Development Journal and Specs of the MESLAS package

May 19, 2020

1 General Considerations

The MESLAS package (**M**ulti-variate **E**xcursion **S**et **L**earning by **A**daptive **S**ampling) is a toolbox for simulation and prediction of mulitivariate Gaussian random fields.

The setup of the package is the following: Z is a p-dimensional random field on a domain d-dimensional domain D.
Our philosophy is to always specify spatial location and response indices together. That is, one should always specify **where** and **what**.
Spatial locations are denoted by s and response indices by ℓ. We will use boldface (or, in the code, alternatively uppercase or plurals) to denote vectors of such objects.

A generalized sampling location is thus entirely defined by specifying two vectors

$$\begin{aligned}\boldsymbol{s} &= (s_1, ..., s_n) \in D^n \\ \boldsymbol{\ell} &= (\ell_1, ..., \ell_n) \in \{1, ..., p\}^n\end{aligned}$$

We will refer to n as the *dimension* of the generalized sampling location and usually just talk of location, using the word *spatial location* when we want to specifically refer to points in D. Also, we will use boldface x as a shortcut to refer to the couple $(\boldsymbol{s}, \boldsymbol{\ell})$ of spatial location vector and response index vector. The shortcut notation $Z_{\boldsymbol{x}}$ thus refers to the vector

$$Z_{\boldsymbol{x}} := \left(Z_{s_1}^{\ell_1}, ..., Z_{s_n}^{\ell_n}\right) \in \mathbb{R}^n.$$

1.1 Covariance Model

We assume a factor model which is the product of a stationary spatial component with a response-index component

$$\mathrm{Cov}\left(Z_s^i, Z_t^j\right) = k\left(s - t\right)\gamma\left(i, j\right). \tag{1}$$

This makes implementation easier, since then, to compute the covariance matrix of a generalized observations (S, L), we first compute the pairwise distance matrix

$$H = \mathrm{cdist}\left(S, S, p = 2\right) = \begin{pmatrix} ||s_1 - s_1|| & \dots & ||s_1 - s_n|| \\ \vdots & & \vdots \\ ||s_n - s_1|| & \dots & ||s_n - s_n|| \end{pmatrix}$$

which can then be feeded to a vectorized stationary covariance function to get $K(H)$.
For the response index part, we compute $L_1, L_2 = \mathrm{meshgrid}\left(L, L\right)$ which yields

$$L_1 = \begin{pmatrix} l_1 & \dots & l_1 \\ \vdots & & \vdots \\ l_n & \dots & l_n \end{pmatrix},\ L_2 = \begin{pmatrix} l_1 & \dots & l_n \\ \vdots & & \vdots \\ l_1 & \dots & l_n \end{pmatrix}$$

and then feed it to a vectorized cross-covariance function $\gamma(L_1, L_2)$. Finally, we get the covariance matrix by elementwise multiplication

$$K = K\left(H\right) \odot \gamma\left(L_1, L_2\right)$$

1.2 Cross-Covariance Models

We here review different usual models for the cross-covariance part $\gamma(.,.)$ of the covariance function. Recall that this is the part that specifies how different components of the response vector at one fixed location interact.

The simplest model we will consider is **uniform mixing**. In this model, all components interact with the same coupling γ_0:

$$\gamma(l,m) = \begin{cases} \sigma_l^2, \ l = m \\ \gamma_0 \sigma_l \sigma_m, \ l \neq m \end{cases} \tag{2}$$

and $\sigma_1^2, ..., \sigma_p^2$ are the variances of the individual components.

1.3 Implementation Details

ATTENTION: torch.meshgrid behaves differently than numpy's one. First of all, it takes single dimensional vectors.

$$L = (1,2,3,4), \ \text{torch.meshgrid}(L,L) = \begin{pmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ n & \dots & n \end{pmatrix}, \ \begin{pmatrix} 1 & \dots & n \\ \vdots & & \vdots \\ 1 & \dots & n \end{pmatrix}$$

1.4 Mean Module

1.5 Covariance Module

1.6 Gaussian Random Field Class and Sampling

1.7 Gridding

2 Example Run

We consider a 2-dimensional GRF on a a 2-dimensional 100×100 regular grid on $[0,1]^2$. The GRF has a factor covariance model, where the spatial part is a Matérn 3/2 with unit variance and lengthscale $\lambda_0 = 0.1$. The cross-covariance is a uniform mixing with parameters

$$\sigma_1^2 = 0.25,\ \sigma_2^2 = 0.6, \gamma_0 = 0.3$$

and the mean function is a constant one with $\mu_0 = (1, -2)$.

The plot below shows one realization of the field on the full grid.

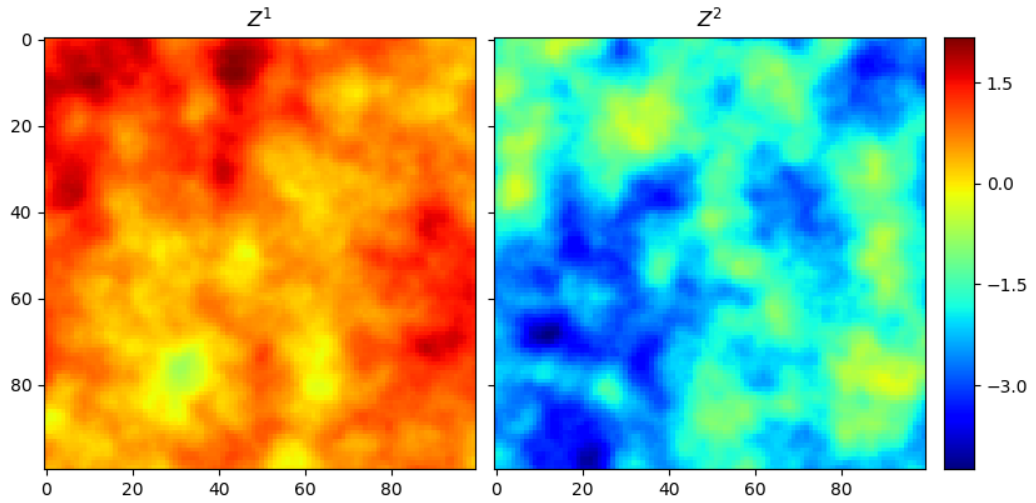


Figure 1: Simulated first (left) and second (right) component of the field.

We can also increase the cross-correlation factor γ_0 to 0.9 to see its effect.

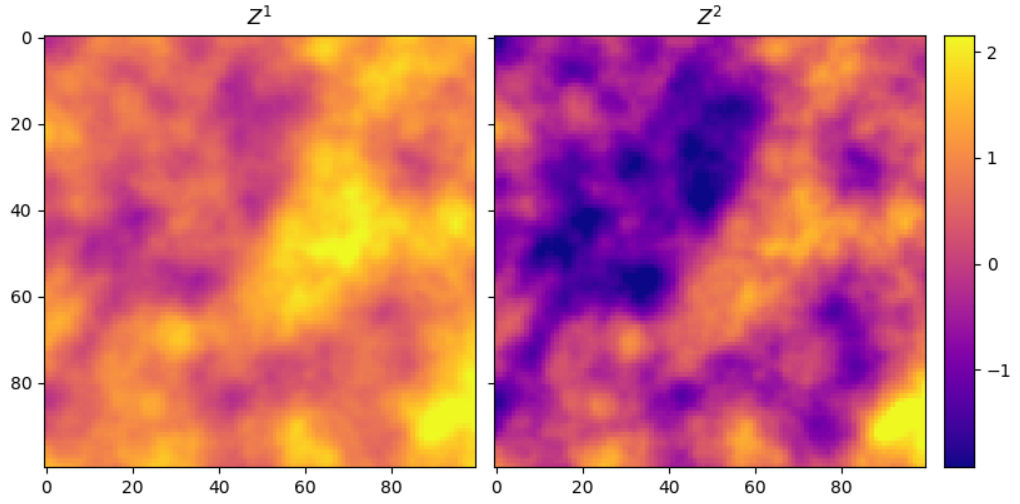


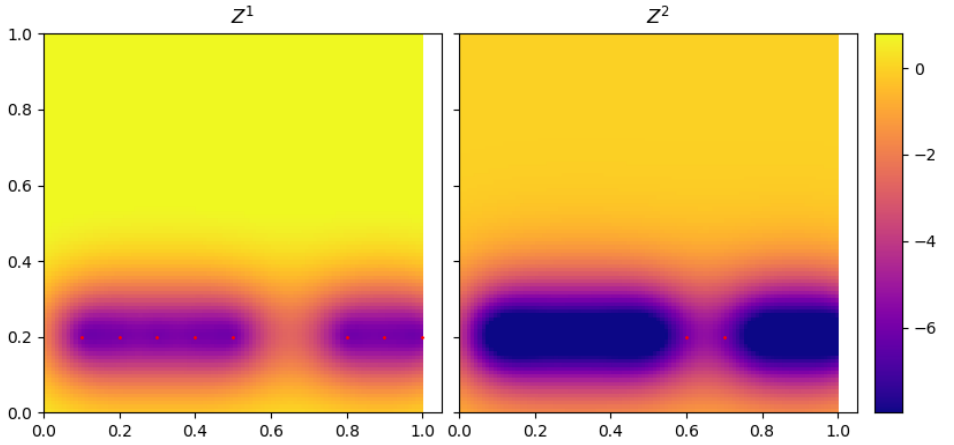
Figure 2: Simulation of highly cross-correlated field.

The code below shows how simple it is to sample a multivariate GRF using MESLAS.

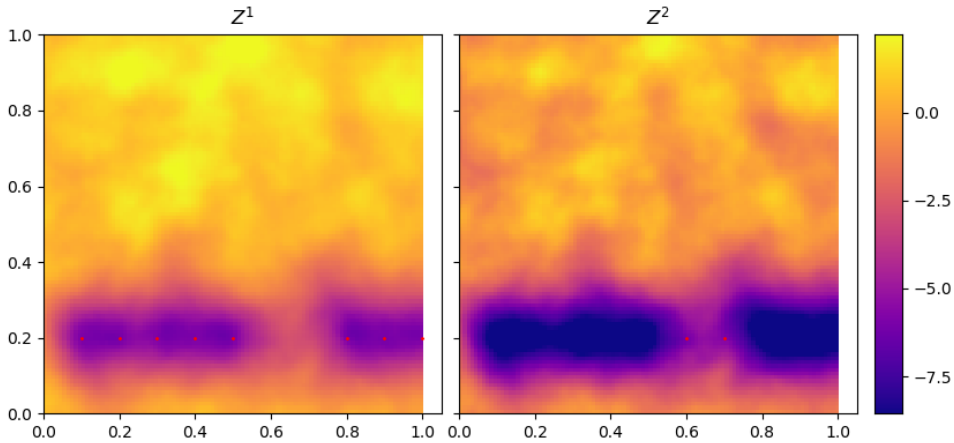
```
""" Demonstrates how to define a multivariate Gaussian Random Field,
sample a realization and plot it.

"""
import numpy as np
import torch
from meslas.means import ConstantMean
from meslas.covariance.covariance_functions import Matern32
from meslas.covariance.cross_covariances import UniformMixing
from meslas.covariance.heterotopic import FactorCovariance
from meslas.grid import Grid
from meslas.sampling import GRF


# Dimension of the response.
n_out = 2

# Spatial Covariance.
matern_cov = Matern32(lmbda=0.1, sigma=1.0)

# Cross covariance.
cross_cov = UniformMixing(gamma0=0.9, sigmas=[np.sqrt(0.25), np.sqrt(0.6)])

covariance = FactorCovariance(matern_cov, cross_cov, n_out=n_out)

# Specify mean function
mean = ConstantMean([1.0, 0])

# Create the GRF.
myGRF = GRF(mean, covariance)

# Create a regular square grid in 2 dims.
# Number of repsones.
dim = 2
my_grid = Grid(100, dim)

# Sample all components at all locations.
sample = myGRF.sample_grid(my_grid)

# Plot.
from meslas.plotting import plot_2d_slice
plot_2d_slice(sample)
```

3 Co-Kriging

Cokriging is also implemented in full generality (heterotopic) in MESLAS. The package also provides conditional simulations. Plots below show an example of cokriging and conditional simulation. The GRF is the same as in the previous section, with a high cross-correlation $\gamma_0 = 0.9$.



(a) Conditional mean



(b) Conditional realisation

Figure 3: Example of co-kriging a 2-dimensional GRF with MESLAS, observation locations in red.

4 Coverage Function

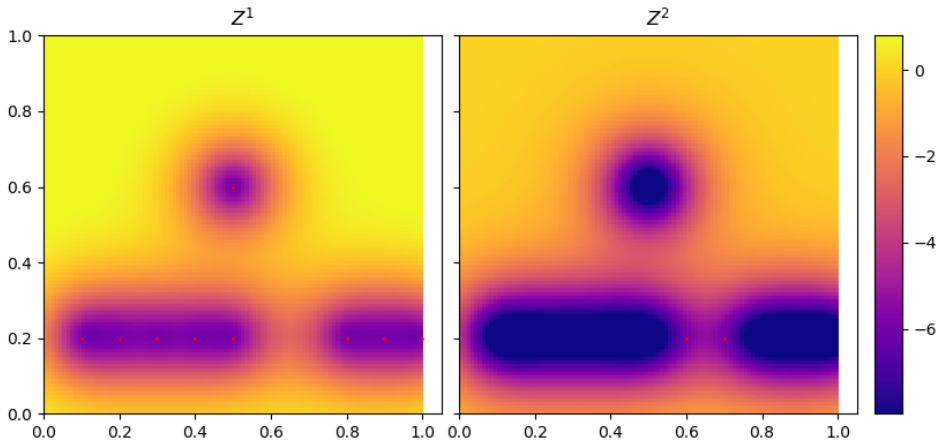
We compute the p-dimensional CDF using the MVNORM package of Sebastion Marmin. We re-packaged it for streamlined distribution via PiPy. The implementation is 3 times faster than barebone PyTorch (in dimension 2). It is also vectorized over the batch dimension.
Below we demonstrate the computation of the coverage function for excursion above $t = (-1.0, -1.0)$.

4.1 Illustrations

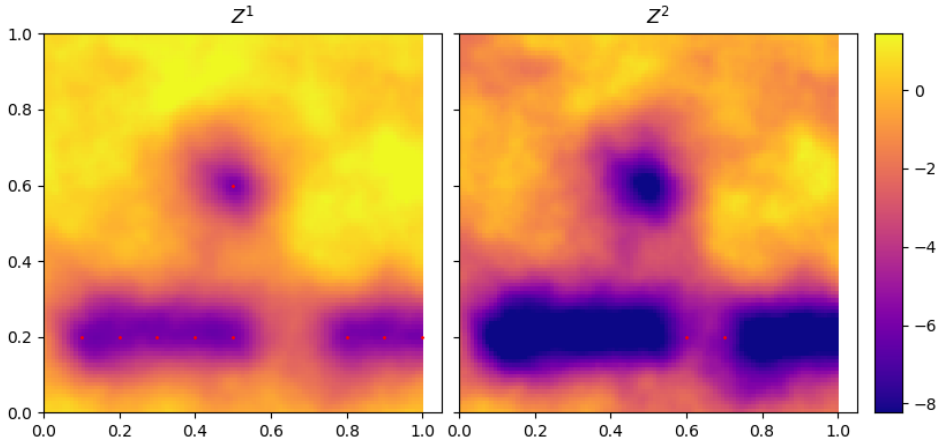
The agreed-upon goal was to produce plots to illustrate multivariate excursion sets vs univariate.
I suggest we do that by trying to replace figure 4 in the paper. So: sample unconditionally, illustrate various excursions, krig using this realisation, plot coverages.

5 Hardware/Software Specifications

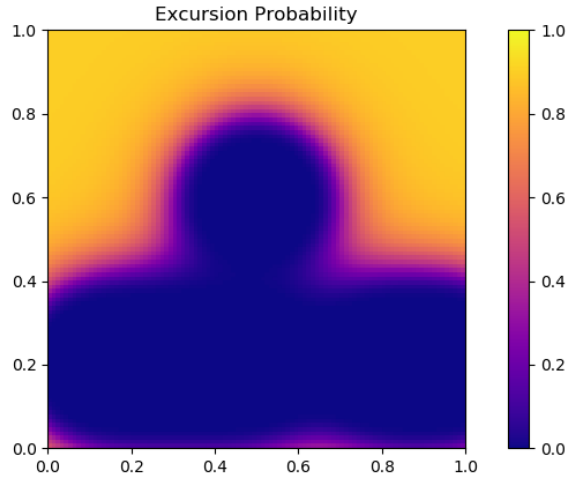
The AUV feature a Nvidia TX1 with ubuntu 16.04. Current programs use Python 2.7, but 3.7 can be used. In order to run 3.7, we have to run another backseat-driver (responsible for sending waypoints to the low-leve controllers).



(a) Conditional mean

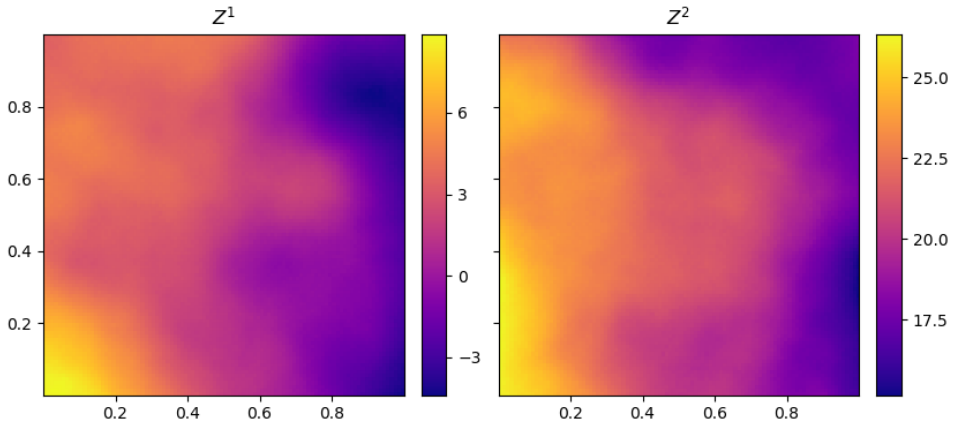


(b) Sample conditional realisation

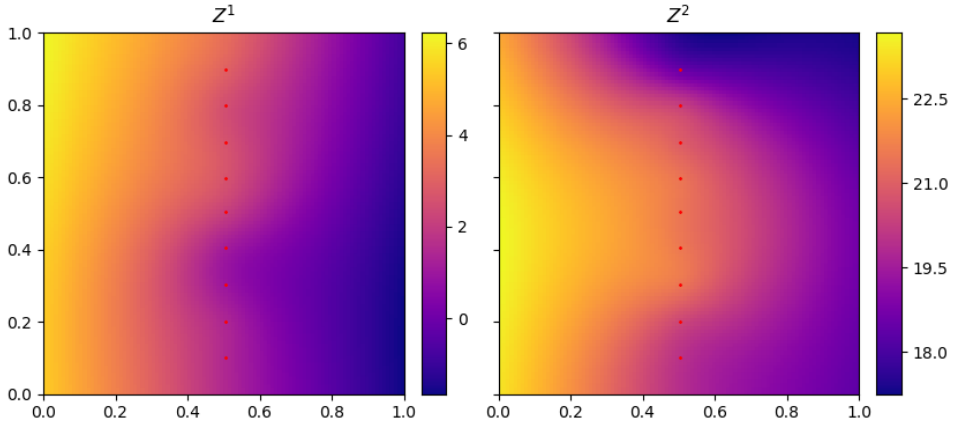


(c) Conditional Coverage Function

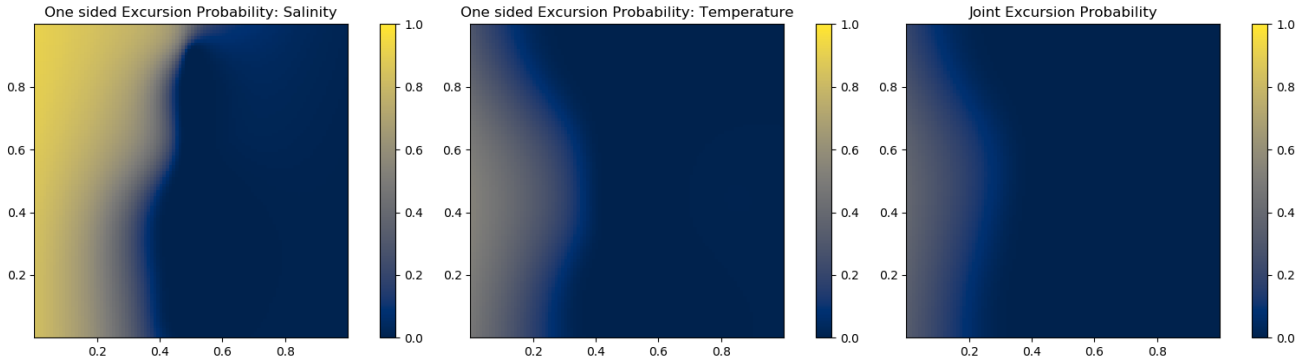
Figure 4: Diagnostics of problems in computation of the coverage function.



(a) Simulated Realisation



(b) Cokriging Mean (observation locations in red)



(c) Coverage Function

Figure 5: Diagnostics of problems in computation of the coverage function.