverser/factoriser B, on fait le **changement de variable**:
$u = B^{-1}(x-x_b)$

$x = x_b + B\,u$

$d = (y - H\,x) = (y - H[\, x_b + B\,u\,])$

$J(x) = (x - x_b)^T B^{-1}(x - x_b) + (y - Hx)^T R^{-1} (y - Hx)$

$\quad = u^T B\,u \qquad\qquad + d^T R^{-1} d$

$J'(u) = 2\,B\,u + 2BH^TR^{-1}\,d$

on initialise la minim avec $u = B^{-1} (x - x_b) = 0$

on minimise J par rapport à u

et finalement:
$x_{min} = xb + B\,u_{min}$

Alternative: préconditionner par L tel que $B=LL^T$
et alors $J(x) = u^Tu + d^TR^{-1}d$: plus sphérique, mais il faut définir $L=B^{1/2}$

ana2d.py : variationnel: def fonction-coût J

```
def jcost(u):      # u is a B-preconditioned vector in control variable space
  bu = np.dot( bmatrix , u )
  jb = np.dot( np.transpose(u) , bu )          # Jb = u^T B u
  gradjb = 2. * bu                             # Jb' = 2 B u

  #on inverse le préconditionnement pour calculer H
  x = vec_ij2cv(xb) + np.dot( bmatrix , u )    # x = xb + B u
  dep = calcdep(x)                             # d = y - Hx

  jo = np.dot( np.transpose(dep) , ri*dep)    # Jo = (y-Hx)^T R-1 (y-Hx)
  gradjo = -2. * np.dot( np.transpose( hmatrix ) , ri * dep )   #dJo/dx
  gradjo = np.dot( bmatrix , gradjo )          # dJo/du = dJo/dx.dx/du

  j = jb + jo
  gradj = gradjb + gradjo
  return( j , gradj )

# test calcul de J(xb)
ub=np.zeros(ncv)                               # u = B(x-xb)
(jub,gradjub) = jcost( ub )
print ' xb -> J=',jub,' normgradJ=',np.dot( gradjub , gradjub )
```

impression:

```
 xb -> J= 1.36  normgradJ= 9.792
       gradJ= [   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
                  0.    0.    0.    0.    0.   -0.24 -0.72 -1.2  -0.72 -0.24  0.    0.    0.
                  0.    0.    0.   -0.4  -1.2  -2.   -1.2  -0.4   0.    0.    0.    0.    0.
                  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
```

ana2d.py : variationnel: test de la dérivée J'

```
# test correctness of the gradient (Taylor formula)
print 'gradient test:  a        deltaJ        gradJ.h       ratio'
for k in range(-1,-8,-1):
  a = 10.**k
  h = a*gradjub              # perturbation h
  (ju,gradju) = jcost( ub+h  )       # J( ub + h )
  varlin= np.dot( gradjub , h )   # linear variation of J predicted by gradjub
  print '       %10.2g  %12.2g  %12.2g  %15.8f' % (a, ju-jub, varlin , varlin/( ju-jub+1.e-10) )
```

```
gradient test:  a        deltaJ        gradJ.h       ratio
             0.1           1.3          0.98          0.78125000
            0.01           0.1         0.098          0.97276264
           0.001        0.0098        0.0098          0.99720781
          0.0001       0.00098       0.00098          0.99971998
           1e-05       9.8e-05       9.8e-05          0.99997098
           1e-06       9.8e-06       9.8e-06          0.99998699
           1e-07       9.8e-07       9.8e-07          0.99989761
```

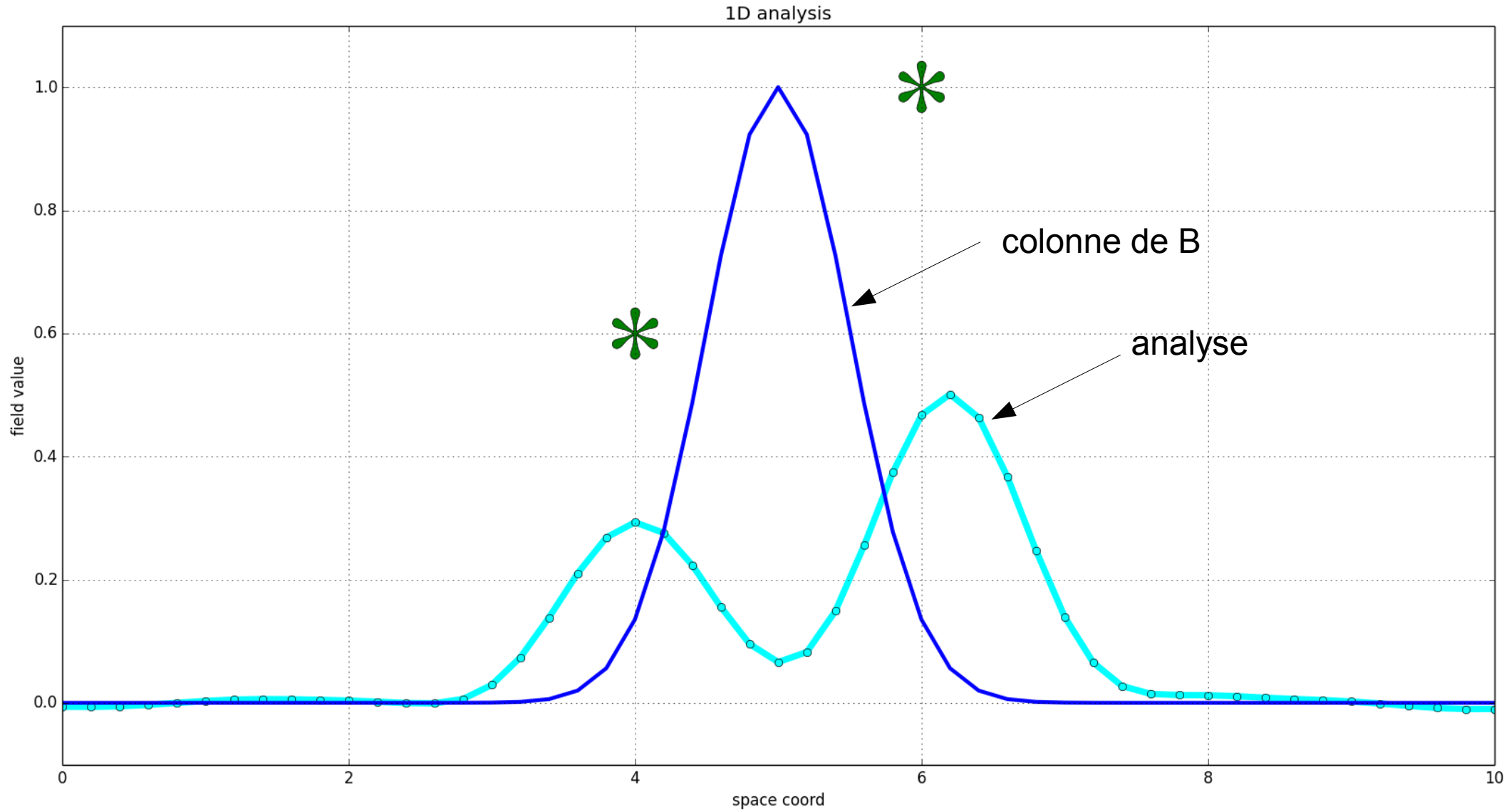
ana2d.py : variationnel: minimisation

méthode BFGS Broyden-Fletcher-Goldfarb-Shanno: sorte de gradient conjugué
ici limité à 10 itérations

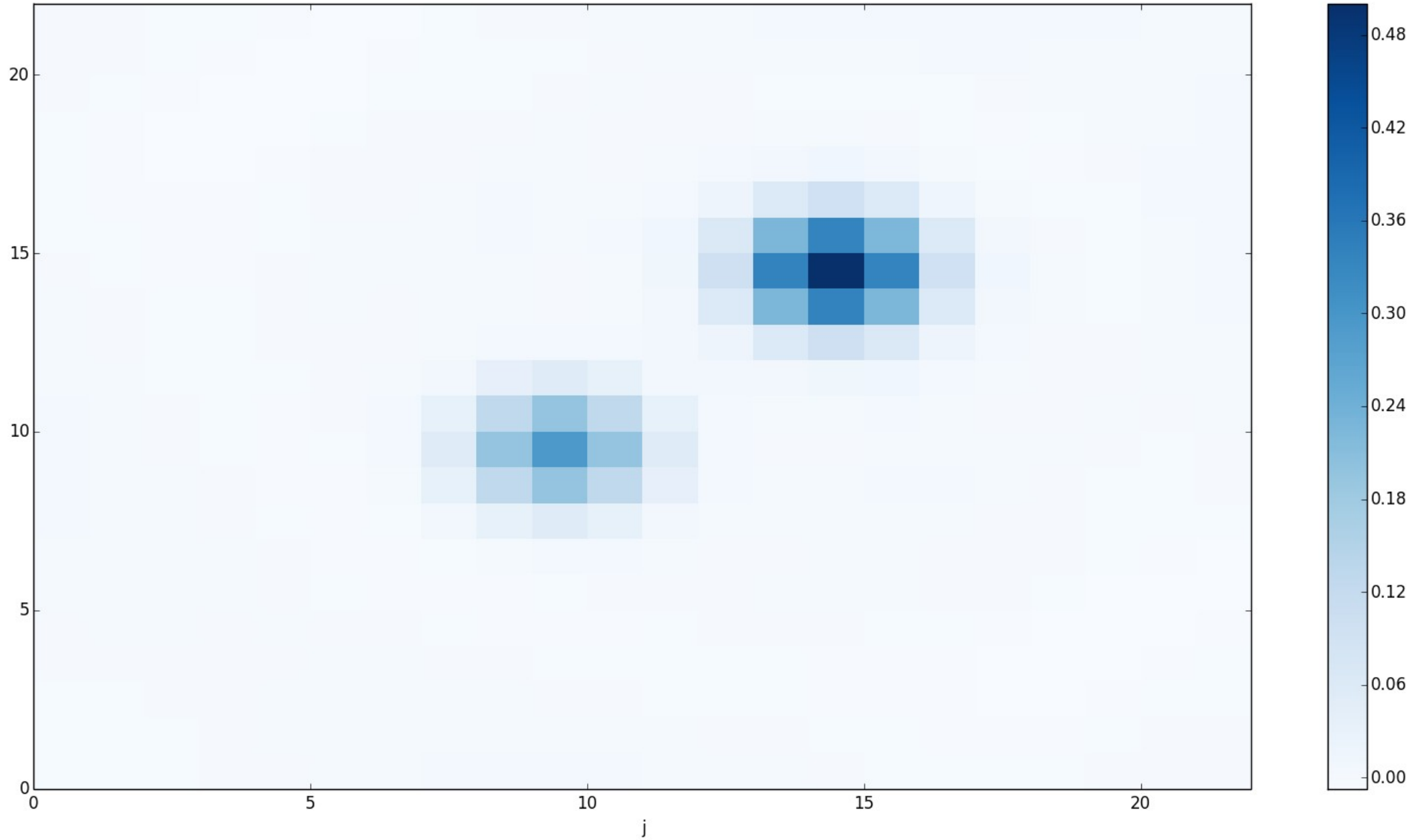
```
# minimization
  inc=np.zeros(ncv)                          # on démarre la minim avec x=xb càd u=0
  result=minimize(jvalue,ub,method='BFGS',jac=jgrad, options={'disp':True,'maxiter':10} )
  print 'success:',result.success
  print 'message:',result.message
  inc = np.dot( bmatrix , result.x )      # on inverse le préconditionnement

### add increment ###
xa = xb + inc
```

```
 xb -> j= 1.36  normgradJ= 9.792
Optimization terminated successfully.
         Current function value: 0.485714
         Iterations: 1
         Function evaluations: 3
         Gradient evaluations: 3
success: True
message: Optimization terminated successfully.
```

analyse variationnelle 1D, 2 obs



analyse variationnelle 2D, grille 23x23, 2 obs



analyse variationnelle 2D, grille 50x50, 2 obs