Chapter 13

On Kriging Techniques & Impedance-based SHM as Applied to Damage Detection in 2D Structures

Chapter details

Chapter DOI:

https://doi.org/10.4322/978-65-86503-88-3.c13

Chapter suggested citation / reference style:

Pereira, Paulo E. C., et al. (2022). “On Kriging Techniques & Impedance-based SHM as Applied to Damage Detection in 2D Structures”. *In* Jorge, Ariosto B., et al. (Eds.) *Uncertainty Modeling: Fundamental Concepts and Models*, Vol. III, UnB, Brasilia, DF, Brazil, pp. 427–458. Book series in Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity.

P.S.: DOI may be included at the end of citation, for completeness.

Book details

Book: Uncertainty Modeling: Fundamental Concepts and Models

Edited by: Jorge, Ariosto B., Anflor, Carla T. M., Gomes, Guilherme F., & Carneiro, Sergio H. S.

Volume III of Book Series in:

Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity

Published by: UnB City: Brasilia, DF, Brazil Year: 2022

DOI: https://doi.org/10.4322/978-65-86503-88-3

On Kriging Techniques & Impedance-based SHM as Applied to Damage Detection in 2D Structures

Paulo Elias C. Pereira[1*], Daniel R. Gonçalves[2], Stanley W. F. Rezende[1*]
Jose dos Reis V. Moura Jr[2] and Roberto M. Finzi Neto[1]

[1]Post-Graduate Program - Mechanical Engineering, Federal University of Uberlandia, Brazil. e-mail: paulo_elias_carneiro@ufcat.edu.br; stanley_washington@ufu.br; finzi@ufu.br
[2]Post-Graduate Program - Modeling and Optimization, Federal University of Catalão, Brazil. e-mail: danielresendeg@gmail.com; zereis@ufcat.edu.br
*Corresponding author

Abstract

In Impedance-based Structural Health Monitoring Systems, it is essential to identify the location of possible damage so that corrective measures can be taken promptly. Several approaches have been developed to predict the location of damage, including the one based on indicator kriging. This chapter illustrates the computational packages in Python for carrying out estimates by kriging methods. A case study of the location of structural damage in the center of an aluminum plate using indicator kriging is presented. They are detailed, sequentially, the necessary steps, and discussing each one. The results obtained with the case study showed that it was possible to map the place with the highest probability of occurrence of damage, which proved to be consistent with the actual position of the same, which highlights the ability of the proposed approach to predict the location of structural damage. In addition, perspectives regarding future research involving the use of the technique are discussed.

Keywords: Impedance-based structural health monitoring; Kriging methods; Indicator kriging; Damage location

1 Introduction

The electromechanical impedance technique has been vastly applied to the monitoring of structures due to its low cost - concerning other approaches - and its characteristic of being a non-destructive evaluation (NDE) method (Giurgiutiu [2014]). Further, the method can access the structure's conditions at relatively high frequencies (usually between 30 kHz and 400 kHz), which reduces the environment's interferences, allowing the structure's monitoring at small scales, such as cracks, delamination, and disbonds, not detectable by other methods (Bhalla and Soh [2012]; Zagrai and Giurgiutiu [2009]).

Post-processed information obtained by ISHM refers to damage metric data (e.g., RMSD and CCD indexes), which can indicate the presence of damage in the structure by comparing the electrical impedance signatures at any moment in time and at the initial (pristine) condition of the structure (Giurgiutiu [2014]; Giurgiutiu and Rogers [1998]). As each damage metric data is related to a specific actuator, it's necessary to interpolate all data collected, each assigned to a point in the space (actuator position), to identify the damage location.

The works of Kravolec et al. [2018] and Cherrier et al. [2013] evidenced the existence of a correlation between the electrical impedance signatures and the distance from the damage, which suggests the presence of spatial correlation between the damage metric data arranged in the form of a network. Results derived from the application of kriging methods (Gonçalves et al. [2021]; Gonçalves et al. [2020]) for the interpolation of RSMD values in an aluminum plate revealed a spatial correlation between these, allowing the identification of the damage location, especially when using the indicator kriging method. Despite the positive results achieved, more research on this approach is necessary, which can allow several new possibilities.

This chapter presents the theoretical concepts related to indicator kriging for applications in structural health monitoring. First, concepts of ISHM and kriging are exposed, followed by computational routines on Python packages for estimations based on kriging methods. Finally, a case study is presented regarding the location of damage from indicator kriging, where the steps involved are sequentially detailed, accompanied by source codes in Python language.

2 Fundamentals of ISHM and Kriging

2.1 Fundamentals of ISHM

There are several techniques for monitoring the occurrence and propagation of structural damage to increase the useful life, improve the performance of structures/equipment and reduce costs (Park et al. [2003]). Among the SHM techniques, there is the Electromechanical Impedance-based Structural Health Monitoring (EMI-SHM), which is based on the electromechanical coupling that results from the bonding of piezoelectric transducers to the structure or equipment that will be monitored, keeping the impedance function defined, which depends on the mechanical characteristics of the transducer and structure to be monitored (Lin and Giurgiutiu [2006]; Park et al. [2003]).

In the EMI-SHM technique, PZT transducers are usually excited by a sinusoidal waveform with an amplitude of approximately 1V RMS (effective voltage) and a frequency range of 10 to 250 kHz (Raju [1997]), or even as high as 1,000 kHz, depending on the structure and type of application (Giurgiutiu et al. [1999]). The lower frequency band covers a larger detection area, while the higher frequency band can determine the location of damage (Sun et al. [1995]). Among the advantages of the high-frequency response is that the wavelength of the signal applied to the structure is short enough to detect even small initial cracks. Such cracks can expand and cause serious failures depending on the structure (Park et al. [2003]).

Electromechanical models were formulated to describe the relationship between the structure and the piezoelectric transducer. Liang et al. [1994] define the electrical impedance of the transducer $Z_E(\omega)$ associated with the mechanical impedance of the structure $Z_s(\omega)$ by Eq. 1.

$$Z_E(\omega) = \frac{1}{j\omega\tau}\left(\epsilon_{33}^T - \frac{Z_s(\omega)}{Z_s(\omega)+Z_p(\omega)}d_{3X}^2 Y_{XX}^E\right)^{-1} \tag{1}$$

where $Z_p(\omega)$ corresponds to the mechanical impedance of the transducer; ω is the angular frequency; τ, a geometric constant; ϵ_{33}^{T}, the dielectric constant for constant mechanical stress (T); Y_{XX}^{E}, Young's modulus for a constant electric field (E); d_{3X}^{2}, a dielectric constant; and j, an imaginary unit.

In the one degree of freedom model presented by Liang et al. [1994], the influence of the adhesive layer (glue) between the transducer and the structure is not considered. Thus, the PZT patch is excited at a high frequency and returns the impedance curve of the structure studied. Changes in the mechanical impedance of the structure due to a failure lead to a modification of the signal (electrical impedance) emitted by the PZT patches coupled or embedded in the structure, whose signal is measured within a frequency range. Changes in structural conditions are then identified by comparing the impedance signatures before and after the fault's occurrence (Maruo et al. [2015]; Bhalla and Soh [2012]; Baptista and Vieira Filho [2010]; Zagrai and Giurgiutiu [2009]; Peairs et al. [2007b]; Peairs et al. [2007a]).

The measurement of the difference between the electrical impedance signals before and after the occurrence of changes in the structure's status is done through some indexes, among which we have the Root Mean Square Deviation-RMSD (Eq. 2) and the Correlation Coefficient Deviation-CCD (Eq. 3).

The RMSD (Eq. 2) measures the similarity of impedance attributed to the same frequency and is supported by the Euclidean standard (Giurgiutiu [2014]; Giurgiutiu and Rogers [1998]). This indicator is based on the comparison between the impedance values before the damage, that is, in the pristine condition ($Z_E^0(k)$), whose signal is also called the baseline, and those obtained at some point in the future ($Z_E(k)$), all measured at several k points in the frequency domain, whose range varies from ω_I (initial frequency) to ω_F (final frequency).

$$RMSD = \sqrt{\frac{\sum_{k=\omega_I}^{\omega_F}\left[Z_E(k) - Z_E^0(k)\right]^2}{\sum_{k=\omega_I}^{\omega_F}\left[Z_E^0(k)\right]^2}} \tag{2}$$

Unlike the RMSD, which measures the average distance/difference between the impedance curves, the CCD (Eq. 3) analyzes the impedance curves globally, based on the correlation coefficient (Marqui et al. [2008]).

$$CCD = 1 - \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\left(\mathrm{Re}\left(Z_{\text{base},i}\right) - \mathrm{Re}\left(Z_{\text{base}}^{-}\right)\right)\left(\mathrm{Re}\left(Z_{FIS,i}\right) - \mathrm{Re}\left(Z_{FIS}^{-}\right)\right)}{S_{FIS}S_{\text{base}}}\right) \tag{3}$$

In Eq. 3, n is the number of frequency-domain values, $Re(Z_{(base,i)})$ is the real part of the impedance signal in the pristine condition, $Re(Z_{base})$ is the average of the real part of the impedance signal in the pristine status, $Re(Z_{(FIS,i)})$ is the real part of the PZT patch impedance signal after damage, $Re(Z_{FIS})$ is the average of the real part of the PZT patch impedance signal after damage, S_{FIS} the standard deviation of the real part of the PZT impedance signal that represents the damage and S_{base} is the standard deviation of the real part of the PZT impedance signal that represents the signal in the intact condition.

An impedance analyzer such as the HP4194A is the simplest way to determine the electromechanical impedance of smart structures (structures containing PZT transducers). However, the equipment cost is approximately US$ 40,000 and weighs about 30 kg. There are other options with limited functions that cost less than $2,000. However, they are still bulky (weighing several pounds). As a consequence, researchers have been looking for alternative ways to perform this task (Maruo et al. [2015]).

2.2 Semivariogram

Estimating unsampled points and regions using kriging methods are based on the spatial correlation of the analyzed variable. The spatial continuity means that sample (or input) data close to each other have similar values attributed to them, whose difference increases as the distance between them becomes higher (Revuelta [2018]; Chilès and Delfiner [2012]). In this sense, closer sample (or input data) points have a more significant correlation with each other than those more distant - spatially - among each other.

The quantification of the spatial continuity is done using the semivariogram function ($\gamma(h)$), defined in the Eq. 4, which calculates the mean quadratic difference between the sample (or input) point pairs $z(x_i)$, located at a point x_i of the - stationary - domain being analyzed, and $z(x_i + h)$, located at the $x_i + h$, distant h from the first point, where h refers to the distance vector between the data being compared (Journel and Huijbregts [1978]). The term $N(h)$ represents the number of the sample (or input) data pairs in the direction under analysis.

$$\gamma(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} (z(x_i) - z(x_i + h))^2 \tag{4}$$

The calculation of the semivariogram for several classes of distances in a specific spatial direction provides an inventory of the continuity in this one in the form of a graphic (experimental semivariogram) comparing the semivariance values ($\gamma(h)$) and their respective classes of distances (Figure 1). The growth of the function represents the progressive loss of spatial continuity between the compared sample (or input) pairs as the distance between them increases (Abzalov [2016]; Chilès and Delfiner [2012]; Sinclair and Blackwell [2002]). The total loss of continuity between the input points is evidenced at the point from which the growth of the function stops, and the next ones stay oscillating, normally around the variance *a priori*, which recurrently corresponds to the variance of the variable being analyzed within the domain.

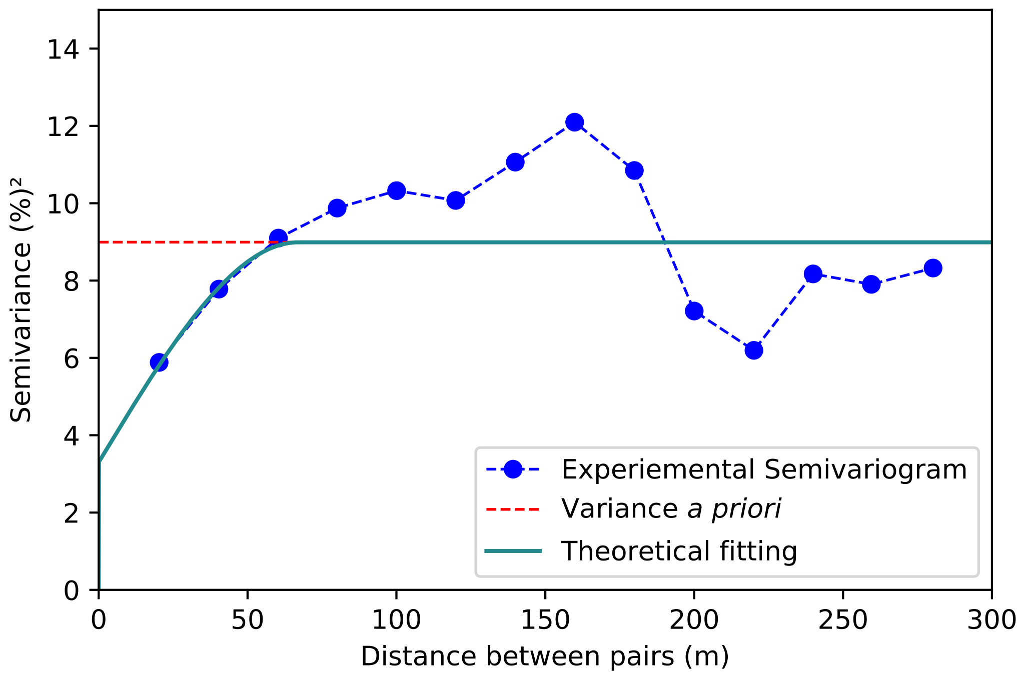

Figure 1: Experimental semivariogram with its respective theoretical fitting

Mathematical functions must adjust experimental semivariograms for their use in the estimation processes (kriging equations). The justifications for this encompass several aspects, whose arguments and discussions can be found in Journel and Huijbregts [1978], Rossi and Deutsch [2014] and Revuelta [2018]. A semivariogram model can be constituted of one or more mathematical functions, depending on the shape of the experimental semivariogram (Kitanidis [1997]). Some mathematical functions for semivariogram fitting include the following (Figure 2): Spherical, exponential, and Gaussian, which are characterized to have a sill (semivariance value) limiting the increase of the function.

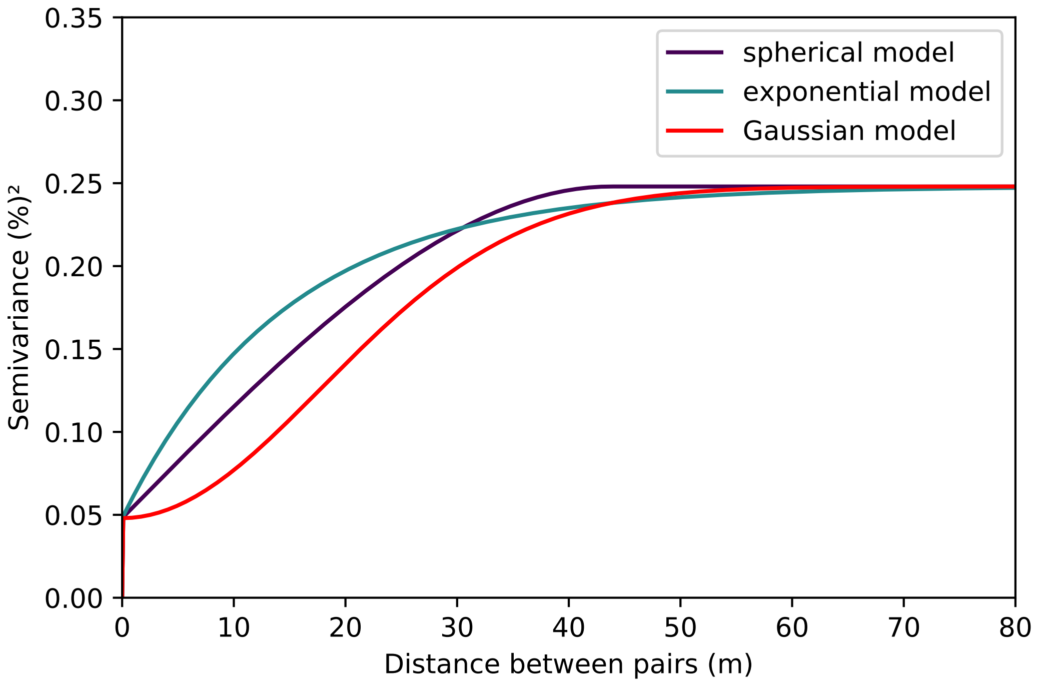

Figure 2: Graphical representation of the spherical, exponential, and Gaussian semivariogram models. All of them are represented using the same parameters.

Three parameters define the semivariogram models (Figure 3): Nugget Effect, Sill, and Range. The first one is a chaotic or unstructured variance. It represents a variability that occurs at a scale lower than the spacing between the sample (or input) data and, therefore, is not mapped by the semivariogram and is also due to sampling errors. The Sill corresponds to the variance of the variable in the direction of the vector h, and the Range is the distance from which there's no more spatial correlation between the pairs of values in the direction being mapped (Journel and Huijbregts [1978]; Hustrulid et al. [2013]; Rossi and Deutsch [2014]).

Calculating the semivariograms in several directions in the analyzed - and stationary - domain quantifies the spatial continuity behavior of the variable under analysis. In this context, directional semivariograms will frequently show different spatial continuities in the function of the spatial direction (vector h), which will indicate the presence of a direction endowed with a greater homogeneity between the values along this one and another one with a lower homogeneity (higher variability). Cases in which the semivariograms' shape are a function of the spatial direction corresponds to the occurrence of anisotropy, among them, the most common is the geometric anisotropy, in which the range varies in function of the spatial direction (vector h), and the sill remains constant (Figure 4) (Armstrong [1998]; Isaaks and Srivastava [1989]; Journel and Huijbregts [1978]).

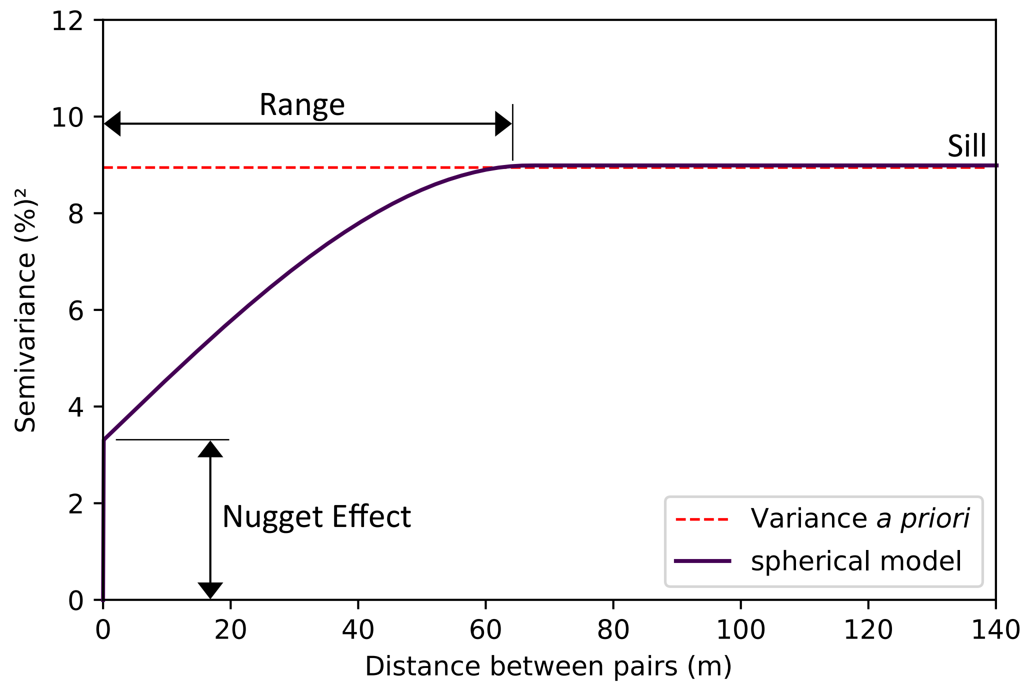

Figure 3: Example of a spherical semivariogram model with its parameters.

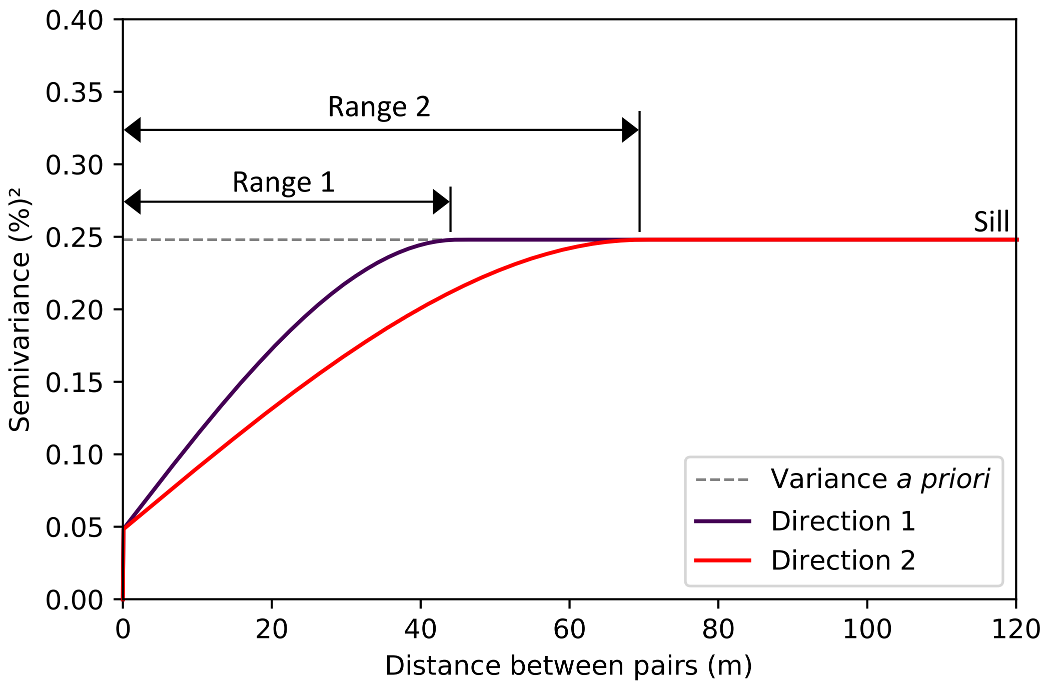

Figure 4: Semivariograms showing a phenomenon with geometric anisotropy.

The possibility to map the spatial continuity of a variable (e.g., damage index) using semivariograms can help identify possible preferential directions of structural damage propagation, which will be shown in Section 4, where there is an occurrence of geometric anisotropy in indicator variables obtained from RMSD-based damage index values.

The information obtained from semivariograms allows the complete characterization of the variable's spatial continuity and is used for estimating its values at the unsampled locations by kriging methods.

2.2.1 Kriging Methods

The term kriging covers a wide variety of estimation methods, characterized to be exact interpolators, whose mathematical formulations of the main types can be obtained in the works of Rossi and Deutsch [2014], Chilès and Delfiner [2012], Sinclair and Blackwell [2002], Kitanidis [1997] and Journel and Huijbregts [1978].

Among the kriging methods, there is the ordinary kriging one, which is a local estimation method, where the estimation of a point at an unsampled location ($Z^*_{\text{KO}}(x_0)$) is obtained by a linear combination (Eq. 5) of n sample (or input) values $Z(x_i)$ contained in the search neighborhood, where the weights λ_i reflects the spatial correlation between the unsampled point (or block) and the actual (or input) values within the search neighborhood, and between the sample (or input) values among each other contained in the search neighborhood (Yamamoto and Landim [2013]).

$$Z^*_{\text{KO}} = \sum_{i=1}^{n} \lambda_i Z(x_i) \tag{5}$$

The weights of the linear combination shown in the Eq. 5 are obtained through the solution of the ordinary kriging's system of equations (Eq. 6), where it's intended to get the combination of weights in such a way that the kriging variance is minimized, considering the aspects of spatial continuity of the variable under estimation, being these, reproduced in the semivariogram values between sample (or input) values in the x_i and x_j locations ($\gamma(x_i - x_j)$), and the semivariogram values between the point to be estimated at the x_0 location and the input data at the x_i locations ($\gamma(x_i - x_0)$). Also, a restriction, which refers to the need for the sum of the weights to be unitary, is imposed on the system to obtain unbiased estimates, called an unbiased constraint. This system can be expressed in the matrix form (Eq. 7), from which the optimal weights are calculated (Yamamoto and Landim [2013]).

$$\begin{cases} \sum_{j=1}^{n} \lambda_j \gamma(x_i - x_j) - \mu = \gamma(x_i - x_0) & \text{for } i = 1, \dots, n \\ \sum_{j=1}^{n} \lambda_j = 1 \end{cases} \tag{6}$$

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ -\mu \end{bmatrix} = \begin{bmatrix} \gamma(x_1 - x_1) & \gamma(x_1 - x_2) & \dots & \gamma(x_1 - x_n) & 1 \\ \gamma(x_2 - x_1) & \gamma(x_2 - x_2) & \dots & \gamma(x_2 - x_n) & 1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \gamma(x_n - x_1) & \gamma(x_n - x_2) & \dots & \gamma(x_n - x_n) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix}^{-1} \times \begin{bmatrix} \gamma(x_0 - x_1) \\ \gamma(x_0 - x_2) \\ \vdots \\ \gamma(x_0 - x_n) \\ 1 \end{bmatrix} \tag{7}$$

In addition to the variable's estimated value at the unsampled location x_0, the kriging methods are capable of providing the estimation variance, which, in the case of the ordinary kriging method, is given by the Eq. 8, where μ is the Lagrange parameter, and n refers to the number of samples (input data) used for the estimation at an unsampled location.

$$\sigma^2_{\text{KO}} = \sum_{i=1}^{n} \lambda_i \gamma(x_i - x_0) + \mu \tag{8}$$

The ordinary kriging estimator can be used as the base for estimation of indicator variables, characterized to be binary, the situation in which there is the so-called indicator kriging, being a non-parametric method, in such a way that the statistical distribution of the variable doesn't restrict its application under estimation (Rossi and Deutsch [2014]; Revuelta [2018]). The indicator kriging performs the estimation of binary variables, whose values are determined according to a specific criteria, as shown in the Eq. 9, where, from an original distribution, are indicated K thresholds z_k, and for each of them, are created indicator variables, coding the values according to the value's position of the original variable ($Z(x)$) relative to the specified threshold z_k (Eq. 9).

$$I(x, z_k) = \begin{cases} 1, & \text{if } Z(x) \leq z_k \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

In the case of using the ordinary kriging as the interpolator for the estimation of indicator variables at the unsampled locations, the estimated values $I(x_0)$ at each x_0 location are calculated as a linear combination (Eq. 10) of the n sample (or input) data $I(x_i)$ located at the x_i points and contained in the search neighborhood, generally with an elliptical shape, whose weights λ_i reflects the spatial correlation between the estimated point and the sample values within the search neighborhood, and between the sample (or input) values among each other (Yamamoto and Landim [2013]).

$$I(x_0) = \sum_{i=1}^{n} \lambda_i I(x_i) \tag{10}$$

The estimated indicator variable values by ordinary kriging are determined in such a way that the variance of the estimation error is minimized, considering the unbiased constraint (average of the estimation error is zero), resulting in optimal weights-based estimations, which minimizes the estimation error variance (Isaaks and Srivastava [1989]).

3 Python packages applied to Kriging

The geostatistical methods and tools (e.g., kriging) have been implemented in several programming languages, including Python. Among the packages used to carry out kriging-based estimates, there is the Geostatspy (Pyrcz et al. [2021]), whose package is characterized to be reimplementations of the functionalities of the Geostatistics Software Library (GSLIB) in the Python language. The GSLIB is a set of pretty robust codes for building spatial modeling workflows. Therefore, this package offers the opportunity to create projects of 2-D spatial modeling in Python without the necessity of trust in the GSLIB's compiled Fortran code, together with the use of datasets that moves between the GSLIB's Geo-EAS format to Pandas' dataframes and grids to NumPy's n-dimensional arrays.

The operation of the Geostatspy package requires some dependencies, which includes the following libraries: (1) NumPy (Harris et al. [2020]), for n-dimensional arrays; (2) Pandas (McKinney [2010]), for DataFrames; (3) numpy.linalg, for linear algebra; (4) numba, for numerical acceleration; (5) scipy, for quickly nearest neighbor search, and; (6) Matplotlib (Hunter [2007]), for plotting.

The Geostatspy package contains *geostatspy.geostats* and *geostatspy.GSLIB*. The first includes GSLIB functions rewritten in Python, including all semivariograms, distribution transformations, and estimation and simulation-based methods. The second consists of the reimplementations of the GSLIB visualizations and low-tech wrappers of numerical methods, which require access to the GSLIB executables.

The first part, geostatspy.geostats, contains numerical methods, for example, the *ik2d* function (GSLIB's *ik3d* program implemented for indicator kriging estimation of 2-D data), *correct_trend* (fix the order relationships of an indicator-based trend model), and *backtr* (GSLIB's *backtr* program for back-transform a normal-shape distribution to an original one). These and other functions are presented and described in the Table 1.

Table 1: Some functions of the geostatspy.geostats part of the Geostatspy package

Function	Application
correct_trend	fix the order relationships of an indicator-based trend model
nscore	Transform an original distribution to a normal-shape one
backtr	Back-transform a normal-shape distribution to the original one
declus	Execution of sample's declustering (cell-based declustering)
gam	Calculation of semivariograms basing on regularly spaced 2-D data
gamv	Calculation of semivariograms basing on irregularly spaced 2-D data
varmapv	Semivariogram map of irregularly spaced 2-D data
vmodel	Semivariogram fitting
kb2d	Simple and/or ordinary kriging for 2-D data
ik2d	Indicator kriging for 2-D data

The second part, *geostatspy.GSLIB*, contains utility functions that support moves between data tables: DataFrames, n-dimensional arrays, Geo-EAS, and grid data. It also contains data transformation functions, spatial continuity, spatial model re-sampling, and visualization using Matplotlib. Some functions are showed and described in the Table A1, Appendix A.

4 Damage location prediction using Indicator Kriging – A Case Study

Herein it's considered an aluminum plate of 100 cm x 100 cm dimensions containing 100 RMSD-based damage metric values arranged in a regular mesh of 9.09 cm x 9.09 cm, with a known damage located at the plate's center. For the development of the case study it was used the python packages geostatspy (Pyrcz et al. [2021]), os-sys (Labots [2019]), NumPy (Harris et al. [2020]), Pandas (McKinney [2010]), pygeostat (Deutsch et al. [2020]), and Matplotlib (Hunter [2007]), which were imported according the Code 1.

Code 1: Source code for import the Python packages used in this case study

```python
import geostatspy.GSLIB as GSLIB
import geostatspy.geostats as geostats
import os
import numpy as np
import pandas as pd
import pygeostat as gs
import matplotlib.pyplot as plt
from matplotlib import gridspec
%matplotlib inline
```

In the sequence, it was selected the working directory using the *chdir* function of the *os-sys* package. The data, in the CSV format, was then imported using the Pandas' *read_csv* function. Subsequently, it was necessary to plot the RMSD values, which was done using the Code 2, resulting in the Figure 5, where one can observe the occurrence of high RMSD values near the damage, with those decreasing gradually from center to the plate's border.

Code 2: Source code for import the dataset and plot the RMSD values in X-Y plane

```
1  os.chdir('C:\\YOUR_DATABASE_PATH')
2  df=pd.read_csv('dataset.csv', header=0, sep=';')
3  xmin = 0.0; xmax = 100.0
4  ymin = 0.0; ymax = 100.0
5  xsiz = 4; ysiz = 4
6  nx = 25; ny = 25
7  xmn = 2; ymn = 2
8  cmap = plt.cm.viridis
9  GSLIB.locmap_st(df,'X','Y','PDANO',xmin,xmax,ymin,ymax,0,5,'Damage
      metric values','X (cm)','Y (cm)','Damage metric values',cmap)
10 plt.savefig('map_points.PDF', dpi=800, bbox_inches='tight')
11 plt.show()
```

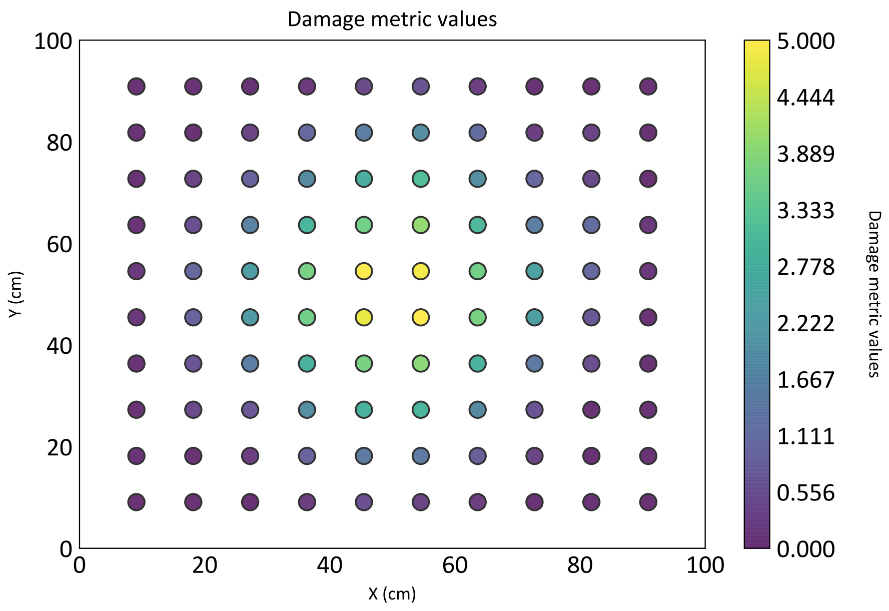

Figure 5: Localization map of the PZT patches in the plate

In the Code 2, the directory was defined in line 1, and the dataset, imported using the *read_csv* function, shown in line 2, where were added the file's name, the number of the header line, and the column's delimiter. Between lines 3 and 7 was defined the followed variables: minimum coordinate in the X direction (*xmin*); maximum coordinate in the X direction (*xmax*), minimum coordinate in Y (*ymin*), maximum coordinate in Y (*ymax*), cell size in the X direction (*xsiz*), cell size in the Y direction (*ysiz*), number of cells in X (*nx*), number of cells in Y (*ny*), origin of the grid in X (*xmn*), and origin of the grid in Y (*ymn*). In the sequence (line 8), the color palette was exposed. Then, using the geostatspy's function named *locmap_st* (line 9), the points (PZTs location) were plotted according to the parameters defined previously.

The statistical analysis of the RSMD values was done by means of boxplot (Figure 6), histogram (Figure 6), and descriptive statistical indicators (Table 2), which were obtained using the Code 3 (boxplot and histogram) and the Code 4 (descriptive statistics). According to the results (Figure 6 and Table 2), one can observe a highly asymmetric, positive skewed distribution, with only 25% of the data higher than 1.9, with presence of outliers, all of them higher than 4.0.

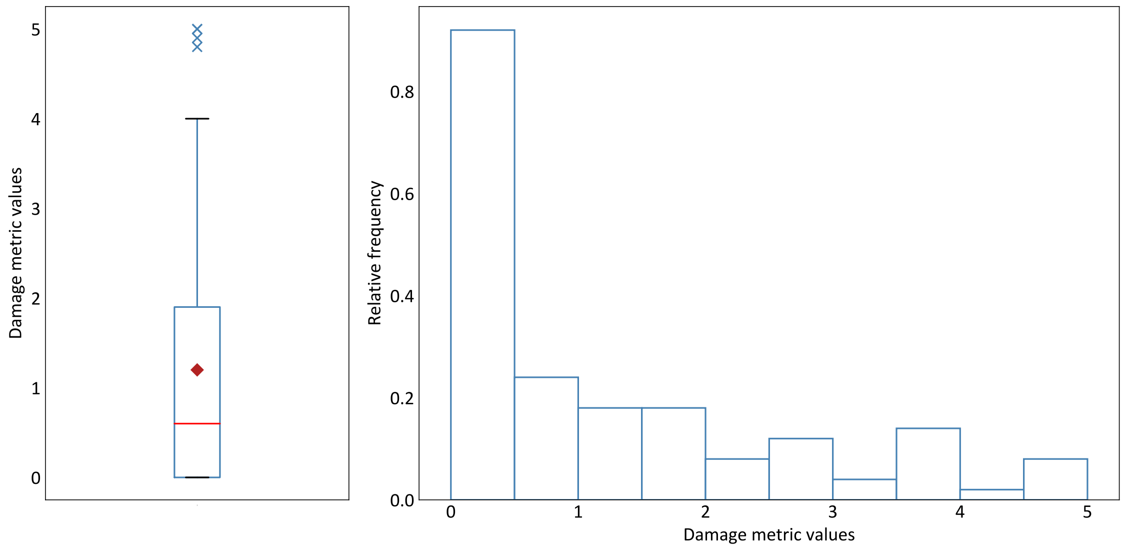

Figure 6: Boxplot and histogram of the RMSD values

Table 2: Descriptive statistics of the damage metric values.

Count	Min	Q1	Median	Mean	Q3	Max	STDEV	Variance	Skewness
100	0.000	0.000	0.600	1.199	1.900	5.000	1.414	1.998	1.151

Code 3: Source code to get the boxplot and histogram of the RSMD values

```
fig=plt.figure(figsize=(10,5))
gs=gridspec.GridSpec(1,2, width_ratios=[1.3,3])

ax1=plt.subplot(gs[0])
meanpointprops=dict(marker='D', markeredgecolor='black',
    markerfacecolor='firebrick')
flier_props=dict(marker='x', markerfacecolor='magenta', markeredgecolor
    ='steelblue')
box_props=dict(color='steelblue')
whisker_props=dict(color='steelblue')
median_props=dict(color='red')
ax1.boxplot(df['PDANO'], showmeans=True, meanprops=meanpointprops,
    flierprops=flier_props, boxprops=box_props, medianprops=
    median_props, whiskerprops=whisker_props)
ax1.tick_params(axis='x', labelsize=0)
ax1.tick_params(axis='y', labelsize=12)
ax1.set_ylabel('Damage metric values', fontsize=12)

ax2=plt.subplot(gs[1])
ax2.hist(df['PDANO'], density=True, facecolor='white', edgecolor='
    steelblue', linewidth=1)
ax2.set_xlabel('Damage metric values', fontsize=12)
ax2.set_ylabel('Relative frequency', fontsize=12)
ax2.tick_params(axis='x', labelsize=12)
ax2.tick_params(axis='y', labelsize=12)

plt.subplots_adjust(wspace=0.2, hspace=0.1)
```

```
23 fig.tight_layout()
24 plt.savefig('statistic_analysis.pdf', dpi=800, bbox_inches='tight')
25 plt.show()
```

Code 4: Source code for obtaining the statistical indicators of the results showed in Table 2

```
1 print(df['PDANO'].describe())
2 print(df['PDANO'].skew(skipna=True))
3 print(df['PDANO'].var())
```

In the Code 3, initially was defined the figure's size (line 1), and then the specifications of the grid, containing the boxplot and the histogram, using the *GridSpec* function (line 2). In the sequence (line 4), it was determined the position of the boxplot in the subplot grid (first position), followed by the boxplot properties relative to the mean (line 5), the outliers (line 6), the boxes (line 7), the whiskers (line 8) and median (line 9), which were used to make the boxplot (line 10), whose general formatting features were defined in the lines 11, 12 and 13. In line 15, it was defined the position of the histogram in the subplot grid (second position) and then the histogram (line 16), whose formatting features were stated between lines 17 and 20. Finally, the horizontal and vertical spacing between the plots was established in line 22, followed by the definition of the tight layout (line 23), the *savefig* function to save the figure (line 24), and a command line to show the plots (line 25).

The Code 4 shows the source code for obtaining the descriptive statistics of the RMSD values. In line 1 was used the *describe* function, which returned the following indicators: number of values; mean; standard deviation; minimum and maximum value, and; the first, second, and third quartiles. The skewness was gotten using the *skew* function (line 2) and the sample variance, the *var* function (line 3).

Based on the statistical information (Figure 6), where one can observe the occurrence of outliers, all of them higher than 4.0, it was done a localization map with only the values higher than 4.0 (Figure 7), obtained using the Code 5.

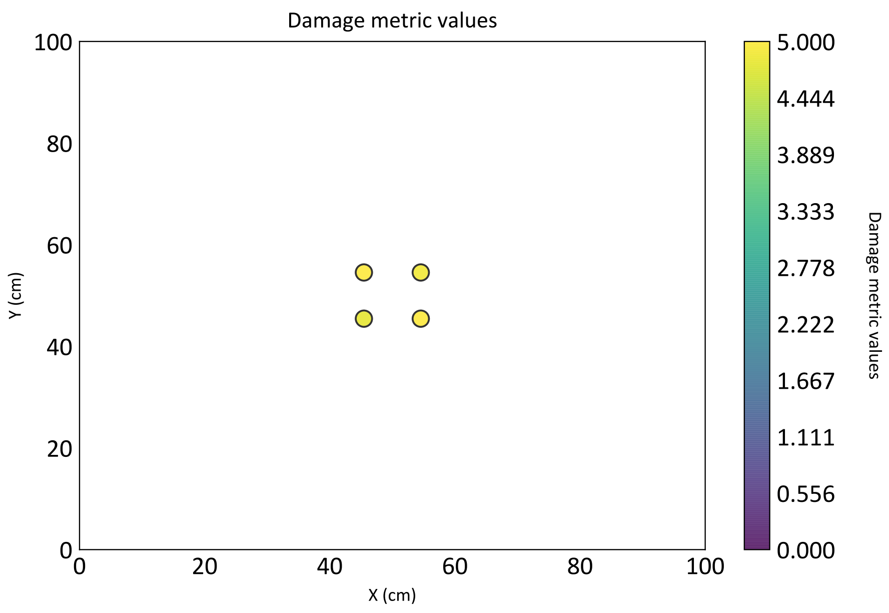

Figure 7: Localization map of the RMSD values higher than 4.0

Table 3: Thresholds and their respective indicator variables.

Threshold	Indicator Variable
1	IND0
2	IND1
3	IND2
4	IND3

Code 5: Source code for make the localization map of the RMSD values higher than 4.0

```
1  df_thr4=df[df['PDANO']>4]
2  df_thr4
3  xmin = 0.0; xmax = 100.0
4  ymin = 0.0; ymax = 100.0
5  xsiz = 4; ysiz = 4
6  nx = 25; ny = 25
7  xmn = 2; ymn = 2
8  cmap = plt.cm.viridis
9  GSLIB.locmap_st(df_thr4,'X','Y','PDANO',xmin,xmax,ymin,ymax,0,5,'Damage
       metric values','X (cm)','Y (cm)','Damage metric values',cmap)
10 plt.savefig('localization_map_thr_4.pdf', dpi=800, bbox_inches='tight')
11 plt.show()
```

In the Code 5, initially (line 1) it was defined a new dataset, containing only the RMSD values higher than 4.0. In the sequence, between lines 3 and 8, it was defined the plot's parameters: minimum (*xmin*) and maximum (*xmax*) coordinates in the X direction (line 3); minimum (*ymin*) and maximum (*ymax*) coordinates in the Y direction (line 4); cell sizes in the X (*xsiz*) and Y (*ysiz*) directions (line 5); number of cells in the X (*nx*) and Y (*ny*) directions (line 6); origin of the grid in the X (*xmn*) and Y (*ymn*) directions (line 7), and; color palette to be used (line 8). Finally, it was used the *locmap_st* function of the Geostatspy package in the line 9 to plot the data, whose resulting figure was saved using the *savefig* function (line 10).

Figure 7 shows an occurrence of values higher than 4.0 near the plate's center, where the known damage is located. Based on this, four thresholds were selected, 1, 2, 3, and 4, to map the occurrence of these values along the plate. The latter threshold (4.0) is particularly interesting, once it's related to the damage position. From the established thresholds, were created four indicator variables, each of them assigned to a specific threshold (Table 3), whose binary values were defined according to Eq. 9, implemented using the Code 6, and indicating the probability below a specific threshold.

Code 6: Source code for defining the indicator variables related to the stablished thresholds

```
1 thrs=[1,2,3,4]
2 for i in range(1,5):
3     cont=i
4     df["IND"+str(cont-1)]=np.zeros(len(df['PDANO']))
5     df["IND"+str(cont-1)]=np.where(df['PDANO']<=thrs[cont-1],1,0)
6 df
```

The Code 6 started with defining the threshold values (line 1). Then, a loop (line 2) was used for creating the indicator variables, all of them assuming initially zero values (line 4) and with the same length of the variable containing the RMSD values. In the sequence, it was used the *where* function of the *NumPy* package to apply the criterion defined in the Eq. 9, resulting in the probabilities below the respective thresholds.

To calculate the semivariograms, it was necessary, initially, to obtain the variance of each indicator variable (Table 4); once these values are, in most cases, the variance *a priori* in the semivariogram plot. This step was done with the Code 7, based on the standard deviation calculation and then the variance.

Table 4: Sample variance of each indicator variable.

Indicator Variable	Sample Variance
IND0	0.2356
IND1	0.1824
IND2	0.1131
IND3	0.0384

Code 7: Source code used for calculating the variance of each indicator variable

```
1 var = np.zeros(4)
2 for i in range(4):
3     cont = i
4     x = "IND"+str(cont)
5     var[i] = np.std(df[x])
6     var[i] = (var[i])**2
7     print(var[i])
```

According to Code 7, firstly it was created a variable named *var* (line 1) with a length equal to 4 to receive the variance value of each indicator variable. A *for* loop was then applied in line 2 to calculate all variances. Inside the loop, a variable named *cont* was defined (line 3), being equal to *i*, and in the sequence, it was defined a variable named *x* (line 4), which was equal to the respective indicator variable. Afterward, each value of the *var* variable was defined as the standard deviation of the respective indicator variable (line 5). Finally, these values were squared, resulting in the respective variances, shown in Table 4.

The results obtained for the indicator variables' sample variance showed that the highest variance is related to the indicator variable attributed to the threshold equal to 1, and the lowest, with the threshold equal to 4. This was expected once 62% of the samples were equal or lower than 1.0 (38% higher than 1.0), and 96% of the samples were equal or lower than 4 (only 4% higher than 4), indicating that the indicator variable related to the threshold equal to 1 would have a variability much higher than the other ones. This also means a spatial variability much higher for the indicator variable attributed to the threshold equal to 1 about the others. This characteristic would be manifested in the semivariogram model used to fit the experimental semivariograms in the next step.

The calculating of the experimental semivariograms was done using the Geostatspy's *gamv* function (see Section 3), which calculates the semivariogram for irregularly spaced data (general case). Firstly, an example of the calculation of the semivariogram for the IND0 variable, related to the threshold equal to 1.0, in the North-South direction (Azimuth equal to 0°), along with its variographic fitting. The calculation and fitting of this semivariogram were done using the Code 8, resulting in the semivariogram shown in Figure 8.

Code 8: Source code for calculating the experimental semivariogram for the IND0 variable in the North-South direction with its fitting by a theoretical model

```
1  azi=0
2  lag_dist=9.09
3  lagtol=2
4  nlag=8
5  bandh=10
6  atol=10
7  isill=var[0]
8  tmin=-1.0e21
9  tmax=1.0e21
10 lag0, por_gamma0, por_npair0 = geostats.gamv(df,"X","Y","IND0",tmin,
       tmax,lag_dist,lagtol,nlag,azi,atol,bandh,isill)
11 plt.plot(lag0[2:-1], por_gamma0[2:-1], color=’darkblue’, marker=’o’,
       linestyle=’dashed’,linewidth=1, markersize=5,label = ’Experimental
       Semivariogram’)
12 plt.xlim(0,80)
13 plt.ylim(0,0.4)
14 plt.plot([0,100],[var[0],var[0]], color=’#238A8DFF’, linewidth=1,
       linestyle=’dashed’, label=’Variance a priori’)
15
16 nugg=0.02
17 nst=1
18 it11=1
19 cc11=var[0]-nugg
20 azi1=0
21 hmaj11=52
22 hmin11=52
23 nlag1=160
24 xlag=0.5
25 vario_IND0=GSLIB.make_variogram(nugg,nst,it11,cc11,azi1,hmaj11,hmin11)
26 index_IND0_azi0, h_IND0_azi0, gam_IND0_azi0, cov_IND0_azi0,
       ro_IND0_azi0 = geostats.vmodel(nlag1,xlag,azi1,vario_IND0)
27 plt.plot(h_IND0_azi0,gam_IND0_azi0, color=’darkblue’, label=’
       Theoretical Fitting (spherical model)’)
28 plt.xlabel(’Distance (cm)’, fontsize=10)
29 plt.ylabel(’Semivariance’, fontsize=10)
30
31 plt.legend(fontsize=12, loc=’upper left’)
32 plt.savefig(’IND0_semivariogram_az0.pdf’, dpi=800, bbox_inches=’tight’)
33 plt.show()
```

The Code 8 starts with the definition of the parameters for calculation of the experimental semivariogram, which were: azimuth direction (line 1); lag separation distance (line 2), usually equal to the lowest spacing between sample pairs (in a regular grid, similar to the sample spacing); lag tolerance (line 3), which should be lower than 50% of the lag separation value; number of lags in the direction (line 4), being equal to the number of lags to cover at least the half of the field being analyzed; horizontal bandwidth (line 5), which can be the sample spacing in the perpendicular direction; angular tolerance (line 6), being lower than 90°, once angular tolerance equal to 90° results in the unidirectional semivariogram; sill of the semivariogram (line 7), usually equal to the variance *a priori*; the minimum value to be considered (line 8), in such a way that values lower than the minimum are ignored, and; the maximum value to be considered (line 9), in such a way that values higher than the maximum are ignored in the semivariogram calculation.

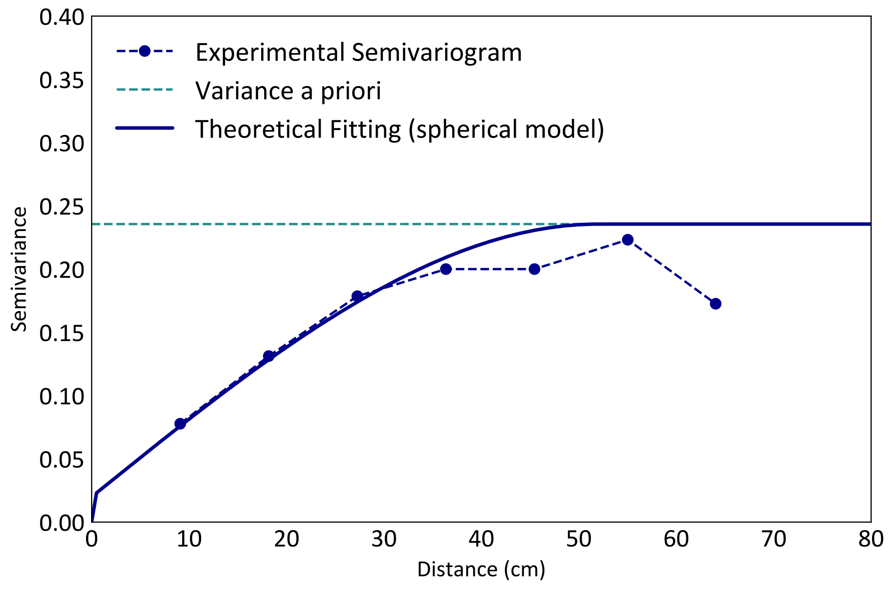

Figure 8: Experimental and theoretical fitting of the semivariogram of the IND0 variable in the North-South direction (Azimuth equal to 0°)

Based on the parameters defined between lines 1 and 9 of the Code 8, the experimental semivariogram was calculated using Geostatspy's *gamv* function in line 10, resulting in three variables: classes of distances (*lag0* variable), semivariance values (*por_gamma0* variable), and the number of pairs (*por_npair0* variable) used of obtaining each point of the experimental semivariogram. In the sequence, the semivariance and their respective classes of distances were plotted (line 11) using the *plot* function, where only the semivariance values were higher than zero using a filter. Subsequently, the limits of the plot (lines 12 and 13) were plotted, using line 14, the variance *a priori* (normally the semivariogram's sill) together with the experimental semivariogram.

The experimental semivariogram needs to be fitted by theoretical models to firstly identify the preferential directions of continuity, which would be the directions of highest and lowest range values, and then; provide spatial continuity information (based on such preferential directions) for the kriging equations' solving (see Section 2.2.1), resulting in the kriging estimates.

The parameters of the semivariogram's fitting were defined between the lines 16 and 24 of the Code 8, which were: (1) Nugget Effect (variable *nugg* in the line 16), which approximately refers to the interception in the Y axis; (2) position of the nested structure (*nst* in the line 17), in this case, it's the first nested structure; (3) code for the model type (*it11* in the line 18), which can be 1, 2, 3, 4 or 5 for spherical, exponential, Gaussian, Power law, or Cosine hole effect models, respectively; (4) contribution of the structure for the semivariogram's sill (*cc11* in the line 19), which was, in this case, the sill minus the nugget effect; (5) azimuth direction of the semivariogram's major axis (*azi1* in the line 20), in this case, 0°; (6) range of the structure in the major direction (*hmaj11* in line 21); (7) range of the structure in the minor direction (*hmin11* in the line 22); (8) number of lags (*nlag1* in the line 23), and; (9) lag separation, defined in the variable *xlag*, line 24.

Based on the aforementioned parameters, it was obtained the semivariogram model by means of the *make_variogram* function, described in line 25, whose model was attributed to a variable named *vario_IND0*. To determine the full information for plotting the semivariogram's model, the variable *vario_IND0* was used in the *vmodel* function (line 26), resulting, among others, in the classes of distances (variable *h_IND0_azi0* in line 27) and the theoretical semivariogram values (variable *gam_IND0_azi0* in line 27), which were used for the semivariogram model plotting (line

27). The resulting figure was, then, saved in the PDF format using Matplotlib's *savefig* function (line 32).

Based on the steps for semivariogram calculation, it was obtained experimental semivariograms for several directions for all indicator variables in the form of a report with the plots. This was done using Code 9, in such used loops, automating the semivariogram calculation, and filtering the results obtained in such a way to plot only those that showed a spatial continuity.

Code 9: Source code for obtain the experimental semivariogram of all indicator variables and plotting only those ones which demonstrates a spatial continuity

```
1  lag_dist = np.zeros(4)
2  azi = [0,30,60,90,120]
3  color = [’red’,’green’,’pink’,’darkblue’,’black’]
4  lag = [np.zeros(1)]*20
5  por_gamma = [np.zeros(1)]*20
6  por_npair = [np.zeros(1)]*20
7  tmin = -1.0e21; tmax = 1.0e21;
8  lag_tol = 1; nlag = 8; bandh = 10; atol = 10;
9
10 k = 0
11 for i in range(4):
12     cont = i
13     ind = "IND"+str(cont)
14     plt.figure(figsize=(7, 4))
15     plt.xlabel(r’Lag Distance $\bf(h)$, (cm)’)
16     plt.ylabel(r’$\gamma \bf(h)$’)
17     plt.title(’Variogram ’+ind+’ 10x10’)
18     plt.plot([0,2000],[var[i],var[i]],color = ’#238A8DFF’,linewidth=1,
    linestyle=’dashed’)
19     plt.xlim(0,80)
20     plt.ylim(0,0.42)
21     for j in range(0,5):
22         lag_dist[i] = 9.09;  azi[j]; isill = var[i]
23         lag0, por_gamma0, por_npair0 = geostats.gamv(df,"X","Y",ind,
    tmin,tmax,lag_dist[i],lag_tol,nlag,azi[j],atol,bandh,isill)
24         lag[k] = lag0; por_gamma[k] = por_gamma0; por_npair[k] =
    por_npair0;
25         filtro = lag[k] != 0; filtro2 = por_gamma[k]!= 0;
26         aux1 = lag[k][filtro]; aux2 = por_gamma[k][filtro2]
27         if len(aux1)==len(aux2):
28             lag[k] = aux1;por_gamma[k] = aux2
29         if len(lag[k])>2:
30             plt.plot(lag[k][0:len(lag[k])], por_gamma[k][0:len(
    por_gamma[k])], color=color[j], marker=’o’, linestyle=’dashed’,
    linewidth=1,markersize=5,label = ’Porosity’)
31             print(’Preferential Direction:’,azi[j],’;cor:’,color[j],’;
    lag[’+str(k)+’], por_gamma[’+str(k)+’]’)
32         k +=1
33     plt.show()
```

In Code 9, firstly, it was defined the arrays to receive the lag separation values (*lag_dist* in line 1), the azimuth directions (*azi* in line 2), the color for each direction (*color* in line 3), the classes of distances (*lag* in line 4), the semivariance values (*por_gamma* in line 5), and the number of pairs for each class of distance (*por_npair* in line 6). In the sequence, it was defined the trimming limits (*tmin* and *tmax* in line 7), the lag tolerance (*lag_tol* in line 8), the number of lags (*nlag* in line 8), the horizontal bandwidth (*bandh* in line 8), and the angular tolerance (*atol* in line 8), all of them used as parameters for the semivariogram calculation.

The information obtained by the experimental semivariograms was used for the semivariogram fitting. Theoretical models fitted those associated with the preferential directions of continuity (highest and lowest range values). The fitting was executed using the *make_variogram* function and then the *vmodel* function for each direction and indicator variable. Taking as an example, the source code used to plot the experimental semivariograms and their respective fittings for the threshold equal to 1 (IND0) is shown in the Code 10, and the resulting plot, in the Figure 9, where one can observe the existence of two semivariogram models (continuous lines), one for each experimental semivariogram.

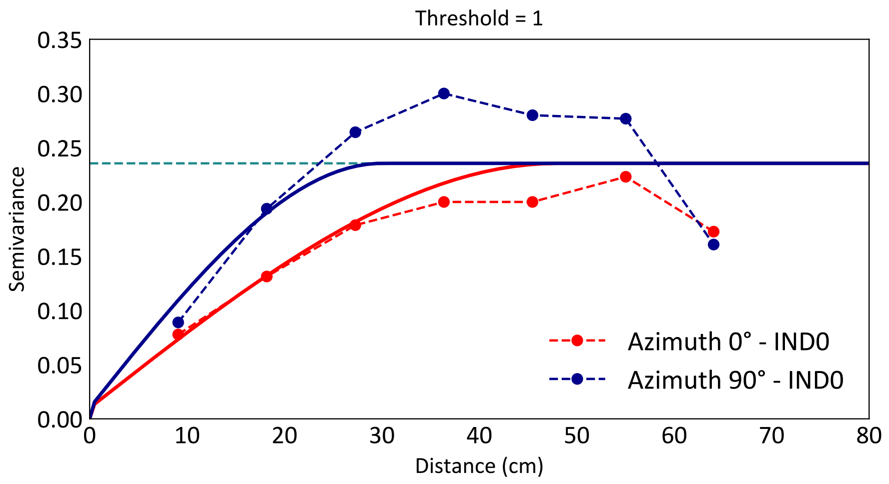

Figure 9: Plot of the experimental semivariograms and its respective fittings in the preferential directions of continuity for the indicator variable related to the threshold equal to 1

The Code 10 starts with determining the figures and the subplots' sizes (lines 1 and 2). In line 4, the position of this specific plot was defined in a grid, in this case, composed of only one plot, using the *subplot2grid* function of the Matplotlib package. Between lines 5 and 10, it was defined the overall parameters of the plot: Position of the variance a priori of the indicator variable under analysis, limits in the X-axis, limitations in the Y-axis, label for the X-axis, label for the Y-axis, and plot's title, respectively. The experimental semivariograms were plotted using the function *plot* (lines 11 and 12) with the respective legends (line 13). In the sequence, the semivariogram models were determined and then plotted, whose procedures and parameters are described between the lines 15 and 20 for the azimuth 0°and between the lines 22 and 27 for the azimuth 90°. Lastly, the figure was saved in PDF format using the *savefig* function (line 29).

Applying similar procedures to that described in the Code 10 for the other semivariogram plots, it was obtained the results shown in the Figure 10, whose fitting parameters used are shown in Table 5.

The analysis of the range values in the experimental semivariograms for each indicator variable (Figure 10 and Table 5) showed the occurrence of geometric anisotropy (differences in the range values, with equal sill), for the thresholds equal to 1 and 3. The preferential directions for these thresholds were the 0° (North-South) and 90° (East-West) azimuths, with North-South being the major axis. The occurrence of anisotropy wasn't identified in the other indicator variables, suggesting an isotropic phenomenon for these.

Code 10: Source code for plotting the experimental semivariogram and its respective fitting for the threshold equal do 1

```
plt.figure(figsize=(6,3))
fig, axs = plt.subplots(1, 1, figsize=(6,3), constrained_layout=True)

plt.subplot2grid((1,1),(0,0))
plt.plot([0,200],[var[0],var[0]], linestyle='dashed', linewidth=1,
    color='#238A8DFF')
plt.xlim(0,80)
plt.ylim(0,0.35)
plt.xlabel('Distance (cm)', fontsize=10)
plt.ylabel('Semivariance', fontsize=10)
plt.title('Threshold = 1', fontsize=10)
plt.plot(lag[0][0:len(lag[0])], por_gamma[0][0:len(por_gamma[0])],
    color='red', marker='o', linestyle='dashed',linewidth=1, markersize
    =5,label = 'Azimuth 0Â° - IND0')
plt.plot(lag[3][0:len(lag[3])], por_gamma[3][0:len(por_gamma[3])],
    color='darkblue', marker='o', linestyle='dashed',linewidth=1,
    markersize=5,label = 'Azimuth 90Â° - IND0')
plt.legend(fontsize=12)

nugg[0] = 0.01; nst = 1; it11[0] = 1; cc11[0] = var[0]-nugg[0];
azith[[0],[0]] = 0; hmaxmin11[[0],[0]]=48; hmaxmin11[[0],[0]] = 48
vario_IND0_az0 = GSLIB.make_variogram(nugg[0],nst,it11[0],cc11[0],azith
    [[0],[0]],hmaxmin11[[0],[0]],hmaxmin11[[0],[0]])
nlag = 160; xlag = 0.5;
index_IND0_az0, h_IND0_az0, gam_IND0_az0, cov_IND0_az0, ro_IND0_az0 =
    geostats.vmodel(nlag, xlag,azith[[0],[0]], vario_IND0_az0)
plt.plot(h_IND0_az0,gam_IND0_az0,color='red')

nugg[0] = 0.01; nst = 1; it11[0] = 1; cc11[0] = var[0]-nugg[0];
azith[[0],[1]] = 90; hmaxmin11[[0],[1]] = 30; hmaxmin11[[0],[1]] = 30
vario_IND0_az90 = GSLIB.make_variogram(nugg[0],nst,it11[0],cc11[0],
    azith[[0],[1]],hmaxmin11[[0],[1]],hmaxmin11[[0],[1]])
nlag = 160; xlag = 0.5;
index_IND0_az90, h_IND0_az90, gam_IND0_az90, cov_IND0_az90,
    ro_IND0_az90 = geostats.vmodel(nlag, xlag,azith[[0],[1]],
    vario_IND0_az90)
plt.plot(h_IND0_az90,gam_IND0_az90,color='darkblue')

plt.savefig('Exp_adjusted_semivariogram_thr_1.pdf', dpi=800,
    bbox_inches='tight')
plt.show()
```

The existence of preferential directions of continuity can indicate a directional propagation of the failure since the RMSD values vary more gradually in a direction in relation to the other ones. Therefore, this information, only derived from semivariograms, can be helpful in mapping this aspect.

The results of the semivariograms' fitting (Table 5) show the presence of two types of models: spherical (threshold equal to 1) and Gaussian (thresholds equal to 2, 3, and 4). Since the Gaussian model indicates a more continuous (homogeneous) relation to the spherical one, the results show a highly constant phenomenon for the thresholds equal to 2, 3, and 4, also occasioned by the absence of the nugget effect in these. Because 62% of the RMSD values are lower than 1, the indicator kriging results for the threshold equal to 1 may be characterized to have a spatial variability much higher than in the other ones, reflected by the semivariogram's shape (fitted by spherical model), and their nugget effect, which represents 4,06% of the sill. Furthermore, the fitting of the indicator semivariogram attributed to the threshold equal to 4 (Figure 10) was based on only one point of the experimental semivariogram. However, this is an expected result once semivariograms based on indicator variables related to values situated on the distribution's upper tail may result in deteriorated semivariograms, with few points attributed to the structured region (distances lower than the range).

The input dataset and the information derived from the semivariogram fitting were used to obtain the estimates by the indicator kriging method. In this step, initially, it was necessary to define the general parameters for estimating, described in the Code 11.

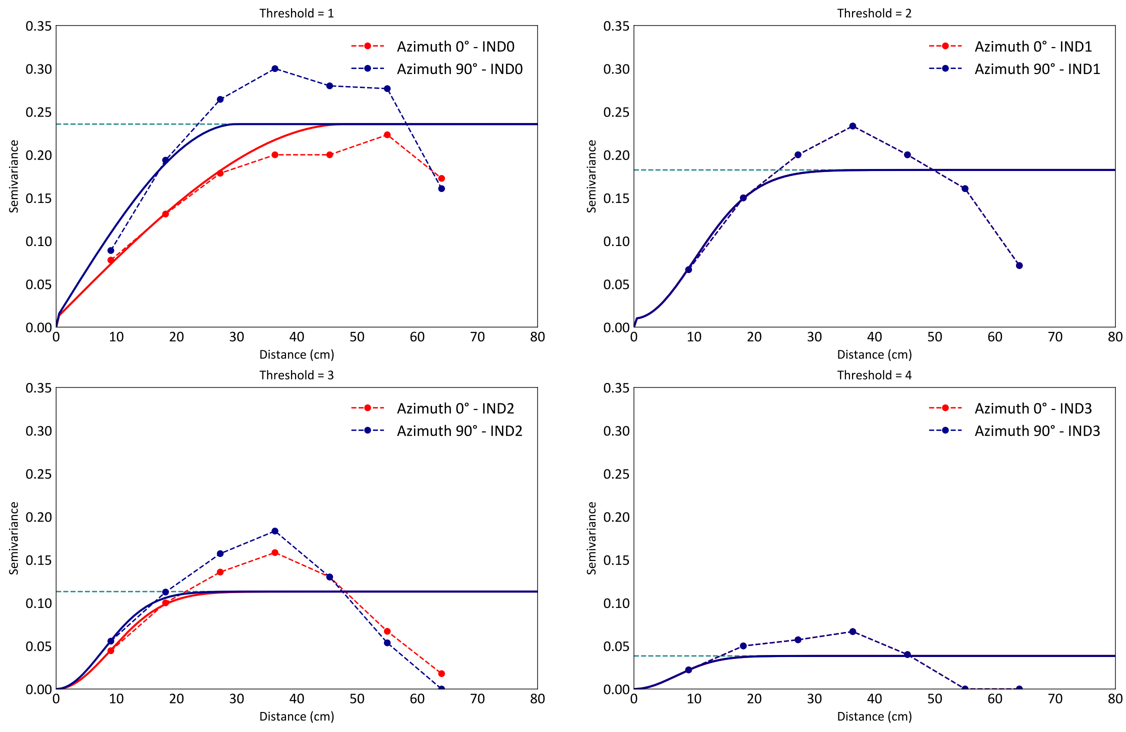

Figure 10: Experimental semivariograms in several directions for all indicator variables

Table 5: Parameters of the indicator semivariograms' fitting.

Threshold	Indicator Variable	Orientation	Nugget Effect	Structure 1		
				Model	Contribution	Range (cm)
1	IND0	Azimuth 0°	0.01	spherical	0.236	48.0
		Azimuth 90°	0.01	spherical	0.236	30.0
2	IND1	Azimuth 0°	0.01	Gaussian	0.182	24.5
		Azimuth 90°	0.01	Gaussian	0.182	24.5
3	IND2	Azimuth 0°	0.00	Gaussian	0.114	22.0
		Azimuth 90°	0.00	Gaussian	0.114	19.0
4	IND3	Azimuth 0°	0.00	Gaussian	0.039	17.0
		Azimuth 90°	0.00	Gaussian	0.039	17.00

Code 11: Source code for determining general parameters to obtain the kriging estimates

```
1  xmin = 0.0; xmax = 100.0
2  ymin = 0.0; ymax = 100.0
3  xsiz = 4; ysiz = 4
4  nx = 25; ny = 25
5  xmn = 2; ymn = 2
6  nxdis = 3; nydis = 3
7  ndmin = 1; ndmax = 4
8  radius = lag_dist[0]
9  ktype = 1
10 ivtype = 1
11 tmin = -999; tmax = 999;
12 vmin = 0.0; vmax = 1.0;
13 ncut = 4
14 pdano = np.array(df['PDANO'])
15 cont1 = 0; cont2 = 0; cont3 = 0; cont4 = 0
16 th1 = 1; th2 = 2; th3 = 3; th4 = 4
17 for i in pdano:
18     if i<=th1:
19         cont1 += 1
20     if i<=th2:
21         cont2 += 1
22     if i<=th3:
23         cont3 += 1
24     if i<=th4:
25         cont4 += 1
26
27 gcdf1 = (cont1/len(pdano))
28 gcdf2 = cont2/len(pdano)
29 gcdf3 = cont3/len(pdano)
30 gcdf4 = cont4/len(pdano)
31 print(gcdf1, gcdf2, gcdf3, gcdf4)
```

In the Code 11 it was necessary to define, initially, the block model's parameters: Minimum and maximum coordinates in the X-axis (line 1); Minimum and maximum coordinates in the Y-axis (line 2); cell's dimensions in the X and Y directions (line 3); the number of cells in the X and Y axis (line 4), and; origin coordinates of the model (line 5). In the sequence, it was specified the general parameters for the kriging process: The number of points to be estimated inside each cell in the X and Y directions (line 6), in such a way that the estimated value in each cell is the average of the estimated points in each one; the minimum and a maximum number of input data inside the search ellipse to be used (line 7); maximum search distance (line 8); kriging estimator to be used, which can be *0* for simple kriging or *1* for ordinary kriging (line 9); type of the variable being estimated, which can be *0* for categorical variable or *1* for a continuous one (line 10); trimming limits (line 11); minimum and maximum value for legend displaying (line 12), and; the number of thresholds (line 13).

Lastly, still in the Code 11, it was calculated the global cumulative distribution function of the variable *PDANO* (lines 14 to 31) related to the damage metric values for the threshold values defined previously (1, 2, 3, and 4), which was executed using a *for* loop to count the number of data lower than the respective threshold value, and then, dividing this amount by the total number of data in the *PDANO* variable, resulting in the following global cumulative distribution function: 0.62, 0.76, 0.87 and 0.96, related to the thresholds equal to 1, 2, 3 and 4, respectively.

The cell's sizes were defined in such a way due to the following: (1) the cell's dimensions are lower than all ranges in the semivariogram models, and; (2) the cell's sizes are multiple of the plate's dimensions in each direction, allowing an adequate adjustment of the model about the actual dimensions of the plate. Also, the chosen cell's dimensions define the model's resolution (40 cm).

Regarding the kriging estimator used to obtain the estimates (parameter *ktype* in Code 11), it was used the ordinary kriging one, due to its capability to reproduce more properly the local variations of the variable being estimated (local estimation method).

In the sequence, it was defined lists for storing some information for the *ik2d* function of the Geostatspy package, shown in the Code 12.

Code 12: Source code for defining the list objects for the *ik2d* process

```
1 thresh = [th1,th2,th3,th4]
2 gcdf = [gcdf1, gcdf2, gcdf3, gcdf4]
3 varios = []
4 varios.append(GSLIB.make_variogram(nug=nugg[0],nst=1,it1=it11[0],cc1=
     cc11[0],azi1=azith[[0],[0]],hmaj1=hmaxmin11[[0],[0]],hmin1=
     hmaxmin11[[0],[1]]))
5 varios.append(GSLIB.make_variogram(nug=nugg[1],nst=1,it1=it11[1],cc1=
     cc11[1],azi1=azith[[1],[0]],hmaj1=hmaxmin11[[1],[0]],hmin1=
     hmaxmin11[[1],[1]]))
6 varios.append(GSLIB.make_variogram(nug=nugg[2],nst=1,it1=it11[2],cc1=
     cc11[2],azi1=azith[[2],[0]],hmaj1=hmaxmin11[[2],[0]],hmin1=
     hmaxmin11[[2],[1]]))
7 varios.append(GSLIB.make_variogram(nug=nugg[3],nst=1,it1=it11[3],cc1=
     cc11[3],azi1=azith[[3],[0]],hmaj1=hmaxmin11[[3],[0]],hmin1=
     hmaxmin11[[3],[1]]))
8 no_trend = np.zeros((1,1,1,1))
9 ikmap = ik2d(df,'X','Y','PDANO',ivtype,0,4,thresh,gcdf,no_trend,tmin,
     tmax,nx,xmn,xsiz,ny,ymn,ysiz,ndmin,ndmax,radius,ktype,vario=varios)
```

The Code 12 starts with the definition of the following list objects: Thresholds to be used (line 1); a global cumulative distribution function for each threshold (line 2), and; a list for receiving the semivariogram fitting information, which was specified in the sequence, between the lines 4 and 7, and appended in a single list named *varios*. In the sequence, an array of ones was created to indicate the absence of trends (line 8). Then, it was defined a variable named *ikmap* for storage of the results of the *ik2d* process, whose parameters are described in line 9 of the Code 12, where each of these is linked to a specific variable defined in the previous steps.

The indicator kriging results are in the form of *probability below thresholds* due to the procedure performed to create the indicator variables (Eq. 9). To represent the results more appropriately, the results were converted to *probability above thresholds* by applying the Eq. 11, subtracting each probability ($I(x_0)$) by one (maximum probability), resulting, then, in the probabilities of each estimated cell (x_0 location) to be higher than the respective threshold.

$$I_{Above}(x_0) = 1 - I(x_0) \tag{11}$$

The Eq. 11 was applied to the indicator kriging results and each input indicator variable, aiming to represent them in the form of *probability above thresholds*. This was done by adapting the original *locpix_st* function of the Geostatspy package, whose part of the source code is shown in the Code 13.

Code 13: Part of the *locpix_st* function's source code where the adaptations were made

```
65      array = 1-array
66      array[len(array)-1,len(array)-1] = 0
67      array[len(array)-1, 0] = 0
68      array[0, len(array)-1] = 0
69      array[0, 0] = 0
70      cs = plt.contourf(
71          xx,
72          yy,
73          array,
74          cmap=cmap,
75          vmin=vmin,
76          vmax=vmax,
77          levels=np.linspace(vmin, vmax, 100),
78      )
79
80      plt.scatter(
81          df[xcol],
82          df[ycol],
83          s=None,
84          c=1-df[vcol],
85          marker=None,
86          cmap=cmap,
87          vmin=vmin,
88          vmax=vmax,
89          alpha=0.8,
90          linewidths=0.8,
91          verts=None,
92          edgecolors="black",
93      )
```

The conversion of the *probabilities below thresholds* to the *probabilities above thresholds* in the indicator kriging results was done using the Eq. 11, applied in line 65 of the Code 13. In the sequence, the map was centralized in the plot using the procedures shown between lines 66 and 69 since the map was slightly displaced in the original function. The last modification was related to the input indicator variable, subtracted from 1 (maximum probability), done using the procedure shown in line 84. Therefore, all data plotted using the *locpix_st* function was in the form of *probability above thresholds*. The indicator kriging results and the respective input data were plotted using the function above, resulting in the plots shown in Figure 11, and the source code used to generate these are displayed in the Code 14.

For plotting the estimated values and the input data, it was used the *locpix_st* function of the Geostatspy package, as shown in the Code 14, starts with the addition of the information (estimated and input data) related to the threshold equal to 1 (lines 1 to 2). Similarly, the other plots were added by the *subplot* function. In each *locpix_st* function, it's necessary to add the succeeding information, ordered in the following sequence: an array containing the indicator kriging results to be plotted; minimum coordinate in the X-axis; maximum coordinate in the X-axis; minimum coordinate in the Y direction; maximum coordinate in the Y direction; cell's size in the X-axis; minimum value for legend; maximum value for legend; the name of the dataset (in the format of a dataframe) containing the input values; the name of the column in the dataset storing the coordinates in the X-axis; the name of the column in the dataset storing the coordinates in the Y-axis; the name of the variable in the input dataset to be plotted, in this case, the indicator variables; plot's title; label in the X-axis; label in the Y-axis; legend's title, and; color palette to be used.

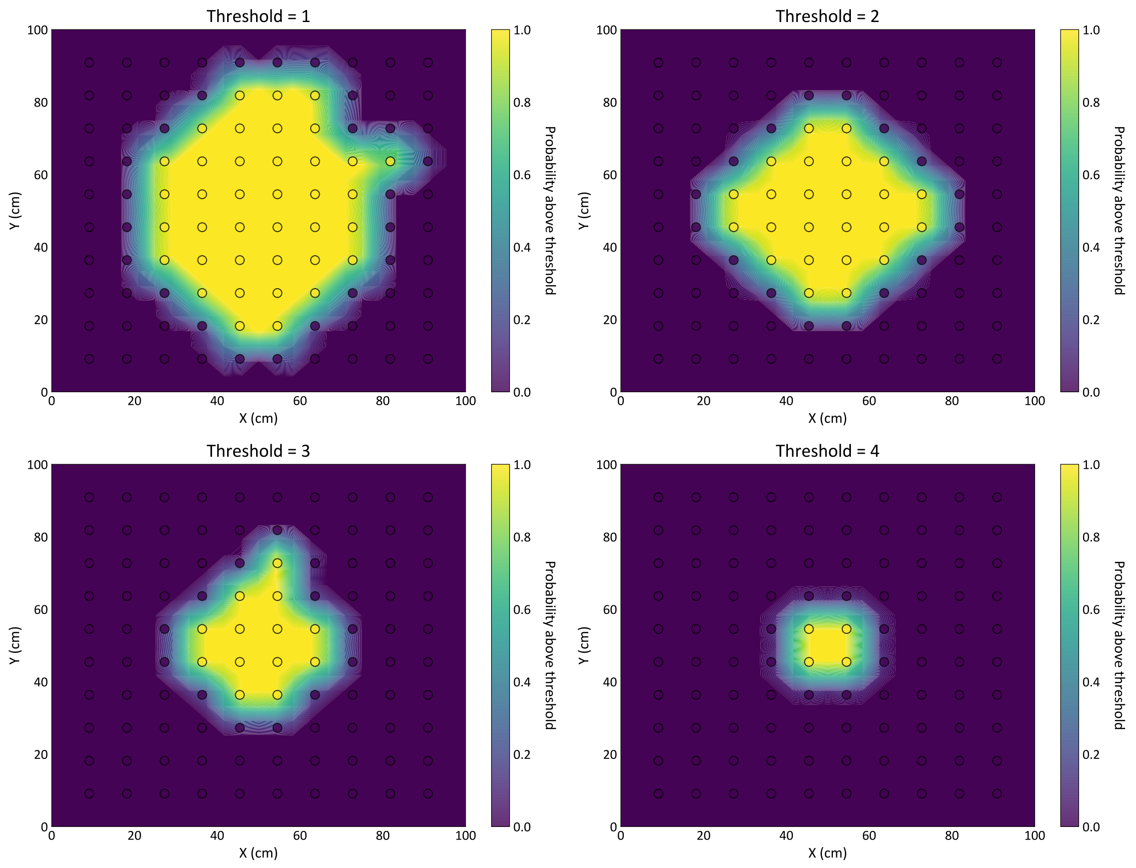

Figure 11: Probability maps of the damage metric values to be higher than each defined threshold, jointly with the respective input data

Code 14: Source code to plot the estimated and the input values of probability above thresholds

```
plt.subplot(221)
locpix_st(ikmap[:,:,0],xmin,xmax,ymin,ymax,xsiz,0.0,1.0,df,'X','Y','
    IND0','Threshold = 1','X (cm)','Y (cm)','Probability above
    threshold',cmap)
plt.subplot(222)
locpix_st(ikmap[:,:,1],xmin,xmax,ymin,ymax,xsiz,0.0,1.0,df,'X','Y','
    IND1','Threshold = 2','X (cm)','Y (cm)','Probability above
    threshold',cmap)
plt.subplot(223)
locpix_st(ikmap[:,:,2],xmin,xmax,ymin,ymax,xsiz,0.0,1.0,df,'X','Y','
    IND2','Threshold = 3','X (cm)','Y (cm)','Probability above
    threshold',cmap)
plt.subplot(224)
locpix_st(ikmap[:,:,3],xmin,xmax,ymin,ymax,xsiz,0.0,1.0,df,'X','Y','
    IND3','Threshold = 4','X (cm)','Y (cm)','Probability above
    threshold',cmap)
plt.subplots_adjust(left=0.0, bottom=0.0, right=2.0, top=2.2, wspace
    =0.1, hspace=0.2)
plt.savefig('IK2D_map.pdf', dpi=800, bbox_inches='tight')
plt.show()
```

The resulting plots were adjusted using the *subplots_adjust* function (line 9) and then saved in the PDF format using the *savefig* function (line 10), resulting in the plots shown in Figure 11.

Figure 11 shows the probability maps of the RMSD values being higher than their respective thresholds. The results for the threshold equal to 1 exhibit a wide area of high probability, indicating an eventual area affected by the damage. Also, the elongated shape of the area where these high values occur can mean the existence of locations more impacted by the damage than others, providing information for supporting the decision-making process in condition-based maintenance programs.

Since the damage is located at the plate's center, as the thresholds increases, the highest probabilities of the RMSD values to be greater than each threshold are sequentially restricted to the plate's center, converging the existence of these to this location, where the RMSD values were higher than 4.0. Therefore, by mapping the RMSD values to be higher than four, it was possible to predict the structural damage's location.

In the sequence, the average of the probabilities below thresholds in the indicator kriging results and in the input data (indicator variables) was calculated to compare them, analyzing the adherence of the estimates about the respective input indicator variables. The averages were calculated using the Code 15.

The Code 15 starts with the creation of the variables: The mean value of each indicator variable (*mean_inds* in line 1), the mean value of each indicator kriging-based estimated probability (*mean_iks* in line 2), the percentage deviation of the estimated probabilities about the input ones (*desv_means_percent* in line 3). In the sequence, a *for* loop (lines 4 to 7) was used to calculate the means and the percentage deviations, each related to a specific threshold value. Lastly, the resulted percentage deviations were formatted to a maximum decimal place of 2, done by the *format* function in a *for* loop (lines 10 and 11).

The average of the probabilities below thresholds obtained by the indicator kriging approach was compared to its respective input indicator variable, resulting in the plot shown in Figure 12A, whose percentage differences are plotted in Figure 12B. The source code used for preparing these is shown in the Code 16.

Code 15: Source code for calculating of the estimate's and input indicator variables' averages

```
1  mean_inds=np.zeros(len(thrs))
2  mean_iks=np.zeros(len(thrs))
3  desv_means_percent=np.zeros(len(thrs))
4  for i in range(0,len(thrs)):
5      mean_inds[i]=np.mean(df['IND'+str(i)])
6      mean_iks[i]=np.mean(ikmap[:,:,i])
7      desv_means_percent[i]=((mean_iks[i]-mean_inds[i])/mean_inds[i])*100
8  
9  desv_means_percent_format=np.zeros(len(desv_means_percent))
10 for i in range(0,len(desv_means_percent)):
11     desv_means_percent_format[i]="{:.2f}".format(desv_means_percent[i
      ],2)
```

Code 16: Source code for plotting the average of the probabilities below thresholds in the estimated and input data, and the percentage differences between these ones

```
1  fig = plt.figure(figsize=(14,5))
2  plt.subplot(121)
3  plt.plot([1,2,3,4],mean_inds,label='Input data',marker='o', color='
       #440154ff')
4  plt.plot([1,2,3,4],mean_iks, label='Estimated data', marker='o', color=
       '#22A884FF')
5  plt.legend(loc='lower right',fontsize=12)
6  plt.xlabel('Threshold value', fontsize=12)
7  plt.ylabel('Mean', fontsize=12)
8  plt.ylim(0,1)
9  plt.xticks([1,2,3,4])
10 plt.text(1,0.92,'A',fontsize=16,fontweight='bold')
11
12 plt.subplot(122)
13 plt.bar([1,2,3,4],desv_means_percent,color='#2A788EFF')
14 plt.xticks([1,2,3,4])
15 plt.xlabel('Threshold value', size=12)
16 plt.ylabel('Deviation (%)', size=12)
17 plt.ylim(0,12)
18 plt.text(0.6,11,'B',fontsize=16,fontweight='bold')
19 plt.text(0.9,10,''+str(desv_means_percent_format[0])+'', fontsize=12)
20 plt.text(1.9,5,''+str(desv_means_percent_format[1])+'', fontsize=12)
21 plt.text(2.9,2.1,''+str(desv_means_percent_format[2])+'', fontsize=12)
22 plt.text(3.9,0.25,''+str(desv_means_percent_format[3])+'', fontsize=12)
23
24 plt.tight_layout()
25 plt.savefig('difference_results_input.pdf', dpi=800, bbox_inches='tight
       ')
26 plt.show
```

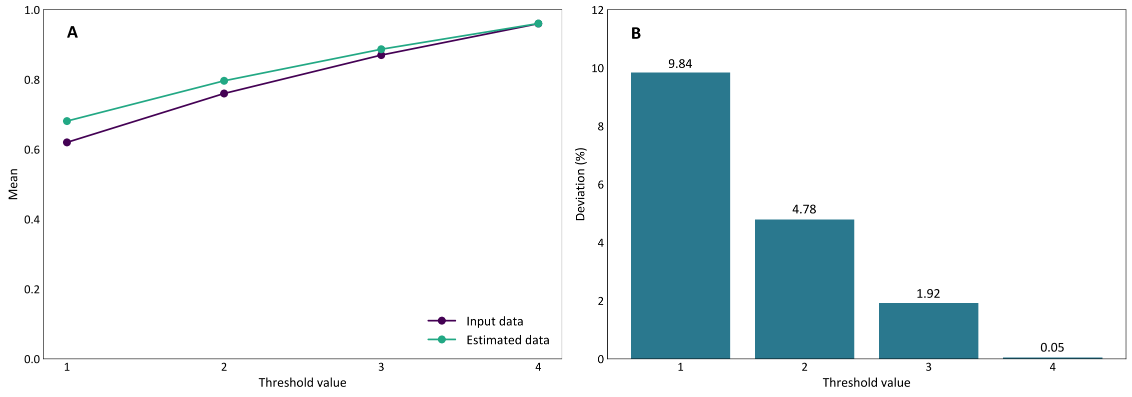

Figure 12: A) Average of the indicator variables in the input data and in the IK-based model (probabilities below thresholds). B) Percentage differences between the averages of the probabilities in the kriging-based model and the input data

In the Code 16, initially, it was defined the plot's size (line 1). In the sequence, using the *subplot* function (line 2), the first plot was positioned in the specified grid (1 row and 2 columns). Then, it was plotted the average of the probabilities below thresholds in the input data against the threshold values (line 3) and the average of the probabilities in the estimated results against each threshold value (line 4); this first plot was formatted using the functions between the lines 5 and 10. Also, using the *subplot* function, the second plot was positioned in the grid (line 12) and was created using the *bar* function (line 13) and subsequently formatted (lines 14 to 22). Lastly, it was applied the *tight layout* option (line 24) and the final plot was saved using the *savefig* function (line 25).

Comparing the average of the probabilities in the kriging-based model with those of the input data (Figure 12A and Figure 12B), one can ensure that the mean values were approximately reproduced in the model, whose highest deviation was 9.84%, attributed to the threshold equal to 1. Such deviation can be a consequence of the higher spatial variability intrinsic to the indicator variable associated with this threshold than the other ones, observed by the presence of the spherical model in the semivariogram fitting of this variable (IND0). In contrast, in the others, the Gaussian one was used, which indicates a phenomenon with a much higher continuity (Table 10).

Also, it's noted in Figure 12B the occurrence of the smallest deviation (0.05%) in the threshold equal to 4. Despite the low reliability of the experimental semivariograms for this threshold, as stated previously, the occurrence of a smaller deviation for this one, in relation to the others, can be a consequence of the high spatial continuity in this scenario (fitting by Gaussian models), with probabilities higher than zero strictly limited to the plate's center, where the damage is located. Thus, despite instabilities in the semivariograms of indicator variables related to very high thresholds, the results were not inconsistent, allowing the prediction of the structural damage's occurrence.

5 Concluding Remarks

This chapter briefly discussed concepts related to ISHM and kriging methods used to locate structural damages. Using a case study built step by step, the elementary phases were detailed using Python language packages to find structural damage in the center of an aluminum plate. The results showed that this new approach made it possible to estimate the location of the damage, resulting in maps of easy visual identification.

Additional studies are needed to better understand the technique's applicability in structural health monitoring systems, especially regarding the dimensions of the piezoelectric element mesh (spacing and number of elements in the mesh) for the identification of the damage occurrence site, through the proposed approach. In this aspect, other geostatistical tools, such as conditional simulation, could be explored for application in structural integrity monitoring systems regarding sensor mesh determination.

References

M. Abzalov. *Applied Mining Geology*. Springer Cham, 1 edition, 2016.

M. Armstrong. *Basic Linear Geostatistics*. Springer Berlin, Heidelberg, 1 edition, 1998.

F. G. Baptista and J. Vieira Filho. Optimal frequency range selection for pzt transducers in impedance-based shm systems. *IEEE Sensors Journal*, 10(8):1297–1303, 2010.

S. Bhalla and C.-K. Soh. Electro-mechanical impedance technique. In C.-K. Soh, Y. Yang, and S. Bhalla, editors, *Smart Materials in Structural Health Monitoring, Control and Biomechanics*, pages 17–51. Springer-Verlag, 2012.

O. Cherrier, P. Selva, V. Pommier-Budinger, F. Lachaud, and J. Morlier. Damage localization map using electromechanical impedance spectrums and inverse distance weighting interpolation: Experimental validation on thin composite structures. *Struct. Health Monit.*, 12(4):311–324, July 2013.

J.-P. Chilès and P. Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. John Wiley & Sons, 2 edition, 2012.

J. Deutsch, M. Deutsch, R. Martin, W. Black, T. Acorn, R. Barnett, and M. Hadavand. pygeostat python package, 2020.

V. Giurgiutiu. *Structural Health Monitoring with Piezoelectric Wafer Active Sensors*. Academic Press, 2 edition, 2014.

V. Giurgiutiu and C. A. Rogers. Recent advancements in the electromechanical (E/M) impedance method for structural health monitoring and NDE. In M. E. Regelbrugge, editor, *Smart Structures and Materials 1998: Smart Structures and Integrated Systems*. SPIE, July 1998.

V. Giurgiutiu, A. P. Reynolds, and C. A. Rogers. Experimental investigation of e/m impedance health monitoring for spot-welded structural joints. *Journal of Intelligent Material Systems and Structures*, 10(10):802–812, 1999.

D. R. Gonçalves, J. d. R. V. d. Moura Junior, and P. E. C. Pereira. MONITORAMENTO DE INTEGRIDADE ESTRUTURAL BASEADO EM IMPEDÂNCIA ELETROMECÂNICA UTILIZANDO O MÉTODO DE KRIGAGEM ORDINÁRIA. *HOLOS*, 36(2):1–16, Apr. 2020.

D. R. Gonçalves, J. d. R. V. d. Moura Junior, P. E. C. Pereira, M. V. A. Mendes, and H. S. Diniz-Pinto. Indicator kriging for damage position prediction by the use of electromechanical impedance-based structural health monitoring. *Comptes Rendus Mécanique*, 349(2):225–240, Apr. 2021.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

W. Hustrulid, M. Kuchta, and R. Martin. *Open Pit Mine Planning and Design*. CRC Press, 3 edition, 2013.

E. H. Isaaks and R. M. Srivastava. *An Introduction to Applied Geostatistics*. Oxford University Press, 1 edition, 1989.

A. G. Journel and C. J. Huijbregts. *Mining Geostatistics*. Academic Press, 1 edition, 1978.

P. Kitanidis. *Introduction to Geostatistics: Applications in Hydrogeology*. Cambridge University Press, 1 edition, 1997.

C. Kravolec, M. Schagerl, and M. Mayr. Localization of damages by model-based evaluation of electro-mechanical impedance measurements. NDT.net, July 2018.

M. Labots. os-sys python package, 2019.

C. Liang, F. P. Sun, and C. A. Rogers. Coupled electro-mechanical analysis of adaptive material systems â determination of the actuator power consumption and system energy transfer. *Journal of Intelligent Material Systems and Structures*, 5(1):12–20, 1994.

B. Lin and V. Giurgiutiu. Modeling and testing of pzt and pvdf piezoelectric wafer active sensors. *Smart Materials and Structures*, 15(4):1085–1093, 2006.

C. R. Marqui, D. D. Bueno, F. G. Baptista, R. B. Santos, J. Vieira Filho, and V. Lopes Júnior. External disturbance effect in damage detection using electrical impedance. In *Conference and exposition on structural dynamics*, volume 3, pages 678–687. Society for Experimental Mechanics, 2008.

I. I. C. Maruo, G. d. F. Giachero, V. Steffen Junior, and R. M. Finzi Neto. Electromechanical impedance - based structural health monitoring instrumentation system applied to aircraft structures and employing a multiplexed sensor array. *Journal of Aerospace Technology and Management*, 7(3):294–306, 2015.

W. McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

G. Park, H. Sohn, C. R. Farrar, and D. J. Inman. Overview of piezoelectric impedance-based health monitoring and path forward. *The Shock and Vibration Digest*, 35(6):451–463, 2003.

D. M. Peairs, D. J. Inman, and G. Park. Circuit analysis of impedance-based health monitoring of beams using spectral elements. *Structural Health Monitoring*, 6(1):81–94, 2007a.

D. M. Peairs, P. A. Tarazaga, and D. J. Inman. Frequency range selection for impedance-based structural health monitoring. *Journal of Vibration and Acoustics*, 129(6):701–709, 2007b.

M. J. Pyrcz, H. Jo, A. Kupenko, W. Liu, A. E. Gigliotti, T. Salomaki, and J. Santos. Geostatspy python package, 2021.

V. Raju. Implementing impedance-based health monitoring. Master's thesis, Virginia Polytechnic Institute and State University, 1997.

M. B. Revuelta. *Mineral Resources: From Exploration to Sustainability Assessment*. Springer Cham, 1 edition, 2018.

M. E. Rossi and C. V. Deutsch. *Mineral Resource Estimation*. Springer Dordrecht, 1 edition, 2014.

A. J. Sinclair and G. H. Blackwell. *Applied Mineral Inventory Estimation*. Cambridge University Press, 1 edition, 2002.

F. P. Sun, Z. A. Chaudhry, C. A. Rogers, M. Majmundar, and C. Liang. Automated real-time structure health monitoring via signature pattern recognition. In I. Chopra, editor, *Smart Structures and Materials 1995: Smart Structures and Integrated Systems*, volume 2443, pages 236–247. SPIE, 1995.

J. K. Yamamoto and P. M. B. Landim. *Geoestatística: Conceitos e Aplicações*. Oficina de Textos, 1 edition, 2013.

A. N. Zagrai and V. Giurgiutiu. Electromechanical impedance modeling. In C. Boller, F.-K. Chang, and Y. Fujino, editors, *Encyclopedia of Structural Health Monitoring*, pages 71–89. John Wiley & Sons, 2009.

A Module geostatspy.GSLIB

Table A1: Some functions of the geostatspy.GSLIB

Funtion	Application
ndarray2GSLIB	Utility to convert a NumPy's 1-dimensional or 2-dimensional array to a GSLIB's Geo-EAS file, which can be used in GSLIB methods
GSLIB2ndarray	Utility to convert a GSLIB's Geo-EAS file to a NumPy's 1-dimensional or 2-dimensional array for using in Python methods
Dataframe2GSLIB	Utility to convert a Pandas' Dataframe to a GSLIB's Geo-EAS file for using in GSLIB methods
GSLIB2Dataframe	Utility to convert a GSLIB's Geo-EAS file to a Pandas' DataFrame one for use in Python methods
DataFrame2ndarray	Utility to convert a Pandas' DataFrame to a NumPy's 2-D array
affine	Affine distribution transformation to correct feature mean and standard deviation
nscore	Transform an original distribution to a normal-shape one
declus	Cell-based declustering
make_variogram	Make a dictionary of semivariogram parameters for use in estimation and/or simulation methods
gamv	Calculation of semivariograms in an irregularly spaced 2-D spatial data
varmap	Map of semivariograms for regularly spaced 2-D data
varmapv	Map of semivariograms for irregularly spaced 2-D data
vmodel	Semivariogram fitting
kb2d	Simple and/or ordinary kriging for 2-D data
sgsim_uncond	Non-conditional sequential Gaussian simulation for 2-D data
sgsim	Sequential Gaussian simulation for 2-D or 3-D data
cosgsim_uncond	Non-conditional sequential Gaussian Co-simulation for 2-D data
sample	Utility to create samples from a 2-D model with provided X and Y, and append to a DataFrame
gkern	make a Gaussian kernel for convolution, moving window averaging
regular_sample	Utility to extract samples regularly spaced from a 2-D spatial model
random_sample	Utility to extract samples randomly spaced from a 2-D spatial model
pixelplt	Plotting of 2-D estimated grids. Reimplementation in Python of the GSLIB's pixelplt with Matplotlib methods
pixelplt_st	Plotting of 2-D estimated grids. Reimplementation in Python of the GSLIB's pixelplt with Matplotlib methods, with support for sub plots.
pixelplt_log_st	Plotting of 2-D estimated grids. Reimplementation in Python of the GSLIB's pixelplt with Matplotlib methods, with support for sub plots and log color bar
locpix	Plotting of 2-D estimated grids and sample points. Reimplementation in Python of a GSLIB's MOD with Matplotlib methods
locpix_st	Plotting of 2-D estimated grids and sample points. Reimplementation in Python of a GSLIB's MOD with Matplotlib methods, with support for sub plots
locpix_log_st	Plotting of 2-D estimated grids and sample points. Reimplementation in Python of a GSLIB's MOD with Matplotlib methods, with support for sub plots and log color bar
locmap	Plotting of 2-D sample points. Reimplementation in Python of a GSLIB's MOD with Matplotlib methods
locmap_st	Plotting of 2-D sample points. Reimplementation in Python of a GSLIB's MOD with Matplotlib methods, with support for sub plots
hist	Utility for histogram plotting. Reimplementation in Python of the GSLIB's hist with Matplotlib methods
hist_st	Utility for histogram plotting. Reimplementation in Python of the GSLIB's hist with Matplotlib methods, with support for sub plots