FigCov

July 6, 2025

[15]:
```
import gstlearn as gl
import gstlearn.plot as gp
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import random
import math
import scipy as sc
import os
#Extension of the simulation grid
nxmax = 500
nymax = 200

#Well Definition
nwell = 6
vlag  = 3

#Anisotropy ratio
ratio=1.5
ranges=[100,10]

#Some seeds
seed1  = 34556643
seed2  = 244212
seednc = 432432
seedw  = 2432145

# Color Scale
zlim = [-1.6, 2.5]

path = "fig"
if not os.path.exists(path):
    os.mkdir(path)
```

[16]:
```
db = gl.DbGrid.create(nx=nxmax)
model = gl.Model.createFromParam(gl.ECov.GAUSSIAN,range=200,space=gl.SpaceRN.
 ↪create(1))
```

```
err = gl.simtub(None,dbout=db,model=model,nbtuba=1000,seed=seed1,namconv=gl.
 ↪NamingConvention("W1"))
err = gl.simtub(None,dbout=db,model=model,nbtuba=1000,seed=seed2,namconv=gl.
 ↪NamingConvention("W2"))
#db["W2"] = db["W1"]+100
db["W1"]=db["W1"]-min(db["W1"])
db["W2"]=db["W2"]-min(db["W2"])
db["W2"]=db["W1"]+db["W2"]+1
db["W1"]=nymax*db["W1"]/max(db["W2"])
db["W2"]=nymax*db["W2"]/max(db["W2"])
db["sel"] = (db["x1"]>200) * (db["x1"]<400)
db.setLocator("sel",gl.ELoc.SEL)
db.display()
```

```
Data Base Grid Characteristics
==============================

Data Base Summary
-----------------
File is organized as a regular grid
Space dimension              = 1
Number of Columns            = 5
Total number of samples      = 500
Number of active samples     = 199

Grid characteristics:
---------------------
Origin :      0.000
Mesh   :      1.000
Number :        500

Variables
---------
Column = 0 - Name = rank - Locator = NA
Column = 1 - Name = x1 - Locator = x1
Column = 2 - Name = W1 - Locator = NA
Column = 3 - Name = W2 - Locator = z1
Column = 4 - Name = sel - Locator = sel
```

[17]:
```
def plotlim(db, title = None):
    indsel = np.where(db["sel"])[0]
    plt.plot(db["x1"][indsel],db["W1"][indsel],c = "black")
    plt.plot(db["x1"][indsel],db["W2"][indsel],c = "black")
    plt.axis("equal")
    plt.axis("off")
    if title is not None:
```

```
        plt.title("Orientation")

def plotpoint(index):
    x0 = [dbgrid["x1"][ind][index], dbgrid["x2"][ind][index]]
    dbPoint = gl.DbGrid.create(x0 = x0,nx = [1,1])
    plt.scatter(x0[0],x0[1],c = "r",s = 1)
```

[18]:
```
nxmax
```

[18]:
```
500
```

[19]:
```
modelStat = gl.Model.createFromParam(gl.ECov.
  ↪MATERN,ranges=ranges,param=1,space=gl.SpaceRN.create(2))
model = modelStat.clone()
limmin = 200
limmax = 400
dbgrid = gl.DbGrid.create([limmax,nymax])
dbgridf = gl.DbGrid.createRefine(dbgrid,[2,2])

ind = (dbgrid["x1"]).reshape(1,-1)[0].astype(int)
indf = (dbgridf["x1"]).reshape(1,-1)[0].astype(int)

dbgrid["sel"] = (dbgrid["x2"] > db[ind,"W1"]) & (dbgrid["x2"] < db[ind,"W2"])&\
                (dbgrid["x1"] > limmin) & (dbgrid["x1"] < limmax)

anglesi = np.arctan(db["W1"][1:]-db["W1"][:-1])/np.pi*180
angless = np.arctan(db["W2"][1:]-db["W2"][:-1])/np.pi*180
anglesi = np.insert(anglesi, 0, anglesi[0])
angless = np.insert(angless, 0, angless[0])

aniso = (dbgrid["x2"]-db[ind,"W1"]) / (db[ind,"W2"]-db[ind,"W1"])
aniso = anglesi[ind] + aniso * (angless[ind]-anglesi[ind])

#ratio = ratio*(db[ind,"W2"]-db[ind,"W1"])/max(db["W2"]-db["W1"])

dbgrid.addColumns(aniso,"aniso")

#dbgrid.addColumns(ratio,"ratio")
dbgrid.setLocator("sel",gl.ELoc.SEL)

dbgrid.display()
```

```
Data Base Grid Characteristics
==============================

Data Base Summary
```

```
-----------------
File is organized as a regular grid
Space dimension              = 2
Number of Columns            = 5
Total number of samples      = 80000
Number of active samples     = 19424

Grid characteristics:
---------------------
Origin :      0.000     0.000
Mesh   :      1.000     1.000
Number :        400       200

Variables
---------
Column = 0 - Name = rank - Locator = NA
Column = 1 - Name = x1 - Locator = x1
Column = 2 - Name = x2 - Locator = x2
Column = 3 - Name = sel - Locator = sel
Column = 4 - Name = aniso - Locator = NA
```

[20]:
```
mesh = gl.MeshETurbo.createFromGrid(dbgrid, mode=0, flag_polarized = True)
```

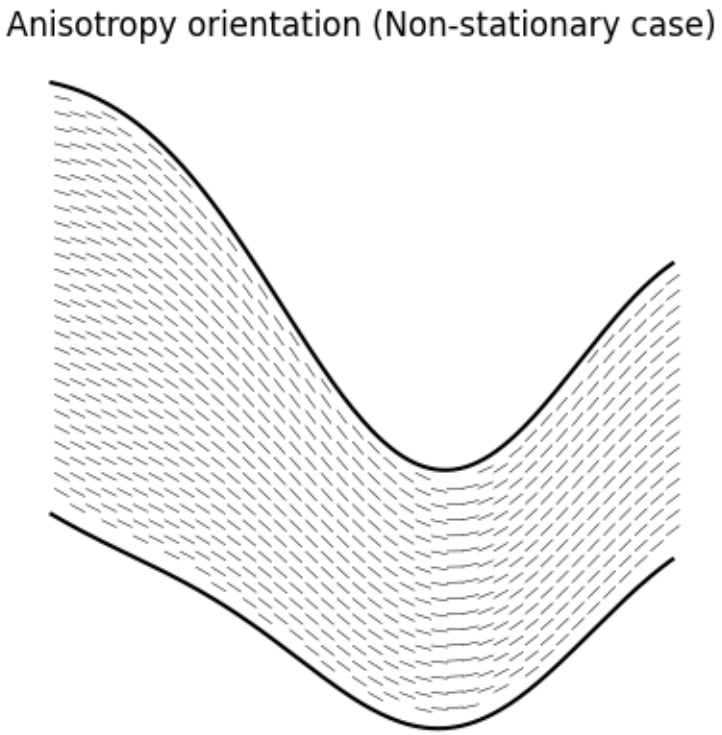
[21]:
```
dbcoarse = dbgrid.coarsify([5,5])
err = model.getCovAniso(0).makeAngleNoStatDb("aniso",0,dbgrid)
```

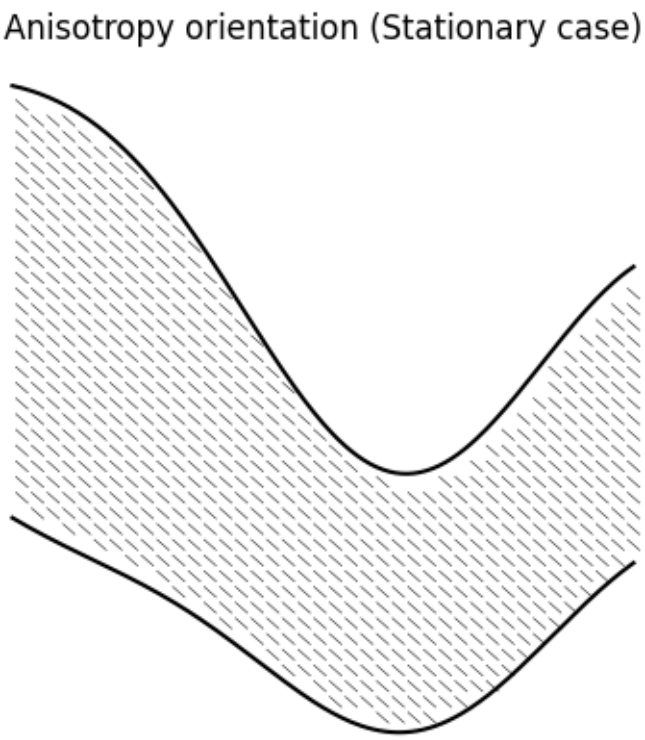
[22]:
```
indexref = 4000

ind = np.where(dbgrid["sel"])[0]
u = np.zeros(shape=dbgrid.getNSample())
angle = dbgrid["aniso"][ind][indexref]
modelStat.getCovAniso(0).setAnisoAngles([angle,0])
```

[23]:
```
dba = db.clone()
dba["W1"] = dba["W1"]-4.5
dba["W2"] = dba["W2"]+3
```

[24]:
```
plotlim(dba)
ax = gp.covaOnGrid(model.getCovAniso(0), db=dbcoarse, scale=800,␣
 ↪flagOrtho=False,headlength=0, headaxislength=0,width=.001)
plt.title("Anisotropy orientation (Non-stationary case)")
plt.savefig(os.path.join("fig","anisoNoStat.pdf"))
#plotpoint(4000)

plt.show()

plotlim(dba)
```

```
ax = gp.covaOnGrid(modelStat.getCovAniso(0), db=dbcoarse, scale=800,␣
  ↪flagOrtho=False,headlength=0, headaxislength=0,width=.001)
plt.title("Anisotropy orientation (Stationary case)")
plt.savefig(os.path.join("fig","anisoStat.pdf"))
#plotpoint(4000)
plt.show()
```



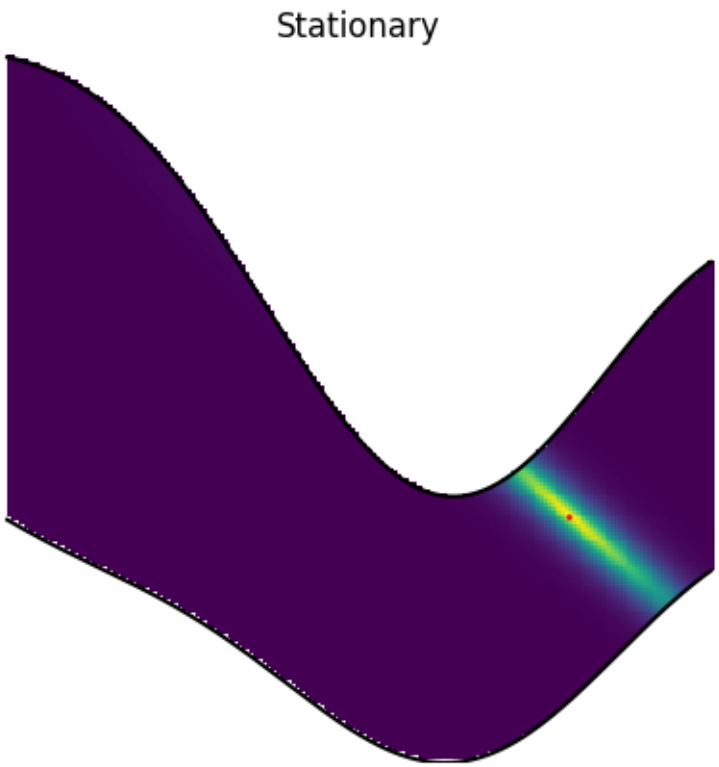


[25]:
```
index = 4000

def plotCor(indexf,indp = 0):

    ind = np.where(dbgrid["sel"])[0]
    u = np.zeros(shape=dbgrid.getNSample())
    x0 = [dbgrid["x1"][ind][indexf], dbgrid["x2"][ind][indexf]]
    dbPoint = gl.DbGrid.create(x0 = x0,nx = [1,1])

    M = modelStat.evalCovMat(dbgrid,dbPoint).toTL()
    u[ind] = M[:,0]
    dbgrid["cov"] = u
    dbrid = gl.DbGrid.createCoveringDb(dbgrid)
    ax = gp.raster(dbgrid,"cov",vmax = 1)
    plotlim(db)
    plotpoint(indexf)
    plt.axis("off")
    plt.xlim(200,400)

    plt.savefig(os.path.join("fig","CorStationary" + str(indp) + ".png"))
    plt.title("Stationary")
    plt.show()
```

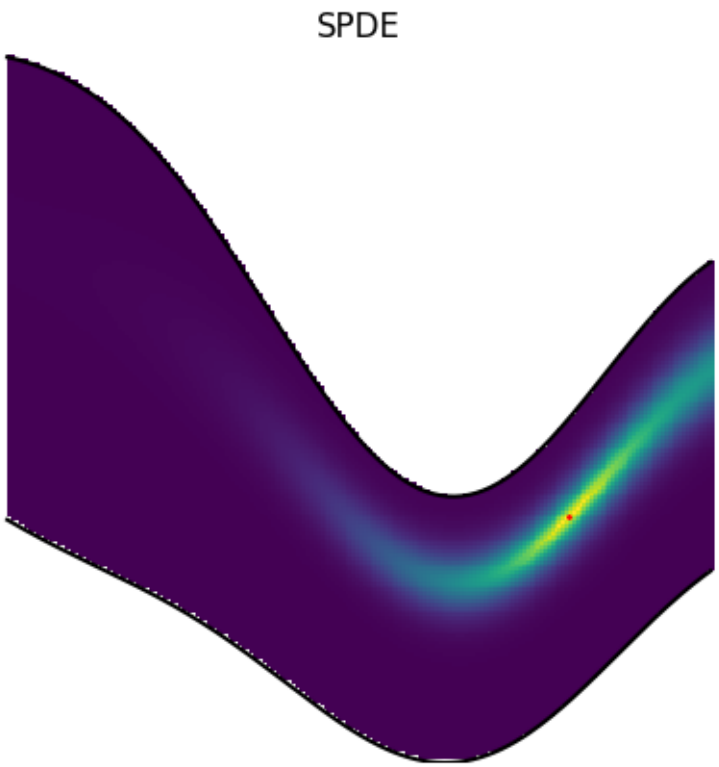
```
    spdeRes = gl.SPDE(model=model,domain=dbgrid,calcul=gl.ESPDECalcMode.
↪SIMUNONCOND, mesh=mesh)
    A = spdeRes.getPrecisionOpMatrix(0)
    Q = A.getQ().toTL()
    cov = sc.sparse.linalg.spsolve(Q,np.eye(Q.shape[0],1,-indexf))
    ind = np.where(dbgrid["sel"])[0]
    u = np.zeros(shape=dbgrid.getNSample())
    u[ind] = cov
    dbgrid["SPDE"] = cov
    gp.raster(dbgrid,"SPDE",vmin = 0,vmax=1)
    plotpoint(indexf)
    plotlim(db)
    plt.xlim(200,400)

    plt.savefig(os.path.join("fig","CorSPDE" + str(indp) + ".png"))
    plt.show()

    N = model.evalCovMat(dbgrid,dbPoint).toTL()
    u[ind] = N[:,0]
    dbgrid["covF"] = u
    ax = gp.raster(dbgrid,"covF",vmin = 0,vmax=1)

    plotpoint(indexf)
    plotlim(db)
    plt.title("Cov-based non-stationary")
    plt.xlim(200,400)

    plt.savefig(os.path.join("fig","CorCestbonbonbon" + str(indp) + ".png"))
    plt.show()
```

[26]:
```
plotCor(8000,2)
```

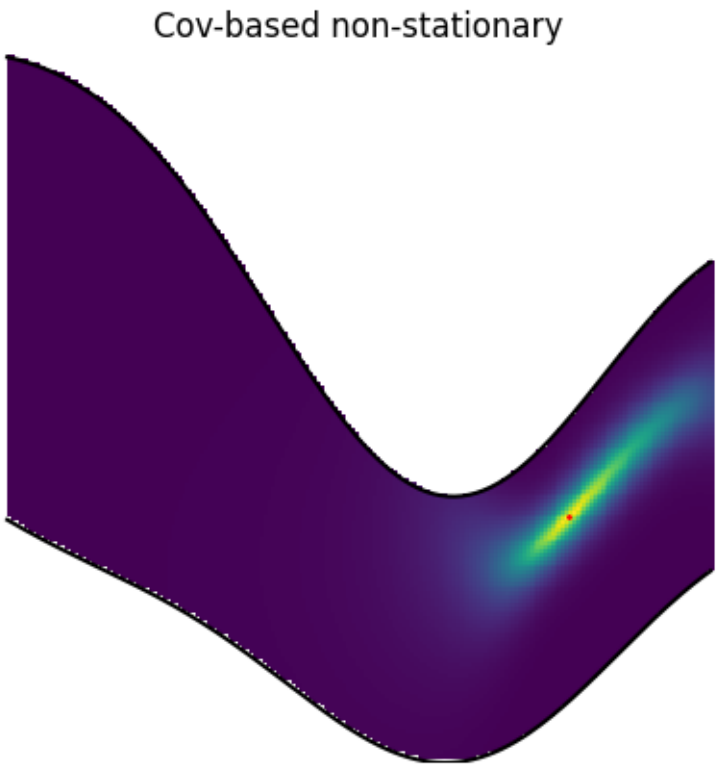
```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```




```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```

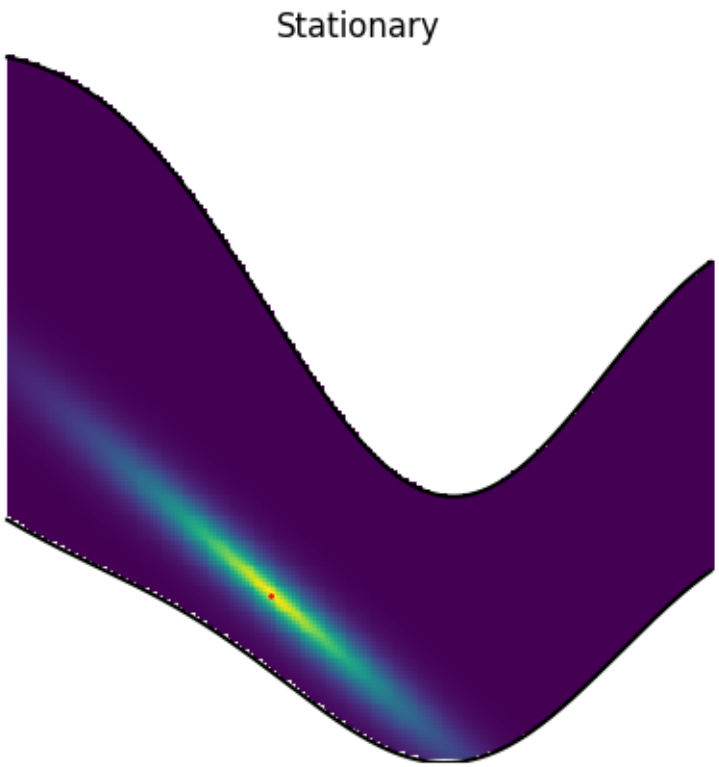



```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```

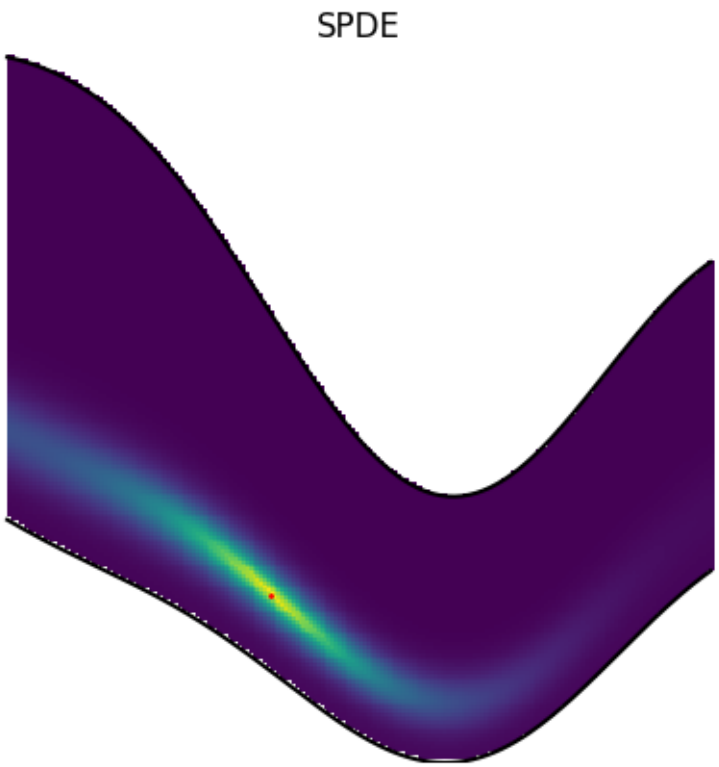


[27]:
```
a=plotCor(4000,1)
```

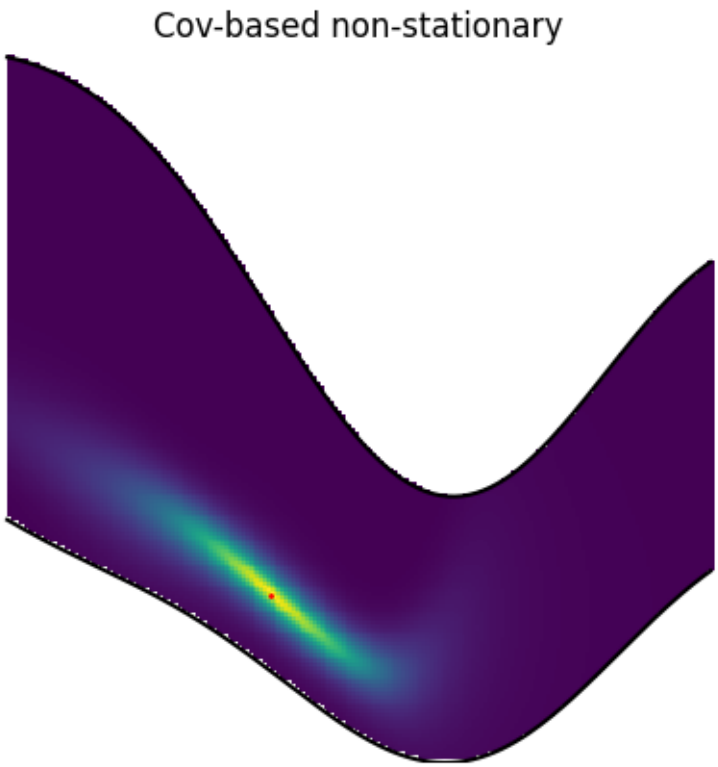
```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```




```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```




```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data
limits.
```



[28]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

# Définir une carte de couleurs
cmap = plt.cm.viridis

# Définir les limites de la colorbar
norm = mpl.colors.Normalize(vmin=0, vmax=1)

# Créer une figure et un axe pour la colorbar
fig, ax = plt.subplots(figsize=(6, 1))
fig.subplots_adjust(bottom=0.5)

# Créer la colorbar
cbar = mpl.colorbar.ColorbarBase(ax,  norm=norm, orientation='horizontal')

# Ajouter un label à la colorbar
cbar.set_label('Correlation')

# Afficher la figure
plt.savefig(os.path.join("fig","legend.png"))
```

```
plt.show()
```