Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE GRAU

Model per a impressió 3D de superfícies basades en Batimetria

Treball realitzat per:

Pol Baños Castelló

Dirigit per:

Alberto García González

Jaume Soler Villanueva

Grau en:

Ciències i Tecnologies del Mar

Barcelona, **29/06/2022**

Departament d'Enginyeria Civil i Ambiental (DECA) - Matemàtica Aplicada i Estadística (MAE)

Agraïments

Aquest treball no hauria estat possible sense el suport incondicional de la meva amiga Carme Feliu i Cremades a la que vull agrair la seva dedicació i comprensió a qualsevol hora del dia o la nit. Gràcies, Carme.

A la meva família per aguantar-me, sé que han passat moments difícils al meu costat aquest últim temps, la vida et posa a prova sempre. L'elaboració d'aquest projecte acompanyat d'una situació difícil viscuda per un amic m'ha portat a viure moments molt crítics i depriments des de fa uns mesos, i sempre els he tingut al costat. Gràcies, família.

A tots i totes les companyes de classe i de la colla Cactus que m'han acompanyat durant aquest llarg viatge. Gràcies, amics i amigues

Al Raúl Gímenez Rodrigo dels serveis TIC UTG Camins - TECH que no m'ha deixat de la ma en cap moment, oferint-me el seu suport incondicional per poder obtenir les impressions 3D. Gràcies, Raúl.

Als meus dos tutors Alberto García González i Jaume Soler Villanueva que van acceptar la meva proposta de projecte i han estat seguint-lo de manera activa i generosa des de l'inici. Gràcies, Alberto i Jaume.

GRÀCIES SENSE VOSALTRES RES HAGUÉS ESTAT IGUAL.

ÍNDEX

RESUM

En la durada de tot el projecte es parlarà del funcionament tècnic de mètodes d'interpolació en l'àmbit de topografia i batimetria a partir dades no estructurades per generar una malla de nous punts, la qual finalment la convertirem a un fitxer STL apte per una impressora 3D. Tot el procés estarà seguint un WorkFlow basat en programari lliure que qualsevol usuari podrà utilitzar.

Els mètodes seleccionats han estat l'Inverse Distance Weighting Interpolation (IDW), el Radial Basis Function Interpolation (RBF) i Kriging Ordinari. Es faran servir els tres mètodes d'interpolació per cinc conjunts de dades de diferents zones geogràfiques: la costa de Vilanova i la Geltrú (Barcelona), el Cap de Creus (Girona), una part de la costa de Gijón (Astúries) i finalment l'illa de La Palma (Canàries). L'algoritme que s'ha emprat per realitzar les interpolacions ha estat escrit amb Python, que al costat dels softwares MeshMixer i BCN3S Stratos, són els softwares lliures que s'han fet servir.

Un cop feta la interpolació encara amb el Python, es generarà el primer STL de la superfície interpolada, a la qual se li arreglaran tots els desperfectes. Seguidament se li donarà gruix, per tenir-lo ja llest per imprimir-lo amb una impressora 3D de l'empresa BCN3D.

Finalment, un cop tenim la nostra zona interpolada i impresa en tres dimensions, s'utilitzarà un SandBox per fer la visualització final del projecte.

Paraules Clau: Interpolació, Batimetria, Inverse Distance Weighting, Radial Basis Function, Kriging Ordinari, STL, impressora 3D.

ABSTRACT

During the whole project we will explain the technical operation of interpolation methods in the context of topography and bathymetry from unstructured data to generate a mesh of new points, which we will finally convert it into an STL file ready for a 3D printer. The whole process will be following a WorkFlow based on free software that any user will be able to use.

The selected methods have been the Inverse Distance Weighting Interpolation (IDW), the Radial Basis Function Interpolation (RBF) and Ordinary Kriging. The three interpolation methods will be used for five data sets from different geographical areas: the coast of Vilanova i la Geltrú (Barcelona), Cap de Creus (Girona), a part of the coast of Gijón (Asturias) and finally the island of La Palma (Canary Islands). The algorithm used to perform the interpolations has been written with Python, which together with the software MeshMixer and BCN3S Stratos, are the free software used.

Once the interpolation is still done with Python, the first STL of the interpolated surface will be generated, to which all the defects will be repaired. Then we will give it thickness, to have it ready to print it with a 3D printer of the company BCN3D.

Finally, once we have our area interpolated and printed in three dimensions, a SandBox will be used for the final visualization of the project.

Key words: Interpolation, Bathymetry, Inverse Distance Weighting, Radial Basis Function, Ordinary Kriging, STL, 3D printer.

1 INTRODUCCIÓ – OBJECTIUS

Una bona interpolació de dades no estructurades, és a dir, que tenen una distribució irregular i que no estan en els vèrtex d'una quadricula cartesiana, com les que es treballaran en aquest projecte, pot tenir moltes utilitats en tots els sectors de simulacions i prediccions climàtiques, on les dades de batimetria són un dels inputs més importants. El motiu principal de tenir dades no estructurades és que hi ha zones de molt difícil accés, on no es poden recollir dades, és per això que realitzem aquestes interpolacions. Per exemple, en el sector de simulacions de models numèrics, programes com el SWAN i el XBEACH utilitzen una malla batimètrica com a un dels inputs del model, que simularà i farà prediccions d'onatge i moviment del sediment.

L'objectiu principal d'aquest projecte és dur a terme un WorkFlow de manera que, només llegint i seguint els passos indicats, l'usuari sigui capaç d'elaborar i passar del seu núvol de punts inicial a un STL a punt per a imprimir-lo en 3D, mitjançant l'ús de diferents entorns, com el Python, el MeshMixer i el software d'una impressora 3D.

S'han seleccionat tres mètodes d'interpolació diferents, l'Inverse Distance Weighting Interpolation (IDW), el Radial Basis Function Interpolation (RBF) i el Kriging Ordinari. Es realitzaran les interpolacions de diferents zones geogràfiques: la costa de Vilanova i la Geltrú (Barcelona), el Cap de Creus (Girona), una part de la costa de Gijón (Astúries) i finalment l'illa de La Palma (Canàries). Un cop realitzada aquesta interpolació feta amb Python seguirem amb aquest entorn per generar la primera fase: generar un fitxer STL 3D de la superfície interpolada del STL. Aquest fitxer serà convertit en un fitxer STL apte per a la impressió 3D amb el software de MeshMixer.

2 ESTAT DE L'ART

Avui en dia existeixen molt tipus d'interpolacions en l'àmbit de topografia i batimetria. Tenim des de mètodes menys complexos a més complexos. Alguns del mètodes més coneguts són (Serreta Oliván & Playán Jubillar, 1993); l'Inverse Distance Weighting Interpolation (IDW) , el Radial Basis Function Interpolation (RBF), Polígons de Thisen, la Xarxa de Triangles Irregulars (TIN), les Sèries de Fourier i Kriging entre d'altres. Donat el temps que es té per a realitzar un projecte com aquest s'ha optat per seleccionar els tres mètodes següents; el IDW, RBF i Kriging Ordinari, ja que el primer és el més simple però amb gran eficiència, el segon és una mica més complex d'utilitzar, ja que computacionalment és més costos, i finalment l'últim que és un dels millors de tots els mètodes per a la interpolació en batimetria i topografia (Adhikary et al., 2016).

S'ha decidit que la representació gràfica de la interpolació fos per mitjà d'una impressora 3D. La impressió 3D és actualment una tècnica molt innovadora que permet fer coses extraordinàries, des de la manufacturació d'una casa de forma econòmica (Plantamura F & Oberti I, 2015; Tobi et al., 2018) fins a la generació d'òrgans, com un pàncreas o un fetge, utilitzant com a material cèl·lules (Schubert et al., 2014; Yoo, 2015). Cal dir, però, que no es l'única opció per a realitzar una impressió com aquesta, també existeix la tècnica de CNC 3D, que consisteix a eliminar material d'un bloc inicial però, a part que és una tècnica més cara, és molt més difícil accedir a una d'aquestes màquines que a una impressora 3D.

Per a finalitzar es treballarà amb un SandBox per fer la presentació dels models impresos en 3D. El SandBox que tenim a la universitat és el resultat final d'un Treball Final de Màster (TFM) realitzat fa uns anys aquí a l'escola de Camins de l'UPC. El codi original del SandBox i tot el seu procediment de construcció va ser cedit gratuïtament pel seu desenvolupador, creador i estudiant en aquell moment de la Univesity of California, Davis Oliver Kreylos (Reed et al., 2016).

3 MÈTODES

3.1 MÈTODES D'INTERPOLACIÓ

Les metodologies mostrades a continuació són mètodes d'interpolació que parteixen d'un núvol de punts amb coordenades (X, Y, Z) per tal de, amb diferents mètodes d'interpolació, podrem calcular la coordenada Z punt (X, Y) en una malla diferent.

3.1.1 KRIGING ORDINARI

Kriging és un model d'interpolació molt potent en la geoestadistica i s'utilitza en molts softwares GIS (SIG - Sistemes d'Informació Geogràfica, o en anglès GIS - Geographical Information System). Atès que es tracta d'un mètode molt potent computacionalment parlant, és recomanat aplicar-lo quan tenim una alta qualitat en les dades i volem un resultat molt acurat (*Introduction to Spatial Analysis*, n.d.). Existeixen tres mètodes diferents dins del model de Kriging; el Kriging Ordinari, el Kriging Simple, i el Kriging Universal.

El mètode Kriging és similar al Inverse Distance Weighting interpolation (IDW), ja que ambdós mètodes d'interpolació expressen la suma ponderada de les dades (ArcGIS, n.d.).

Aquests mètodes utilitzen la SemiVariança per obtenir els valors de la coordenada Z en els nous punts d'interès. El problema és estimar l'altura $\hat{Z}_0$ d'un punt (X_0, Y_0) a partir dels valors observats Z_i de cada punt (X_i, Y_i),

$$1 \leq i \leq N \qquad N \text{ és el nombre de dades}$$

Això es calcula

$$\hat{Z}_0 = \sum_{i=1}^{N} w_i Z_i \tag{1}$$

on els w_i són els pesos tal que

$$\sum_{i}^{N} w_i = 1 \tag{2}$$

L'estimació Kriging s'obté agafant els w_i de manera que la variància estimada entre el punt interpolat $\hat{Z}_0$ i el punt real Z_0,

$$\sigma_E^2 = \frac{1}{N} \sum \left(Z_0 - \hat{Z}_0\right)^2 \tag{3}$$

sigui mínima.

A partir de les dades, el primer que es fa per a calcular el SemiVariograma és calcular la relació de la distància entre parells de punts i la diferència absoluta de les seves coordenades (x, y), tot calculat en metres. Obtenint així un gràfic com el de la Fig. 1 on cada punt blau fa referència a una parella de punts.

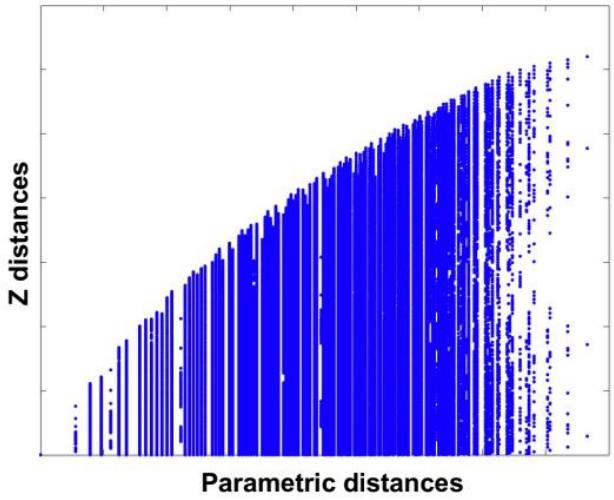


Fig. 1 Gràfica que relaciona la distància entre parells de punts i la seva diferència absoluta en la coordenada Z.

Seguidament, i a partir del que s'ha calculat en el gràfic anterior, es comença a calcular el SemiVariograma en funció SemiVariança (Eq. 4). Es divideix l'eix X de la Fig. 1 en subintervals de longitud h divisions i es calcula la SemiVariança per cada interval d'amplada h. La distància h que s'acaba d'esmentar és el que més endavant anomenarem lag, que serà la distància entre punts del SemiVariograma. En termes pràctics normalment el que es fa és dividir el SemiVariograma en un nombre determinat de $lags$ i després calcular la distància h de separació.

$$\gamma(h) = \frac{1}{2\,N(h)} \sum_{i=1}^{N(h)} \big(Z(i+h) - Z(i)\big)^2 \qquad (4)$$

$N(h) \rightarrow Nombre\ de\ punts\ aparellats\ separats\ a\ una\ distància\ \leq h$
$Z(i) \rightarrow El\ valor\ de\ Z\ d'un\ punt$
$Z(i+h) \rightarrow El\ valor\ de\ Z\ d'una\ altre\ punts\ a\ distancia\ \leq h$

A partir del càlcul de la SemiVariança obtenim la gràfica del SemiVariograma Empíric $\big(\gamma(h)\big)$ com podem veure a la Fig. 2.

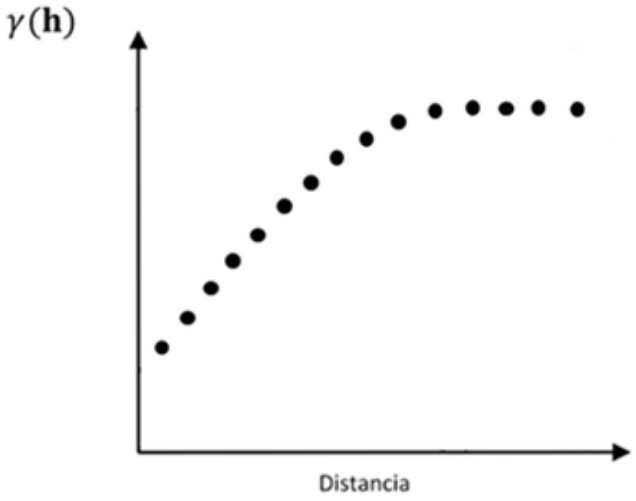


Fig. 2 SemiVariograma Empíric (Córdoba et al., 2019).

El que podem determinar a partir d'aquest SemiVariograma (alguns autors en diuen autocorrelació espacial quantificada), és que és més probable que els parells de punts que es trobin més a prop seran més semblants entre ells que no pas els que es troben més llunyans (ArcGIS, n.d.; Córdoba et al., 2019).

Els paràmetres que defineixen un SemiVariograma Teòric $\left(\hat{\gamma}(h)\right)$ són: el Nugget (C_0) és el valor on el SemiVariograma està més a prop (o quasi talla) de l'eix Y, el Sill Parcial (C), el Sill $(C + C_0)$ que és el valor del model és constant i el Rang (R) és la distància que té el model fins que el valor del SemiVariograma és constant, com es poden veure en la Fig. 3. El terme Sill fa referència al punt on la variància entre parells de punts deixa d'augmentar i roman constant amb la distància.

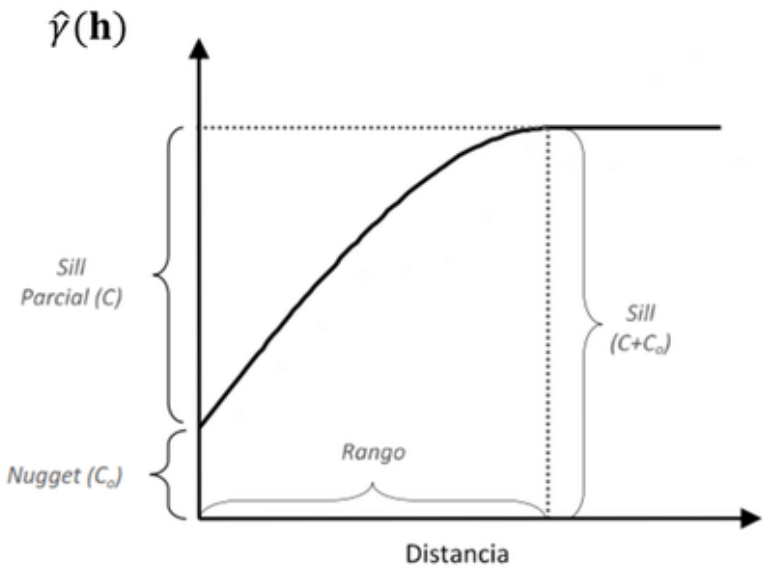


Fig. 3 SemiVariograma teòric de un model Esfèric (Córdoba et al., 2019).

En aquest punt hem de decidir quin dels models que es descriuen a continuació s'ajusta millor al nostre SemiVariograma. Aquest serà un punt d'inflexió en la resta del procés, ja que l'elecció del model determinarà la descripció espacial i les futures prediccions que es realitzaran.

Alguns dels models que s'utilitzen per ajustar Kriging Ordinari són: el Model Esfèric, el Model Exponencial, el Model Circular, el Model Gaussià i el Model Lineal. L'elecció del model serà un dels factors més importants a tenir en compte en aquest mètode, particularment en els parells de punts més propers a l'origen del SemiVariograma, ja que cada model està dissenyat per modelar diferents fenòmens (ArcGIS, n.d.).

Aquí podem veure alguns del models d'ajustament de SemiVariogrames més simples i d'altres més complexos, la majoria dels quals seran utilitzats a la part pràctica (Gabri, 2018; GISGeography, 2022; Mälicke, 2021; Montero et al., 2012).

Les següents equacions estan descrites per les mateixes variables que segueixen la nomenclatura de la Fig. 3, que corresponen a:

$C_0 \rightarrow Nugget.$
$C_1 \rightarrow Sill\ parcial.$
$h \rightarrow \text{És } la\ distància\ entre\ punt\ i\ el\ 0\ de\ l'eix\ X\ del\ Semivariograma.$
$a \rightarrow \text{És } el\ Rang\ del\ SemiVariogrma$

MODEL LINEAL

El model lineal (Fig. 4) és el més bàsic de tots, ja que la varietat espacial va augmentat linealment amb la distància. És un model que no té Sill, i és l'usuari qui els ha de posar manualment.

$$\gamma(h) = \begin{cases} C_0 + C_1\left(\frac{h}{a}\right) & si \quad 0 \leq h \leq a \\ C_0 + C_1 & si \quad h > a \end{cases} \tag{3}$$

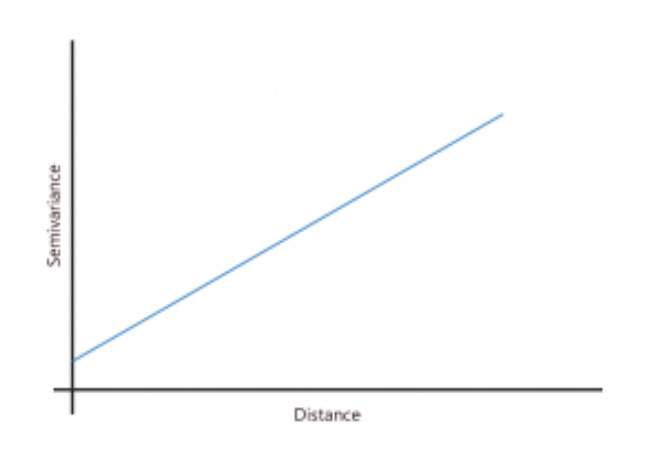


Fig. 4 SemiVariograma Teòric ajustat amb un model lineal.

MODEL ESFÈRIC

El model esfèric (Fig. 5) és un dels més utilitzats en modelat del SemiVariogrames, ja que mostra una disminució progressiva de l'autocorrelació espacial, de la mateixa forma que creix la SemiVariança però fins a cert punt on aquesta autocorrelació passa a ser zero i on tindrem un Sill.

$$\gamma(h) = \begin{cases} 0 & si \quad h = 0 \\ C_0 + C_1\left(\frac{3}{2}\left(\frac{h}{a}\right) - \frac{1}{2}\left(\frac{h}{a}\right)^3\right) & si \quad 0 < h \leq a \\ C_0 + C_1 & si \quad h > a \end{cases} \tag{5}$$

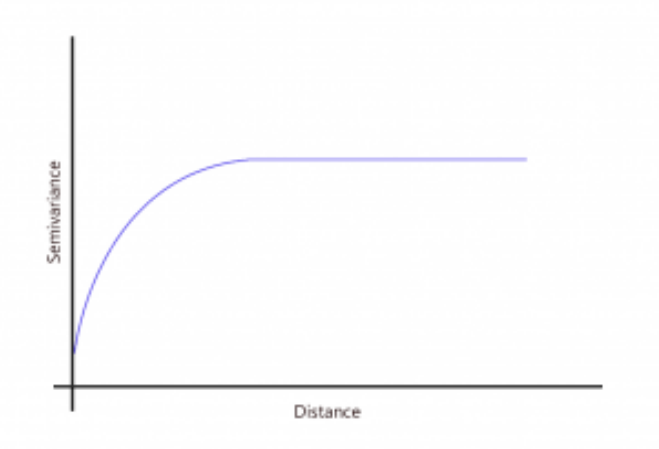


Fig. 5 SemiVariograma Teòric ajustat amb un model esfèric.

MODEL EXPONENCIAL

En el model exponencial (Fig. 6) veiem com de forma gradual la variabilitat del SemiVariograma va augmentant en funció de la distància entre parells de punts, de tal forma que a més distància tindrem una major variabilitat, sense arribar mai a un valor constant (Sill).

$$\gamma(h) = \begin{cases} 0 & si \quad h = 0 \\ C_0 + C_1\left(1 - e^{\left(-\frac{3h}{a}\right)}\right) & si \quad h > 0 \end{cases} \tag{6}$$

Fig. 6 SemiVariograma Teòric ajustat amb un model exponencial.

MODEL GAUSSIÀ

En el model gaussià (Fig. 7) veiem com variabilitat segueix una distribució de probabilitat normal o gaussiana. Això és molt útil quan a distàncies curtes tenim uns valors molt semblants, ja que en distàncies curtes tindrem una baixa variabilitat, i en distàncies altes una alta variabilitat.

$$\gamma(h) = \begin{cases} 0 & si \quad h = 0 \\ C_0 + C_1\left(1 - e^{\left(-\frac{3h^2}{a^2}\right)}\right) & si \quad h > 0 \end{cases} \tag{7}$$

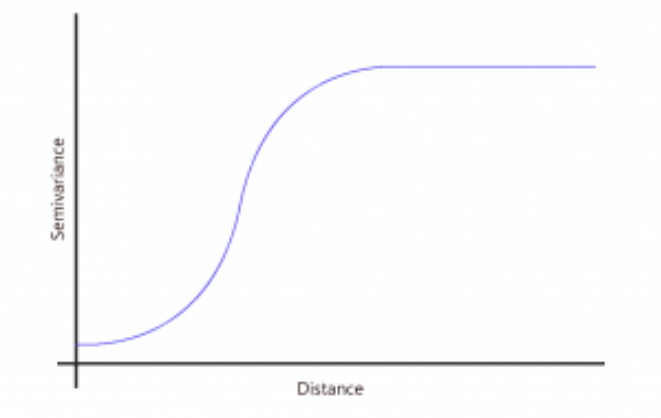


Fig. 7 SemiVariograma Teòric ajustat amb un model gaussià.

MODEL CIRCULAR

El model circular (Fig. 8) s'ajusta de forma molt semblant al model esfèric exposat anteriorment, utilitzant una funció circular per ajustar el SemiVariograma.

$$\gamma(h) = \begin{cases} 0 & si \quad h = 0 \\ C_0 + C_1\left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{h}{a}\right) + \frac{2h}{\pi a}\sqrt{1 - \left(\frac{h}{a}\right)^2}\right) & si \quad 0 < h \leq a \\ C_0 + C_1 & si \quad h > a \end{cases} \tag{8}$$

Fig. 8 SemiVariograma Teòric ajustat amb un model cúbic.

MODEL CÚBIC

El model cúbic és molt semblant al model Gaussià explicat anteriorment, també té la mateixa tendència amb un comportament parabòlic a prop del origen.

$$\gamma(h) = \begin{cases} C_0 + C_1\left(7\left(\frac{h}{a}\right)^2 - \frac{35}{4}\left(\frac{h}{a}\right)^3 + \frac{7}{2}\left(\frac{h}{a}\right)^5 - \frac{3}{4}\left(\frac{h}{a}\right)^7\right) & si \quad 0 \leq h \leq a \\ C_0 + C_1 & si \quad h > a \end{cases} \tag{9}$$

MODEL ESTABLE I MATERN

Són dos models d'ajustament que s'utilitzaran a la part pràctica degut a què, per defecte, venen implementats en la llibreria utilitzada, però no se'ls donarà importància.

Cada un dels models d'ajustament exposats s'utilitzen en diferents ocasions i en diferents àmbits de treball en funció de la naturalesa de les dades amb què es treballa. Com a norma general, un dels models que s'ajustarà millor a les nostres necessitats serà un model esfèric (Adhikary et al., 2016).

Per poder decidir quin dels següents models cal escollir podem valorar quin és l'Error Quadràtic Mitjà (RMSE), que compara el valor predit segons el SemiVariograma Teòric $(\hat{y}_i)$ i els valors del SemiVariograma Empíric (y_i), seguint la formula següent:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - \gamma_i)^2}{N}} \tag{10}$$

$\hat{y}_i \rightarrow Valors\ predits\ en\ SemiVariograma\ Teòric$.
$\gamma_i \rightarrow Valors\ observat\ en\ SemiVariograma\ Empíric$.

Un cop ja s'ha escollit quin model utilitzarem per ajustar el nostre SemiVariograma utilitzarem aquella funció per calcular els nous punts de la nostra interpolació.

3.1.2 INVERSE DISTANCE WEIGTHED INTERPOLATION

El mètode Inverse Distance Weighting interpolation (IDW) és un mètode senzill i no requereix de costos computacionals elevats, sent així un dels models deterministes més utilitzats en la interpolació espacial i amb uns costos computacionals relativament baixos (Lu & Wong, 2008). Per aquesta raó es considera aquest model d'interpolació com a un dels mètodes estàndard en l'àmbit de la ciència de la informació geogràfica (Burrough & McDonnell, 1998) i s'utilitza en molts softwares GIS, ja que no són necessaris molts coneixements en estadística geoespacial.

Tal com s'ha explicat abans, aquest mètode s'utilitza per determinar el valor $\hat{Z}(S_0)$ en el punt S_0. $\hat{Z}(S_0)$ es calcula a partir de l'Eq.11.

$$\hat{Z}(S_0) = \sum_{i=1}^{N} w_i Z(S_i) \tag{11}$$

Amb la condició de

$$\sum_{i}^{N} w_i = 1 \tag{12}$$

Com observem en l'Eq. 11 l'estimació de la nova profunditat/coordenada Z és una combinació lineal dels pesos (w_i) i els valors de profunditat/coordenada Z de les dades, on w_i segueix la fórmula següent:

$$w_i = \frac{1/d_{0i}^2}{\sum_{j=1}^{N} 1/d_{0j}^2} = \frac{d_{0i}^{-2}}{\sum_{j=1}^{N} d_{0j}^{-2}} \tag{13}$$

d : És $la\ distància\ que\ hi\ ha\ entre\ les\ coordenades\ (X_0, Y_0)\ del\ punts\ S_0\ i\ (X_i, Y_i)\ del\ punt\ S_i$.

Per al càlcul de la distància d s'ha de tenir en compte quina és l'estructura de les dades, ja que podríem tenir els punts en coordenades UTM, donades en metres, o bé en coordenades geogràfiques, donades en graus. És important saber-ho perquè per la fórmula del càlcul de distàncies en coordenades UTM podem utilitzar Pitàgores. En coordenades geogràfiques (x, y), $x = longitud, y = latitud$, la longitud d'un grau sobre un paral·lel disminueix en augmentar la

valor absolut de la latitud. La distància entre dos punts de coordenades geogràfiques (x_1, y_1) i (x_2, y_2), suposades en radians ve donada per (Whittlesey, 2020),

$$d = R * \arccos\left(\cos\left(\frac{\pi}{2} - y_0\right)\cos\left(\frac{\pi}{2} - y_i\right) + \sin\left(\frac{\pi}{2} - y_0\right)\sin\left(\frac{\pi}{2} - y_i\right)\cos(x_i - x_0)\right) \quad (14)$$

$x_0 \rightarrow Coordenada\ X\ que\ volem\ interpolar.$
$y_0 \rightarrow Coordenada\ Y\ que\ volem\ interpolar.$
$x_i \rightarrow Coordenada\ X\ d'un\ punt\ observat.$
$y_i \rightarrow Coordenada\ Y\ d'un\ punt\ observat.$
$R \rightarrow Radi\ terrestre\ en\ funció\ de\ la\ localització\ dels\ punts\ geogràfics.$

El radi de la terra dependrà de la zona on estiguem, generalment se n'utilitzen tres, el radi de la terra a l'equador, el radi en els pols i el radi mitjà de la terra.

Radi Equatorial	6378.10 Km
Radi Polar	6356.80 Km
Radi Mitjà	6371.00 Km

Taula 1. Valors en kilòmetres del radi de la Terra.

Llavors, segons les fórmules que acabem de veure, s'hauran de calcular uns pesos diferents per a cada un dels nous punts que vulguem calcular, on aquests són inversament proporcionals a la distància (d_{0i}) al quadrat que hi ha entre el nou punt (S_0) i cadascun dels punts de les nostres dades (S_i) (Lu & Wong, 2008).

Com podem extrapolar del que s'acaba d'explicar, l'elecció de quants veïns s'agafin per realitzar el càlcul dels pesos (w_i), aquests tindran un efecte directe en com es comportarà la interpolació. L'elecció del nombre de veïns també es veu determinat per la variança que tenen les nostres dades, ja que si no tenim grans variacions en zones petites agafant pocs veïns ja tindrem una interpolació acceptable, però si al contrari tenim una zona amb grans variacions en zones molt petites el més segur és que necessitem més veïns per calcular la interpolació.

3.1.3 RADIAL BASIS FUNCTION INTERPOLATION

El mètode Radial Basis Funtion (RBF) Interpolation és un mètode d'interpolació multidimensional de dades no estructurades molt potent i dels més utilitzat que hi ha (Wright, 2003). S'utilitza en sectors com el Cartogràfic (que és amb la finalitat que l'utilitzarem aquí), però fins i tot s'utilitza en xarxes neuronals destinades a Machine Learning (Karayiannis & Randolph-Gips, 2003).

La definició d'aquest mètode ve donada per les següents equacions. Aquestes signifiquen que, per un conjunt de N dades amb les seves corresponents coordenades XYZ, la nova coordenada $\hat{Z}$ del punt S_0, està definida pel sumatori d'un pes (w_i) multiplicat pel seu kernel $([K]_{0i})$.

$$\hat{Z}(S_0) = \sum_{i=1}^{N} w_i [K]_{0i} \quad (15)$$

El kernel pot tenir diferents definicions, com per exemple la Multiquadràtica, la Multiquadràtica Inversa o la Gaussiana entre d'altres.

$$k(x^i,\ x^j) = \sqrt{1 + \ (\beta \cdot \|x^i - \ x^j\|\ )^2} = [\boldsymbol{K}]_{\boldsymbol{ij}} \tag{16}$$

Radial Basis Function Kernel - > Multiquadràtic

$$k(x^i,\ x^j) = \frac{1}{1 + \ (\beta \cdot \|x^i - \ x^j\|\ )^2} = [\boldsymbol{K}]_{\boldsymbol{ij}} \tag{17}$$

Radial Basis Function Kernel - > Multiquadràtic Inversa

$$k(x^i,\ x^j) = \mathrm{e}^{-(\ \beta \cdot \|x^i - x^j\|\ )^2} = [\boldsymbol{K}]_{\boldsymbol{ij}} \tag{18}$$

Radial Basis Function Kernel - > Gaussiana

Els kernels són distribucions de densitat simètriques al voltant del punts escollit per a cada un d'ells. Les hipòtesis que hem fet ens permeten transformar les equacions en un sistema d'equacions lineal com es veu en l'Eq. 19, per trobar la solució a la nostra funció d'interpolació.

$$\begin{pmatrix} [K]_{11} & [K]_{12} & [K]_{13} & \dots & [K]_{1N} \\ [K]_{21} & [K]_{22} & [K]_{23} & \dots & [K]_{2N} \\ [K]_{31} & [K]_{32} & [K]_{33} & \dots & [K]_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ [K]_{N1} & [K]_{N2} & [K]_{N3} & \dots & [K]_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{pmatrix} \tag{19}$$

Resolent el sistema d'equacions lineal de l'Eq. 18 podem resoldre l'Eq. 14, que ens permetrà així calcular la $\hat{Z}(S_0)$ del nou punt que volem interpolar. El procés es repeteix per a cada un dels nous punts que volem calcular.

3.1.4 MÈTODES DE VALIDACIÓ

Per dur a terme una validació de com són de fiables cadascun dels mètodes d'interpolació proposats, el que fem és interpolar en coordenades (X, Y) on ja tenim una Z coneguda per després calcular quin error tenim entre el valor real i el valor interpolat. D'aquesta manera, podem seguir un mateix criteri a l'hora d'avaluar l'eficàcia de cada un dels mètodes.

Per a cada una dels punts de validació es calcula l'Error Absolut i l'Error Relatiu.

L'Error Absolut (Eq. 20) és la diferencia entre un valor real i un valor calculat.

$$Error\ Absolut = |Valor\ Real - \ \ Valor\ Interpolat| \tag{20}$$

L'Error Relatiu (Eq. 21) és la ratio entre l'Error Absolut i el Valor real.

$$Error\ Relatiu = \frac{|Valor\ Real - \ \ Valor\ Interpolat|}{|Valor\ Real|} \tag{21}$$

D'aquesta manera obtindrem la coordenada (X, Y) amb la $Z\ Real$ i $Z\ Interpolada,$ i el seu respectiu error absolut i error relatiu.

Per tal d'intentar estandarditzar el càlcul de l'error total que té el model es calcularà el Mean Squared Error (MSE) i el Root Mean Square Error (RMSE).

El MSE (Eq. 22) ens diu com de propera és la línia als nostres punts. Per fer-ho utilitza la distància entre els punts reals i els interpolats. La quadratura s'utilitza per eliminar signes negatius i donar-li més pes a aquells errors que tenen una distància més gran. Llavors, com més petit sigui el MSE, millor serà el model (Statistics How To, n.d.-a).

$$MSE = \frac{\sum_{i=1}^{N}(\hat{z}_i - z_i)^2}{N} \tag{22}$$

$\hat{z}_i \rightarrow Valors\ predits\ en\ el\ punt\ i.$
$z_i \rightarrow Valors\ observat\ en\ el\ punt\ i.$

El RMSE és l'arrel quadrada del MSE i és la desviació estàndard de l'error que hem tingut en el moment d'interpolar (Statistics How To, n.d.-b).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{z}_i - z_i)^2}{N}} \tag{23}$$

$\hat{z}_i \rightarrow Valors\ predits\ en\ el\ punt\ i.$
$z_i \rightarrow Valors\ observat\ en\ el\ punt\ i.$

En ultima instància, calculem la diferència entre el vector de coordenades $Z\ Real$ i $Z\ Interpolada$ en els punts de validació, i fem la norma d'aquesta diferència.

$$D = \|Z\ Real - Z\ Interp\| \tag{24}$$

$Z\ Real \rightarrow Vector\ amb\ les\ coordenades\ Z\ Reals\ dels\ punts\ de\ validació.$
$Z\ Interp \rightarrow Vector\ amb\ les\ coordenades\ Z\ Interpolades\ dels\ punts\ de\ validació.$

3.2 GENERACIÓ DE LA MALLA 3D

Un cop interpolats els punts d'interès el que tindrem és un arxiu .csv on la primera columna són les coordenades X, la segona columna són les coordenades Y i la darrera columna són les Coordenades Z, per tant, per cada una de les files tenim un punt (X, Y, Z).

Per a poder imprimir necessitem crear un arxiu STL (de l'anglès "STereoLithography"). És un format d'arxiu utilitzat en sistemes CAD (disseny assistit per computadora), que defineix quina és l'estructura d'un objecte en tres dimensions. El principal objectiu dels fitxers STL és codificar l'estructura d'un objecte 3D. (All3DP, 2021).

El primer pas que fem per aconseguir un arxiu STL per enviar-lo a la impressora 3D, és generar una malla de triangles en format (STL), però sense gruix, dels resultats de la interpolació. Per fer-ho s'ha fet ús de la Triangulació de Delaunay. Un cop arribats a aquest punt emprant el software de MeshMixer li donarem gruix al STL per enviar-lo a imprimir.

3.2.1 TRIANGULACIÓ DE DELAUNAY

La Triangulació de Delaunay és una xara de triangles que compleix una condició, la condició de Delaunay. Aquesta diu que la circumferència circumscrita és la que passa per tots els vèrtexs del triangle i no conté en el seu interior cap altra punt de la triangulació (Simmons, 2017). Qualsevol conjunt de punts en el pla admet una triangulació de Delaunay (Cheng et al., 2013).

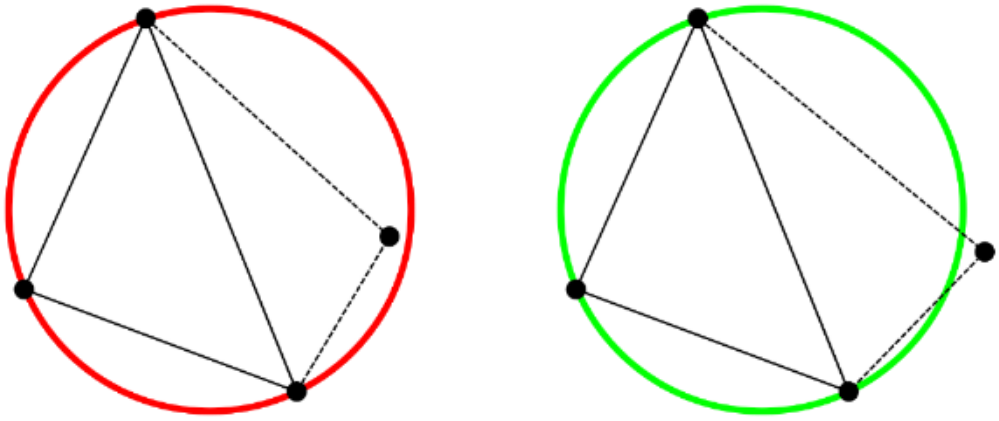


Fig. 9 Exemple de la Condició per fer la Triangulació de Delaunay. A l'esquerra observem una triangulació de Delaunay no admissible, i a la dreta n'observem una d'admissible.

Un cop generat el fitxer STL de la superfície interpolada, l'importem al software MeshMixer.

3.2.2 SOFTWARE MESHMIXER

El primer pas que s'ha de fer és importar el fitxer STL generat amb la triangulació de Delaunay al software MeshMixer, és importar aquest fitxer, per fer-ho li donem a "Import" i busquem el STL al nostre directori de treball.

Abans de prosseguir hem de tenir clares les dimensions en les quals voldrem treballar, ja que en el moment de fer la impressió 3D dependrem molt de les dimensions de la impressora.

El següent que farem és centrar l'objecte en les coordenades (0, 0, 0). Per fer-ho anirem a "Edit" – "Transform", i un cop s'obri el menú de transformació, canviarem el "Translate X", el "Translate Y" i el "Translate Z" a zero.

A continuació en el mateix menú de "Transform" ajustem les dimensions dels eixos en funció de les dimensions de la impressora. Com a norma general els eixos estan distribuïts com veiem en la Fig. 10. Hem d'escalar l'eix que abans s'apropi al límit de la impressora. Per fer-ho canviarem el valor de "Size" en l'eix que volem i mentre l'opció de "Uniform Scaling" estigui activada, el programa escalarà l'objecte en tots tres eixos per igual.

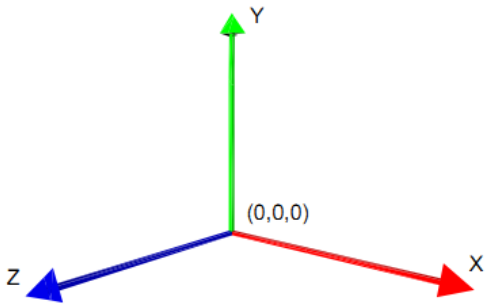


Fig. 10 Eixos en el software MeshMixer.

El pas següent el farem únicament si després de generar el STL de la superfície, la direcció de les normals, que són els vectors perpendiculars als plans generats per cada un del triangles de la malla, està en direcció cap avall, es a dir, la component de la normal és negativa, com es mostra a la Fig. 11.

Si és el cas, únicament hem d'anar a l'opció de "Select" i fer dos clics en l'objecte, això farà que el seleccionem tot. Un cop el tenim tot seleccionat anem a "Edit..." i fem clic a l'opció de "Flip normals". Si hem seguit els passos correctament, tindrem l'objecte com el veiem a la Fig. 12.

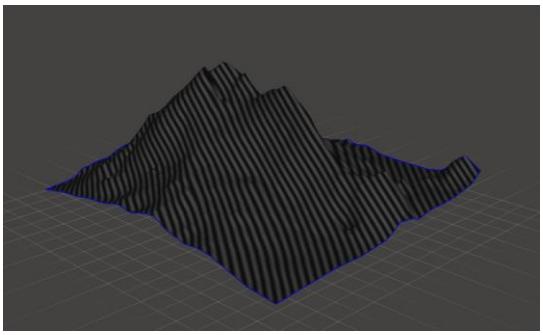


Fig. 11 Visualització del STL abans de direccionar el vector de les normals.

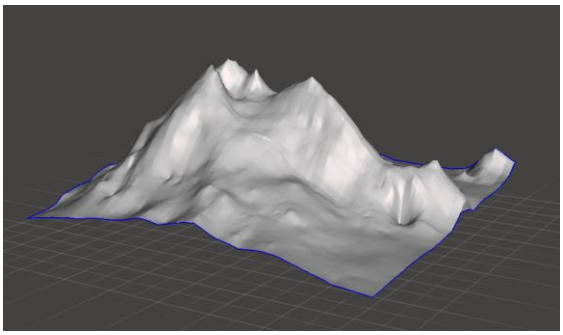


Fig. 12 Visualització del STL després de direccionar el vector de les normals.

Arribats a aquest punt farem una ullada per tal de veure si la nostre malla té defectes i si els té arreglar-los. Exemples de defectes són els que podem veure a la Fig. 13 i la Fig. 14. Ens trobem dos tipus de defectes, un d'ells és el que es generen en els extrem de la malla on tenim pendents pronunciades, que fan que es formin parets que uneixen la pendent en els extrems de la malla, l'altre es genera pel fet que tenim una zona amb pocs nodes (vèrtex del triangles) i una pendent molt pronunciada, com a conseqüència veiem que els triangles no s'han generat del tot bé.

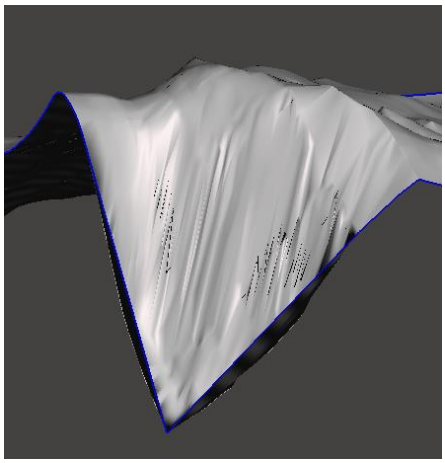


Fig. 13 Defecte probocada per tenir pocs punts en una zona amb molta pendent.

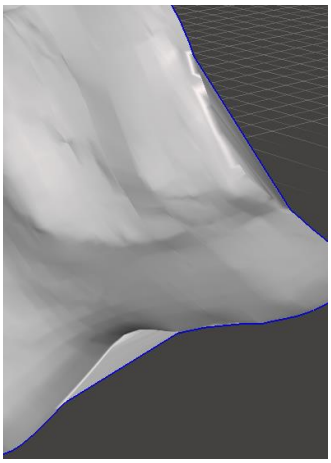


Fig. 14 Defecte en l'extrem de la malla, que forma una paret.

Per solucionar el primer defecte, el que farem és anar a "View" i activar l'opció "Show Wireframe" o prémer la tecla W per veure la malla de triangles. Després amb el "Select" amb l'opció "Size" a zero seleccionarem els triangles que volem eliminar com es mostra en la Fig. 15. Una vegada seleccionats, premem la tecla "Supr" per eliminar-los.

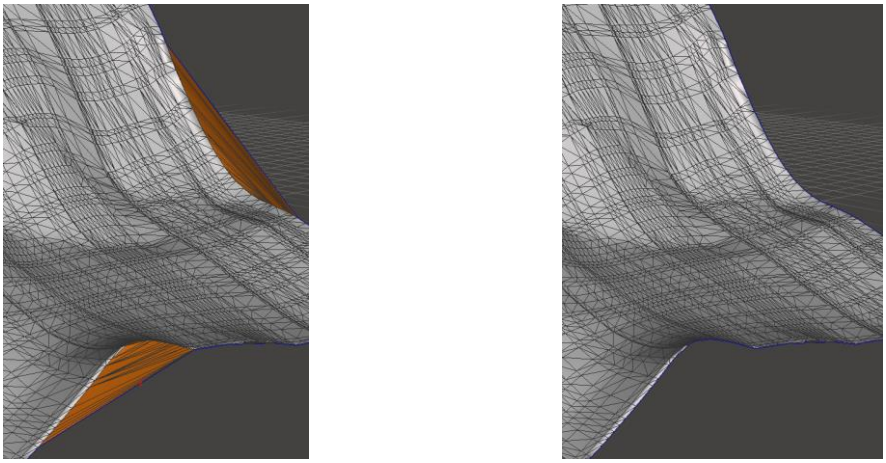


Fig. 15 Abans i despres de soluciona el defecte que presenta la malla de la Fig. 14.

Per eliminar l'últim defecte comentat, el que farem és seleccionar tot l'objecte/malla anant a l'opció de "Select" i farem dos clic a l'objecte, i aquest cop anem a "Edit..." i fem clic a l'opció de "Remesh" o prement la tecla R. I deixant els valors per defecte li donarem a acceptar. Aquest procés no només ens ajuda a reparar la nostra malla sinó que a més ens la suavitza fent-la una malla regular.

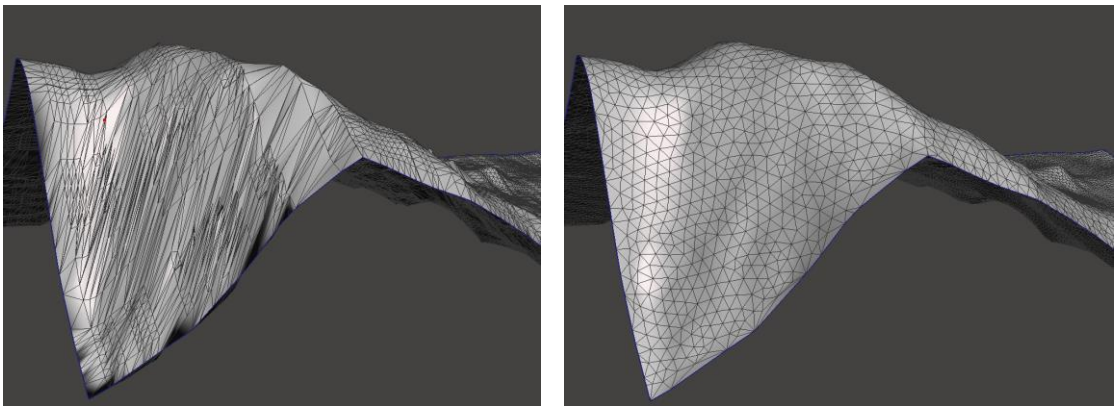
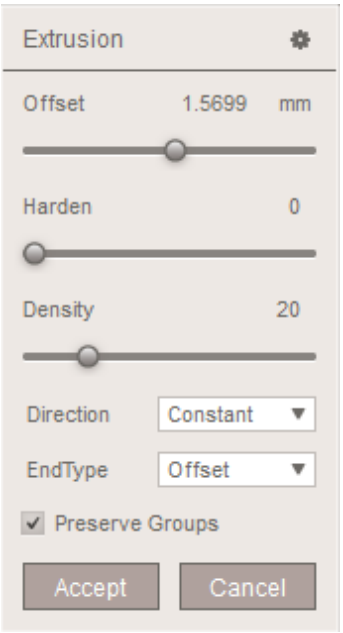


Fig. 16 Abans i despres de soluciona el defecte que presenta la malla de la Fig. 13

Aquest pas és molt important, ja que quan es genera el primer fitxer STL, la triangulació es fa en dues dimensions i després se'ls dóna els valors de profunditat provocant aquests desperfectes. És per això, que remallar és un pas molt rellevant, pel fet que estem regularitzant tota la malla però aquest cop en tres dimensions. El que significa és que tenim en compte les tres coordenades de cada punt per generar els triangles, per això queden tots amb una forma molt similar.

Finalment, haurem de donar-li gruix. Per fer-ho tornarem a anar a l'opció de "Select" i farem dos clic a l'objecte, i aquest cop anem a "Edit..." i fem clic a l'opció de "Extrude" o amb tot seleccionat, premem la tecla D i s'obrirà el següent menú.



Les opcions signifiquen:

- Offset: el gruix que li volem donar.
- Harden: com seran de suaus les cantonades del gruix, com més a prop de 0 més suaus seran les cantonades.
- Density: la densitat que tindrà l'interior del volum. Aquesta densitat serà condicionant a l'hora de la impressió, ja que a més densitat gastarem més material i trigarem més temps a imprimir.
- Direction: És la direcció que volem que segueixi el gruix. Tenim l'opció per defecte, "Constant", que donarà el gruix en direcció als vectors normal dels triangles. I després tindrem l'opció de seguir qualsevol dels eixos XYZ.
- EndType: Per defecte, tindrem l'opció de "Offset", que el que fa és donar la mateixa forma de la superfície original per generar la segona cara del gruix. Per altra banda, tenim l'opció de "Flat" que, com diu la paraula, l'altra cara serà un pla.

Per finalitzar l'"Extrude" cliquem el botó "Accept" i aconseguirem un objecte amb gruix com el que veiem a la següent Fig. 17.

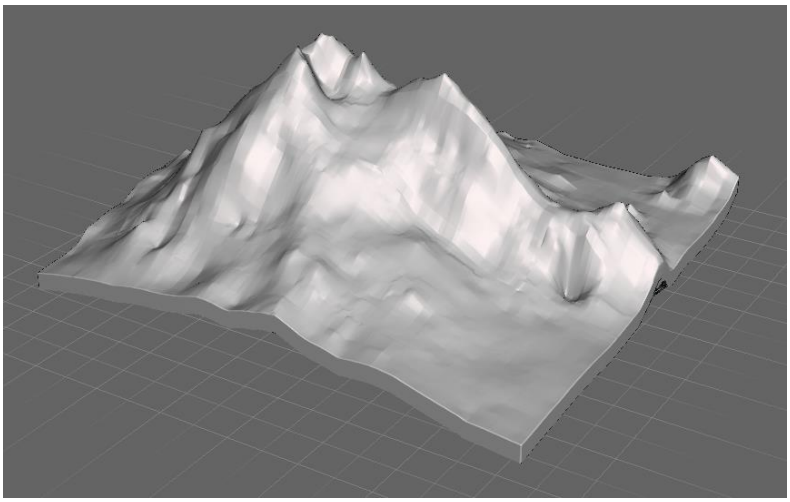


Fig. 17 Visualització de com quedarà el STL amb gruix.

Abans d'exportar el model hem de comprovar que les dimensions siguin aptes per a la nostra impressora i si no és el cas i volem mantenir les dimensions originals del nostre objecte no tenim més remei que tallar-lo. Per fer-ho, abans de tot hem de saber en quants trossos el volem tallar i les dimensions de cadascun d'ells.

Una vegada sabem com farem els talls, anem a "Edit" i fem clic a l'opció de "Plane Cut". En el menú que s'obrirà tindrem dos variables, "Cut type" i "Fill type". Com el que volem aconseguir és dividir el l'objecte en diferent blocs i que cadascun d'ells tingui un gruix propi, aquestes variables s'han desar en: "Slice (Keep Both)" i "Remeshed Fill" respectivament.

Per realitzar el tall apareixerà un pla que talla la figura com el que veiem a la Fig. 18. Aquest es pot ajustar canviant l'orientació del pla i/o desplaçant-lo. Quan tinguem el pla en el lloc on volem fer el tall li donem a acceptar i ens retornarà un objecte dividit en dos.

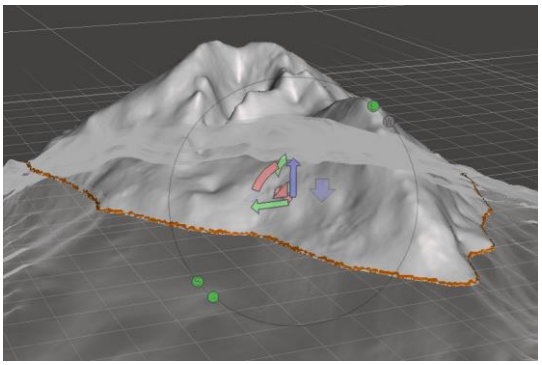


Fig. 18 Pla que talla l'objecte.

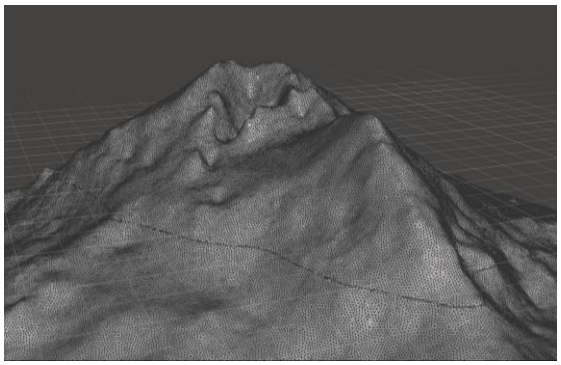


Fig. 19 Com és veu l'objecte un cop tallat.

Però encara faltaria un últim pas per a poder exportar aquests dos objectes per separat. En el mateix menú de "Edit" anem a l'opció "Separate Shells". Automàticament ens dividirà l'objecte en dos i ens obrirà el menú "Object Browser" on veurem que efectivament ja tindrem dos objectes STL diferents.

Ara ja només quedarà exportar els models perquè una impressora 3D el pugui obrir. Per fer-ho només haurem de clicar a "Export" i guardar-lo en format STL ASCII Format (*.stl).

3.3 IMPRESSORA 3D

La impressora utilitzada per fer totes les impressions és la Sigma R19 de l'empresa BCN3D. Algunes de les especificacions més importants són (BCN3D, n.d.):

Volum d'impressió	210 x 297 x 210 mm
Numero d'extrusors	2
Resolució de posició	Eix X: 0.0125 mm Eix Y: 0.0125 mm Eix Z: 0.001 mm
Temperatura de funcionament	15 ºC - 35 ºC
Temperatura d'extrusió màxima	290 ºC
Materials admissibles	PLA / ABS / Nylon / PET-G / TPU / PVA / Composites / Altres
Connectivitat	Targeta SD, USB
Consum elèctric	240W
Software preparació fitxers	BCN3D Cura = BCN3D Stratos

Taula 2. Especificacions de la impressora 3D Sigma R19 de BCN3D.

L'elecció d'aquesta impressora i no una altra és perquè és la que ens ha proporcionat els serveis TIC de la UPC. Com bé diu en les especificacions s'utilitzarà el software del BCN3D Cura/Stratos per preparar els fitxer abans d'imprimir-los.

Fig. 20 Impressora 3D, Sigma R19 de BCN3D.

Amb el software de BCN3D Cura/Stratos prepararem el fitxers STL generats en els passos anteriors.

Un cop importem el STL que volem imprimir, el primer que farem serà centrar l'objecte al centre de la base virtual de la impressora, seguidament desplegarem el menú de configuració de dalt a la dreta. Però no ens serveix la configuració d'impressió predeterminada, per canviar-la

premerem l'apartat de "Custom", ho deixarem tot en per defecte menys dos apartats, "Infill" i "Support".

El primer dels apartats que hem de canviar a criteri de l'usuari és el "Infill" que és la densitat i forma que tindrà l'interior del volum del STL. La forma que tindrà dependrà molt de la forma que tingui el STL. D'altra banda, la densitat és un punt molt important, ja que com més densitat establim a la impressió, més material gastarem i el temps d'impressió serà més llarg. L'altre apartat que hem de modificar és el "Support", el qual hem de tenir igual que a la Fig. 21, que és la configuració que s'ha utilitzat per fer les impressions.

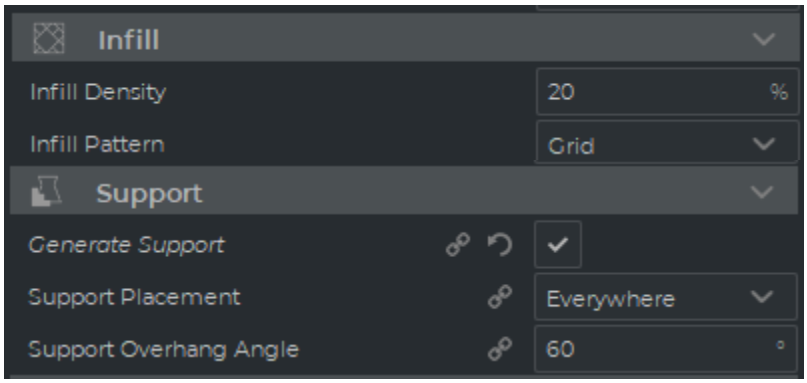


Fig. 21 Configuració utilitzada en cada una de les impressions.

Seguidament li donarem al botó "Slice" o "Segmentación" de a baix a la dreta que començarà a preparar com imprimirà el STL, el cost aproximat del material, de la impressió, i el temps que trigarà. Si li donem a "Preview" podrem veure detalladament com es farà la impressió., incloent tot els suports que necessitarem per imprimir. Ens mostrarà en color blau la impressió del STL, en color verd, tota l'estructura dels suports que necessitarem per imprimir, i en color vermell els moviments que farà l'extrusor.

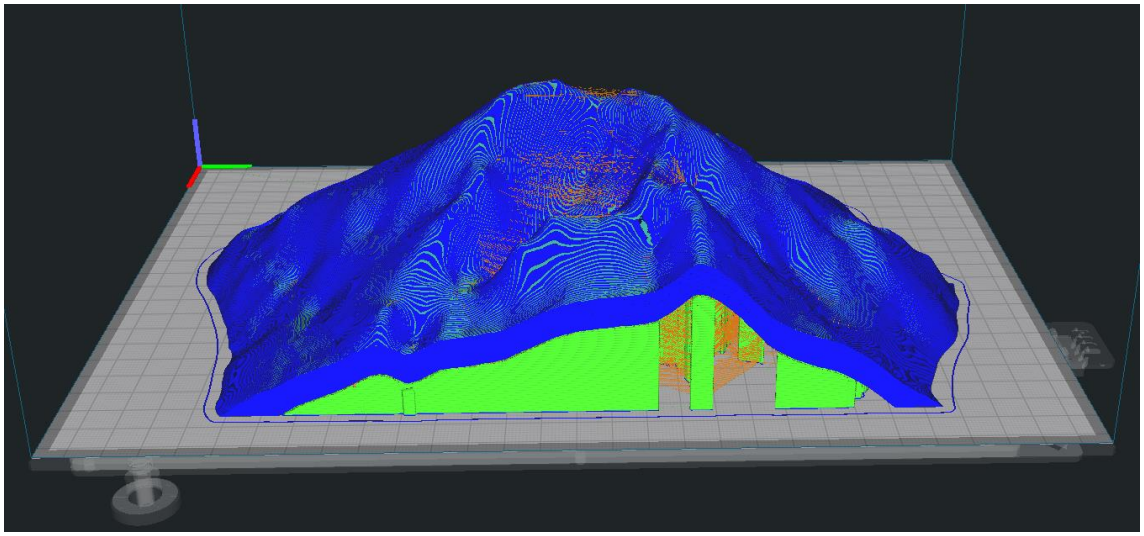
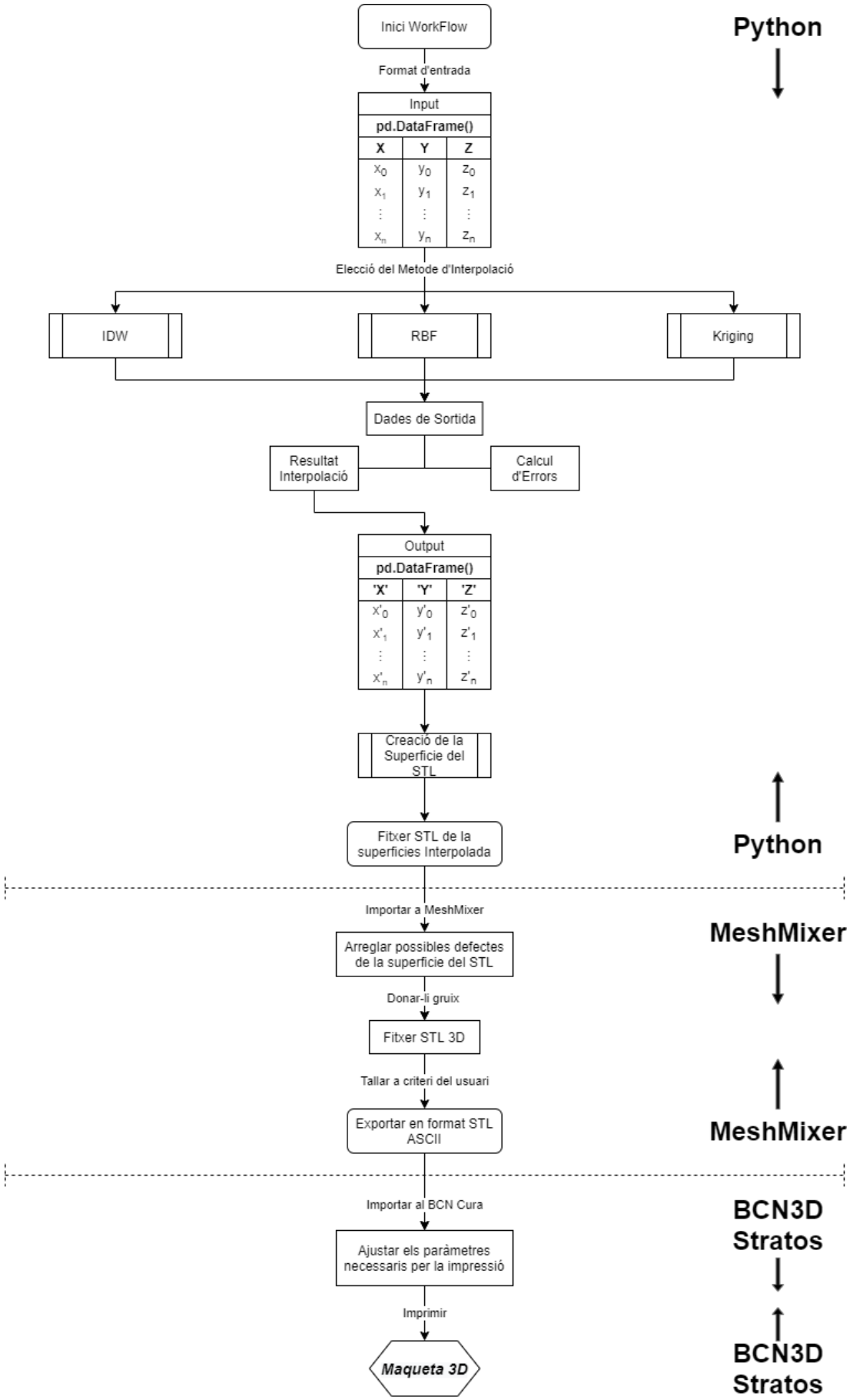


Fig. 22 Visió detallada de com és farà la impressió.

Finalment, el darrer pas és exportar-ho tot en format *.gcode que serà el que la impressora podrà llegir a l'hora de la impressió.

3.4 WORK FLOW


Inici WorkFlow
Python
Format d'entrada
Input
pd.DataFrame()
X Y Z
Elecció del Metode d'Interpolació
IDW
RBF
Kriging
Dades de Sortida
Resultat Interpolació
Calcul d'Errors
Output
pd.DataFrame()
'X' 'Y' 'Z'
Creació de la Superficie del STL
Fitxer STL de la superficies Interpolada
Python
Importar a MeshMixer
Arreglar possibles defectes de la superficie del STL
MeshMixer
Donar-li gruix
Fitxer STL 3D
Tallar a criteri del usuari
Exportar en format STL ASCII
MeshMixer
Importar al BCN Cura
Ajustar els paràmetres necessaris per la impressió
BCN3D Stratos
Imprimir
Maqueta 3D
BCN3D Stratos

4 RESULTATS

S'han estudiat les següents zones geogràfiques: la costa de Vilanova i la Geltrú (Barcelona), la topografia de Cap de Creus (Girona), una part de la costa de Gijón (Astúries) més específicament el Cerro de Santa Catalina i les dues platges que té al costat i, finalment, l'illa de La Palma (Canàries). Aquesta última zona és la que s'explicarà més detalladament en aquests apartat, la resta de resultats els podrem trobar a l'apartat d'Annexes.

Les dades inicials de la illa de La Palma són una matriu 3 x 2 305 on la primera columna correspon a la component x dels punts, la segona columna a la coordenada y dels punts i finalment la tercera columna que correspon a la coordenada z dels punts. Cada una de les files correspon a un punt amb coordenades (X, Y, Z).

Les dades estan en format de coordenades geogràfiques, el que significa que les components x i y no estan en metres sinó en graus (Latitud i Longitud). La representació visual de les nostres dades la podem observar a la Fig. 23.

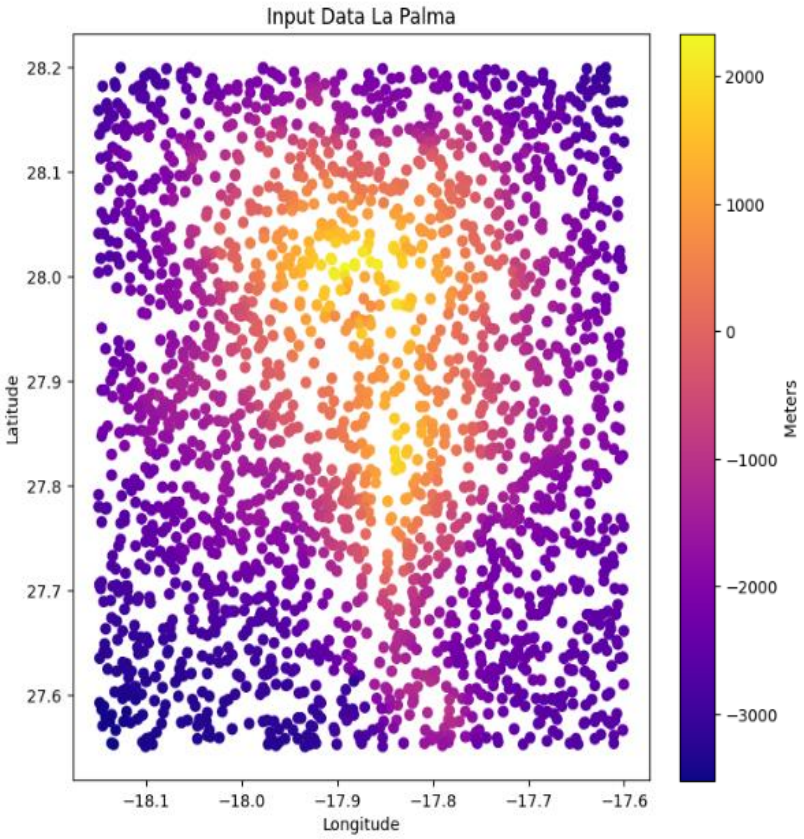


Fig. 23 Representació visual del Data Set de La Palma.

Dels 2305 punts (X, Y, Z) inicials, 210 es destinaran a fer la validació del model i els 2095 restants s'utilitzaran per fer les interpolacions.

4.1 IDW INTERPOLATION RESULTATS

Per a la primera de les interpolacions agafarem diverses distàncies al voltant de cada punt per tal de veure com afecta el nombre de veïns seleccionats per interpolar en cada uns dels punts. Els radis seleccionats són: 0.10, 0.25, 0.50, 1.00, 1.50, 2.00 (recordem que estem treballant amb graus), per a cada una de les variables de la interpolació s'han interpolat 67 600 nous punts (X, Y, Z).

Com podem veure, a la taula comparativa de l'error obtingut de les interpolacions amb diferents radis de veïns, com més gran és el radi tindrem uns errors més elevats que no pas si tenim un radi més petit. Això és pel fet que com més veïns utilitzem per calcular un nou punt tindrem més informació del terreny i conseqüentment els pesos estaran més repartits, reduint així la importància dels veïns més propers. Però per altra banda, a menys veïns els pesos estaran repartits entre menys veïns i d'aquesta forma se li donarà més importància als veïns més propers que no pas abans. Cal, però, vigilar perquè si agafem un radi massa petit tindrem el problema que depèn del punt que estiguem interpolant potser no trobem veïns dins del radi de recerca i desafortunadament tindrem un valor NaN $(No\ Data)$.

	IDW Radial Distances					
	0.01	0.025	0.05	0.1	0.15	0.2
Absolute Error	NaN	100.0567	118.6891	170.0483	221.8803	270.2978
Relative Error	NaN	3.0915	3.9345	5.8202	7.6967	9.4585
$\|\|z\ Real - \ z\ Interp\|\|$	NaN	1449.9596	1719.9679	2464.2343	3215.3514	3916.9878

Taula 3. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

Aquí podem veure tres exemples de les distàncies 0.1, 0.25 i 10, de manera que a la primera de totes veiem que agafant un radi massa petit no podrem fer una bona interpolació, ja que tindrem llocs on no tindrem dades, i uns altres dos per comprovar la diferència entre la un radi gran i un altre petit.

Per tal de comprovar que no tinguem un error en el càlcul de les distàncies, ja que quan parlem de latitud i longitud un grau no fa referència a la mateixa distància a causa del fet que la terra no és plana sinó que és el·lipsoide, s'ha repetit les interpolacions tenint en compte que 0.01 graus de latitud i longitud són aproximadament 1,1 km i 1 km respectivament, a la latitud de l'illa de La Palma. Les interpolacions s'han repetit amb els següents paràmetres; 1, 2.5, 5, 10, 15, i 20 km. Com a resultat de les interpolacions de comprovació obtenim la següent taula d'errors.

	IDW Radial Distances (Km)					
	1.00	2.50	5.00	10.0	15.0	20.0
Absolute Error	NaN	100.1564	118.8183	167.0847	218.3348	266.6001
Relative Error	NaN	3.0530	3.8561	5.6285	7.4639	9.1852
$\|\|z\ Real - \ z\ Interp\|\|$	NaN	1451.4044	1721.8408	2421.2873	3163.9719	3863.4019

Taula 4. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

A continuació podem veure el resultat de les dues interpolacions que tenen l'error més petit.

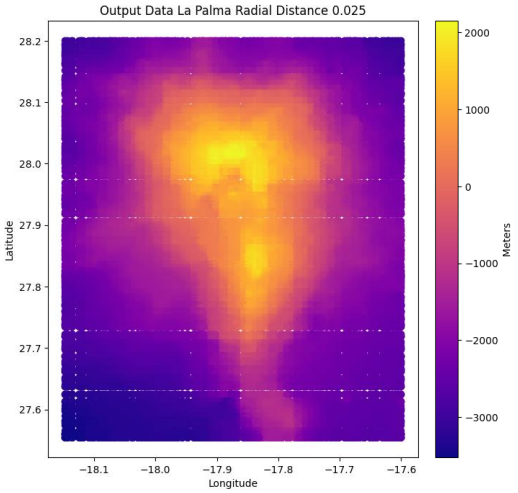


Fig. 24 Resultats de IDW amb Coordenades UTM i Radi 0.025 (GRAUS).

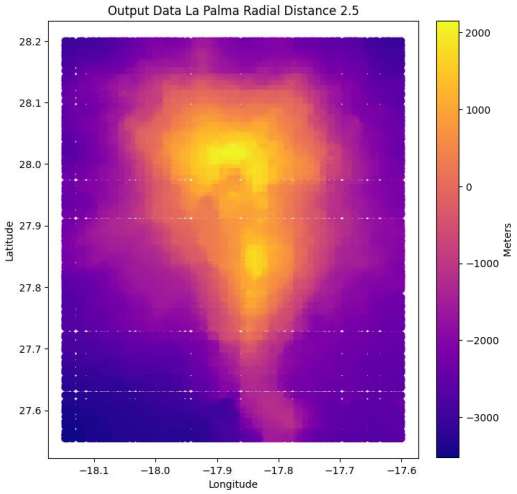


Fig. 25 Resultats de IDW amb Coordenades Geogràfiques i Radi 2.50 Km.

Podem veure la diferència absoluta entre ambdues interpolacions a la Fig. 26

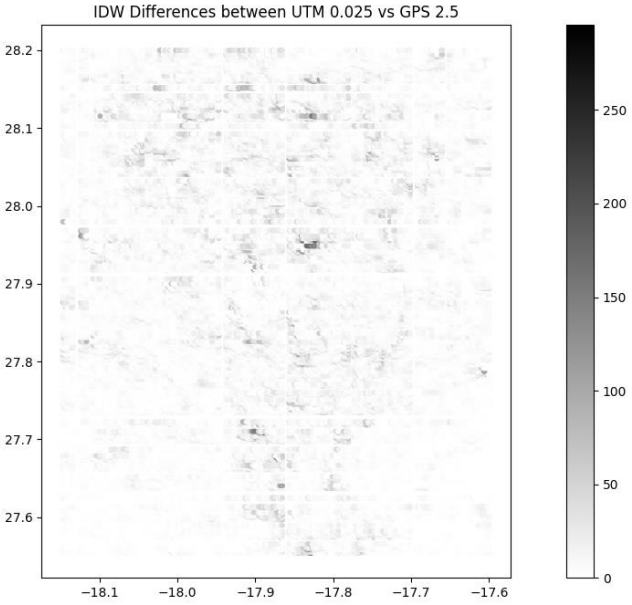


Fig. 26 Diferencia absoluta en metres entre les dos interpolacions de la Fig. 24 i la Fig. 25.

4.2 RBF INTERPOLATION RESULTATS

Per a la segona de les interpolacions agafarem diverses β per tal de veure com afecta aquest valor en l'error de la interpolació, tal i com s'ha fet en el mètode anterior amb les distàncies. Les β seleccionades són: 0.25, 0.50, 1.00, 2.50, 5.00, i 10.0 (no unitats, perquè β és adimencional), per a cada una de les variables de la interpolació s'han interpolat 67 600 nous punts XYZ.

Per a realitzar aquesta interpolació s'ha utilitzat el Kernal de tipus multiquadràtic.

Com podem veure, a la taula comparativa de l'error obtingut de les interpolacions amb diferents valors de β, com més gran sigui aquesta β tindrem uns errors més petits que no pas si agaféssim una β més petita.

	RBF β					
	2.50	5.00	10.0	25.0	50.0	75.0
Absolute Error	14836.194	22068.353	136324.31	271.4132	124.5988	95.7327
Relative Error	776.0150	1343.6044	9510.1604	9.5296	3.7539	2.9715
$\|\|z\ Real - \ z\ Interp\|\|$	214996.88	319800.81	975526.87	3933.1511	1805.6087	1387.2983

Taula 5. Diferents Errors calculats en les interpolacions RBF, segons el paràmetre β.

A continuació podem veure el resultat de la interpolació amb l'error més petit.

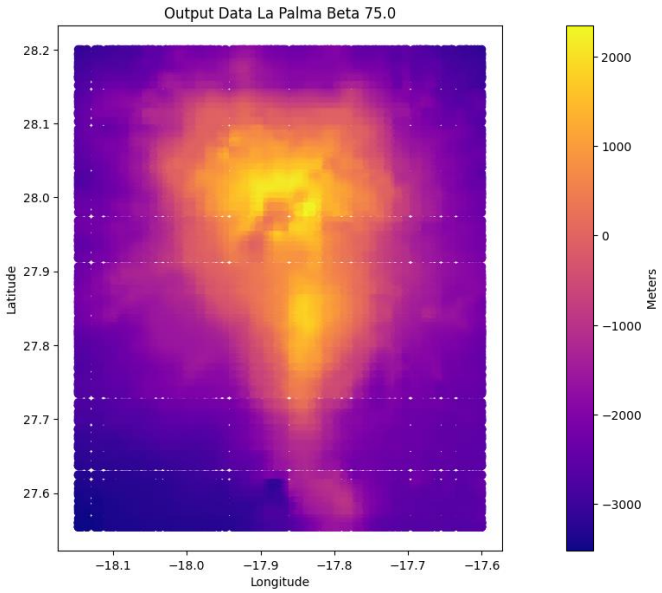


Fig. 27 Resultats de RBF amb un β de 75.

4.3 KRIGING INTERPOLATION RESULTATS

Com hem vist a l'apartat 3.1.1, el primer que farem és fer el càlcul de les distàncies paramètriques entre cada una de les parelles de punts com veiem a la Fig. 28.

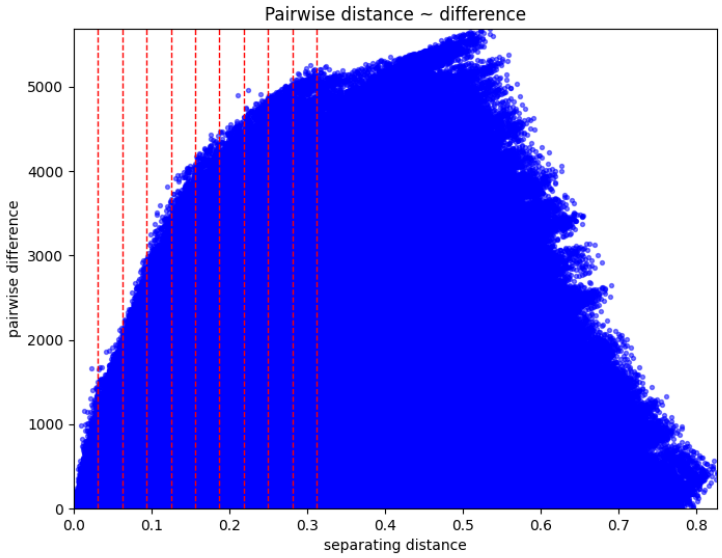


Fig. 28 Parametric Distace (cada punt correspona a la distancia entre una parella de punts i la seva diferencia de Z).

A continuació s'han decidit diferents $lags$(numero de punts) pels quals voldrem separar el SemiVariograma empíric, com són 10, 20, i 30 $lags$ (Fig. 29). D'aquesta manera ja podrem veure quina funció s'ajusta millor al SemiVariograma empíric, juntament amb aquest també s'ha calculat el RMSE per tenir una idea anticipada de com de fiable serà cada una dels SemiVariogrames teòrics que s'utilitzaran per fer les interpolacions.

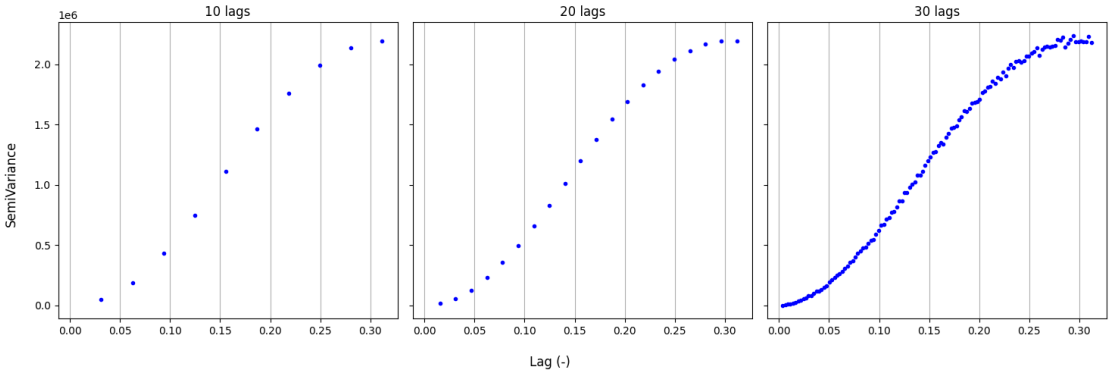


Fig. 29 SemiVariograma Empíric amb diferents valors de $lags$.

Com podem veure a la Fig. 30, a priori veient els ajustaments dels SemiVariogrames diríem que el mètode Estable seria el que millor s'ajusta seguit per el model Gaussià.

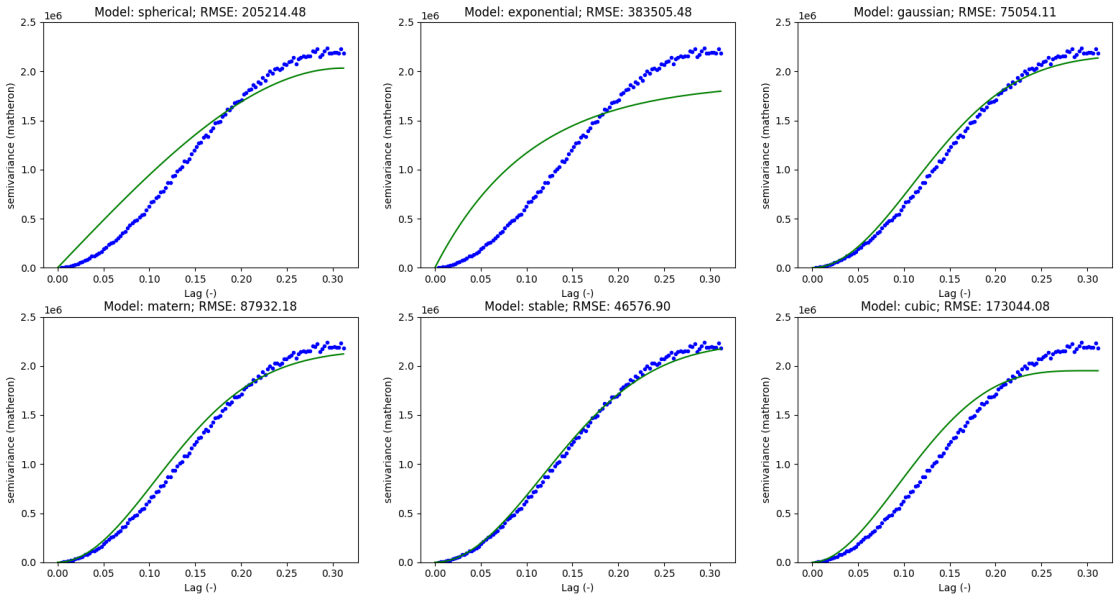


Fig. 30 SemiVariogrames empírics i teòrics amb el càlcul del RMSE corresponent.

Una vegada ja tenim els SemiVariogrames teòrics amb les seves funcions definides es farà la interpolació per a cada un dels models seleccionats, i un cop més calcularem l'error en base als punts de validació comentats prèviament aconseguint els resultats de la taula següent.

	Kriging					
	Spherical	Exponential	Gaussian	Matern	Stable	Cubic
Absolute Error	74.9663	75.7820	263.9398	230.2162	100.7198	87.1327
Relative Error	2.5443	2.5952	10.7858	9.3413	3.2734	2.8298
$\|\|z\ Real - \ z\ Interp\|\|$	1086.3649	1098.1861	3824.8515	3336.1493	1459.5684	1262.6732

Taula 6. Diferents Errors calculats en les interpolacions Kriging Ordinari, segons els diferents models.

A continuació podem veure el resultat de la interpolació que té l'error més petit.

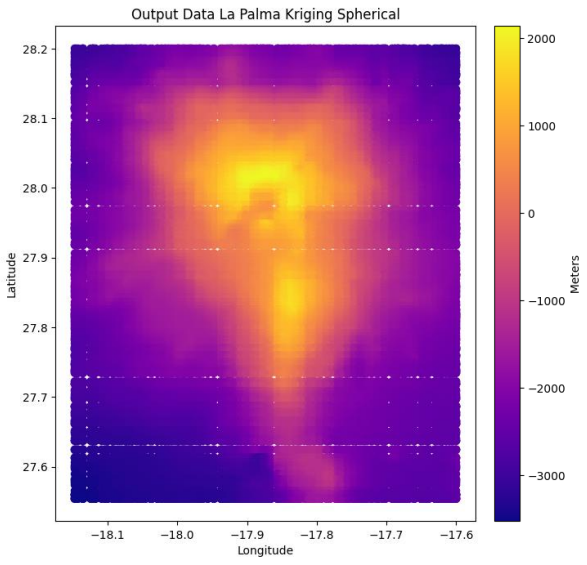


Fig. 31 Resultat de la interpolació per Kriging Esfèric.

4.4 RESULTATS I PARÀMETRES PER LA CREACIÓ DEL FITXER STL

Una vegada ja sabem quina de les interpolacions volem imprimir, el que farem és executar el codi de Python 3 que s'encarrega de la generació d'un primer fitxer STL de la superfície de la interpolació seleccionada. És important que en aquest moment ajustem les dimensions dels eixos XYZ perquè es pugui apreciar bé els resultats. En el nostre cas que tenim que els eixos XY estan en graus de latitud i longitud, amb una diferència de dos graus com a molt, i que l'eix de les Z està en metres, amb una diferència de 6000 metres. Si representéssim els resultats tal com els tenim els que veuríem en el STL seria una columna de punts molt alta on no podriem diferenciar res, és per això que en casos com aquesta, amb coordenades geogràfiques, que redimensionem l'eix Z, en aquest cas el dividim per 3000 cada un dels valors del vector de zetes.

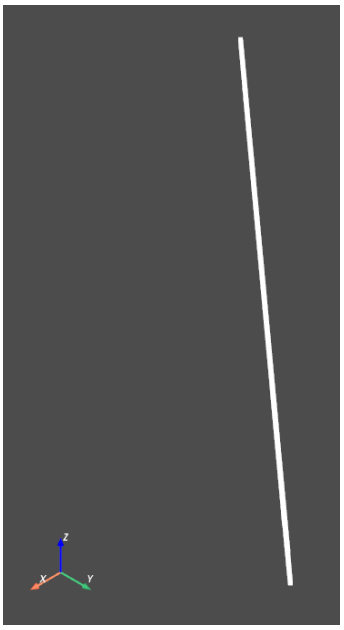


Fig. 32 Resultat visual després de fer Delaunay sense fer el redimensionat en l'eix Z.

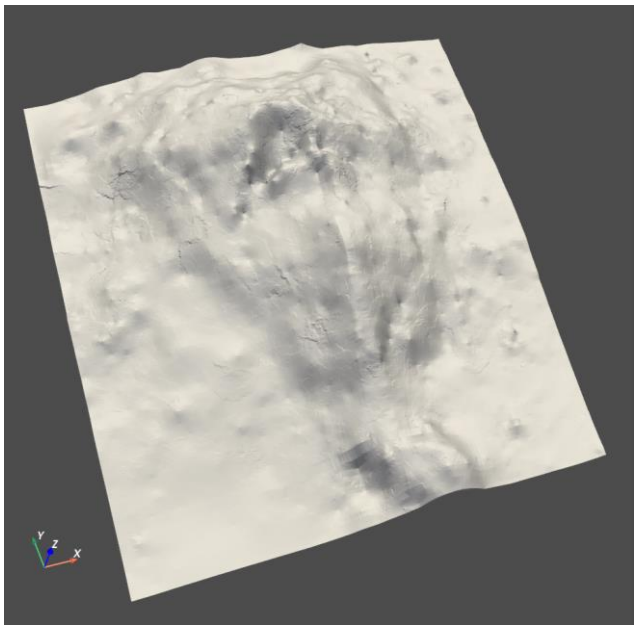


Fig. 33 Resultat visual després de fer Delaunay fent el redimensionat en l'eix Z.

Seguidament obrim el fitxer STL al software MeshMixer i seguim els passos indicats a l'apartat 3.2.2. I en el moment que anem a donar el gruix al STL, a l'opció d' "Extrude", en el nostre cas els valors de les variables són les següents:

- Offset: 10 mm.
- Harden: 50 %.
- Density: 20 %.
- Direction: "Y Axis".
- EndType: "Offset".

Un cop ja tenim el STL amb el seu corresponent gruix, només caldrà separar el STL en diferents STL per tal de poder-lo imprimir a més escala. El factor limitant de la mida d'aquestes particions és el màxim de la impressora, en el nostre cas és de 210 x 297 x 210 mm, el factor limitant són les eixos X i Y.

Aquest STL de l'illa de La Palma en concret s'ha dividit en 8 subfitxers, dos que corresponen a la part topogràfica i les altres 6 a la part batimètrica. Per tal de separar els dos grups principals el que fem és tallar al nivell del mar. Per la resta, en el cas de les 6 parts batimètriques s'ha optat per realitzar els 6 subfitxer de les mateixes dimensions en X i Y, i pels altres dos de la part

topogràfica, hem seguit un criteri per tal d'evitar errors en les impressions intentant deixar tota la part amb mes canvis d'altimetria en la peça més gran i la resta en una més petita, degut a que no hi cabia tot en la primera.

La decisió de fer les particions d'aquesta manera és deu a què en el moment de la impressió d'aquest tipus d'estructures es gastava molt de material per generar els suports de les partes més elevades. Separant la topografia de la batimetria en el moment de la impressió ens estalviem un 15% del material a utilitzar en aquesta maqueta.

Un cop ja tenim totes les peces impreses i enganxades obtindríem un resultat com el que es mostra en la Fig. 34.

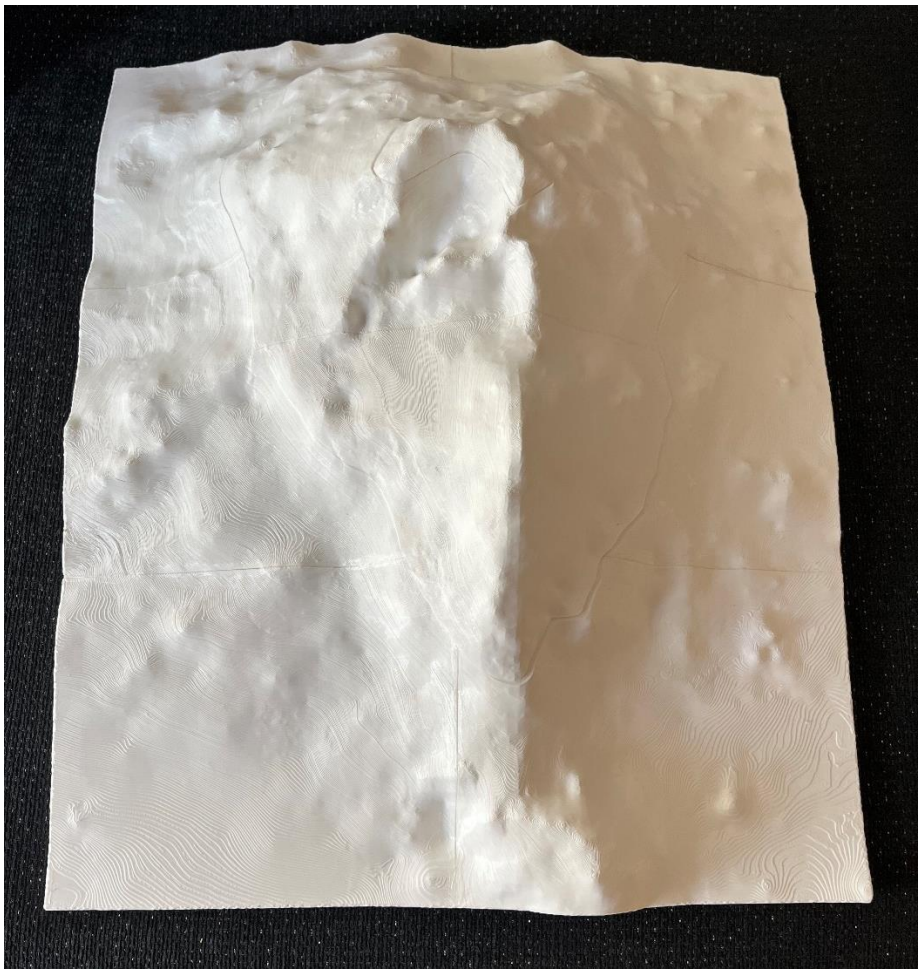


Fig. 34 Resultat final de les 8 impressions un cop muntades.

4.5 IMPRESSIÓ 3D DINS EL SANDBOX

Podem veure com queda la impressió del 8 fitxer STL muntats dins del SandBox. En la primera Fig. 35 veiem només les isolínies d'altimetria, les quals no diferencien entre la part submergida de la part que no ho està. Per altra banda, en la Fig. 36 veiem la mateixa representació que abans, però, aquest cop s'ha augmentat el nivell del mar fins a la cota corresponent a zero metres del NMM (Nivell Mitjà del Mar).

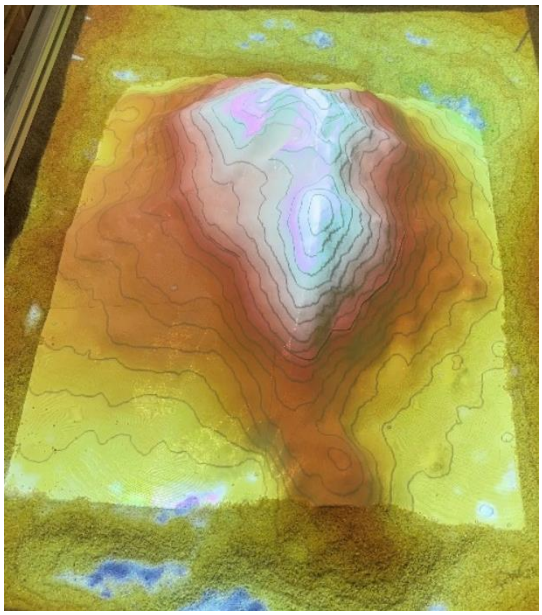


Fig. 35 Projecció de les isolínies del SandBox sobre la impressió 3D.

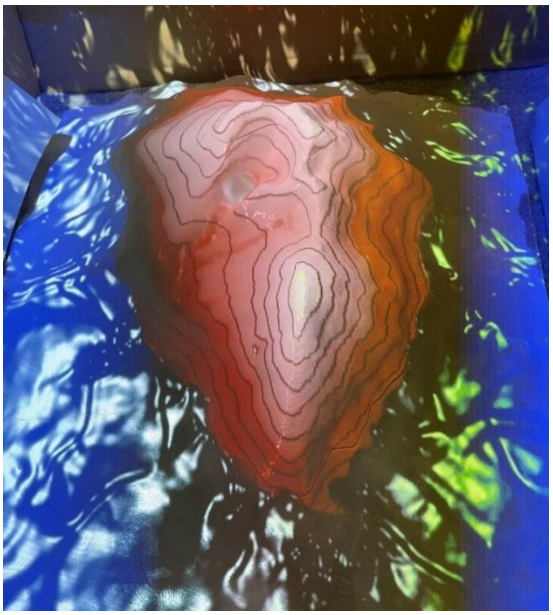


Fig. 36 Projecció de la Fig. 35 augmentant el nivell del mart fins a la cota 0 respecte al NMM.

5 DISCUSSIONS I CONCLUSIONS

Com hem vist, a l'hora de realitzar les interpolacions s'ha de decidir entre tres models i per cada un d'ells també s'han de decidir paràmetres o models d'ajustament dins de cada un d'ells. És per això que s'ha intentat donar la millor opció a escollir, però això no es pot fer, ja que cada DataSet és diferent i necessitarà diferents paràmetres per trobar el millor ajustament. El que sí s'ha aconseguit és considerar la millor elecció de paràmetres o el millor ajustament pels mètodes amb els quals s'ha treballat.

El primer de tots els models utilitzats del que parlarem serà l'Inverse Distance Weighting Interpolation (IDW). És un model que, com hem vist, es basa en la informació dels veïns que es consideren dins d'un radi de cerca. Com menor sigui el radi, millors resultats obtindrem, a condició que el radi sigui prou gran per a poder considerar algun veí dins la zona. Com més gran és el radi tindrem més veïns que donen informació, estarem repartint els pesos entre tots ells i això acaba sent contraproduent, ja que traiem importància als veïns més propers que realment són els que ens interessen.

Com hem vist en la Fig. 26, la diferència entre interpolacions utilitzant Pitàgores i l'Eq. 14 és gairebé nul·la. Per tant, podem dir que en zones properes a l'equador, com són les nostres zones d'estudi, podem fer servir Pitàgores directament sobre coordenades geogràfiques. La qual cosa ens reduirà molt el temps de computació.

Per al cos del Radial Basis Function Interpolation (RBF) tenim dos paràmetres que influiran en el càlcul de la interpolació, el tipus de kernel i el valor que li donem a la β. En aquest cas s'ha fet l'estudi amb el kernel de tipus exponencial i s'ha buscat la millor β en funció de les dades. En termes generals, en tots els data sets utilitzats veiem una tendència a què com més elevat és el valor de la β millor són les interpolacions. El valor de beta depèndra de les unitats que s'utilitzin. El fet que la β sigui més gran farà que els valors propers als kernels tinguin una variabilitat molt petita respecte al centre del kernel.

Finalment, en l'últim mètode, el Kriging Ordinari, tenim diferents models. A priori, veient el comportament dels SemiVariogrames podríem dir que clarament els models d'ajustament Matern, Estable i Gaussià seran els que millors resultats donaran. Però el que millor s'ajusta a totes les interpolacions realitzades és el model Esfèric, cosa que ja s'havia comentat anteriorment.

La generació del fitxer STL per imprimir és un tema complicat, ja que no ha estat fàcil degut a la manca de bibliografia específica d'aquest tema.

Finalment, s'ha generat un workflow molt complet des de l'inici fins a la generació del STL i la seva impressió, on es contemplen molts punts d'error que podem trobar i com solucionar-los i tot amb programes de llicència gratuïta, que era un dels principals reptes que es van platejar en aquest projecte.

Cal comentar que per facilitat de comprensió i utilització els millors mètodes personalment són l'IDW i el Kriging Ordinari.

6 LIMITACIONS I FUTUR PRÒXIM

Les principals limitacions durant tot el procés del projecte han estat a la part més practica, de programació i de 3D. Tota la part de programació s'ha fet amb Python, que és un software lliure, complint així un dels objectius principals del projecte, no necessitar cap mena de llicencia o comprar cap programa per realitzar aquest WorkFlow. La limitació principal amb el llenguatge de programació és que en el Grau de Ciències i Tecnologies del Mar de la UPC no es programa en Python, pel contrari s'acostuma a utilitzar MatLAB. Però no és fins a l'últim quadrimestre a la menció de Tecnologies a Vilanova i la Geltrú, que es comença a utilitzar Python. El tipus de codi que s'ha utilitzat en el projecte requeria coneixements avançats, que finalment es van poder adquirir. Tot i així, el procés d'interpolació complet on s'interpola amb diferents paràmetres per a una posterior comparació requereix de molt de temps de computació per a un ordinador convencional. És per això que, gracies a la universitat, s'ha tingut accés a un servidor capaç de suportar aquesta càrrega de feina i realitzar totes les interpolacions de manera raonablement ràpida.

L'altre principal limitació ha estat la impressora 3D que molt amablement ens ha deixat el servei TIC de la UPC de Camins. És un model relativament petit i, per les dimensions en què volíem imprimir, s'han hagut de fer talls en els models per tal de poder-los imprimir tots i posteriorment unir-los.

Futur Pròxim

Com a futura línia de treball en aquest projecte, en el qual seguiré treballant, l'objectiu principal és deixar-lo del tot optimitzat i llest per la seva utilització com a programari lliure per a qualsevol usuari. Tot i que hi ha vàries parts del codi que ja han estat correctament optimitzades, n´hi ha d'altres en les que encara es poden fer millores, per tal que aquest programari es pugui fer servir des de qualsevol dispositiu sense cap problema.

7 BIBLIOGRAFIA

Adhikary, S. K., Muttil, N., & Yilmaz, A. G. (2016). Ordinary kriging and genetic programming for spatial estimation of rainfall in the Middle Yarra River catchment, Australia. *Hydrology Research*, *47*(6), 1182–1197. https://doi.org/10.2166/nh.2016.196

All3DP. (2021). *The STL File Format Simply Explained*. Oct 28, 2021. https://all3dp.com/1/stl-file-format-3d-printing/

ArcGIS. (n.d.). *Cómo funciona Kriging*. Retrieved April 11, 2022, from https://desktop.arcgis.com/es/arcmap/10.3/tools/3d-analyst-toolbox/how-kriging-works.htm

BCN3D. (n.d.). *Manual De Usuario Sigma R19*. https://www.bcn3d.com/documents/Manual_de_Usuario_Sigma_R19.pdf

Burrough, P. A., & McDonnell, R. A. (1998). *Principles of Geographical Information Systems*. Oxford University Press.

Cheng, S.-W., Dey, T. K., & Shewchuk, J. (2013). Delaunay Mesh Generation. *CRC Press*.

Córdoba, M., Paccioretti, P. A., Giannini Kurina, F., Bruno, C. I., & Balzarini, M. G. (2019). *Guía para el análisis de datos espaciales en agricultura*.

Gabri. (2018). *¿Cómo funciona el semivariograma en la interpolación?* https://acolita.com/como-funciona-semivariograma-interpolacion/

GISGeography. (2022). *Semi-Variogram: Nugget, Range and Sill*. https://gisgeography.com/semi-variogram-nugget-range-sill/

Introduction to Spatial Analysis. (n.d.). Retrieved April 11, 2022, from https://planet.uwc.ac.za/nisl/gis/spatial/chap_1_31.htm

Karayiannis, N. B., & Randolph-Gips, M. M. (2003). On the construction and training of reformulated radial basis function neural networks. *IEEE Transactions on Neural Networks*, *14*(4), 835–846. https://doi.org/10.1109/TNN.2003.813841

Lu, G. Y., & Wong, D. W. (2008). An adaptive inverse-distance weighting spatial interpolation technique. *Computers and Geosciences*, *34*(9), 1044–1055. https://doi.org/10.1016/j.cageo.2007.07.010

Mälicke, M. (2021). *Variogram models*. https://scikit-gstat.readthedocs.io/en/latest/reference/models.html?highlight=Variogram

Montero, J. M., Fernández-Avilés, G., & Mateu, J. (2012). Spatial and Spatio-Temporal Geostatistical Modeling and Kriging. In *Spatial and Spatio-Temporal Geostatistical Modeling and Kriging*. https://doi.org/10.1002/9781118762387

Plantamura F, & Oberti I. (2015). Is 3D Printed House Sustainable? *Proceedings of International Conference CISBAT 2015 Future Buildings and Districts Sustainability from Nano to Urban Scale*, 173–178. https://infoscience.epfl.ch/record/213312

Reed, S., Hsi, S., Kreylos, O., Yikilmaz, M. B., Kellogg, L. H., Schladow, S. G., Segale, H., & Chan, L. (2016). Augmented reality turns a sandbox into a geoscience lesson. *Eos*, 97. https://doi.org/https://doi.org/10.1029/2016EO056135

Schubert, C., Van Langeveld, M. C., & Donoso, L. A. (2014). Innovations in 3D printing: A 3D overview from optics to organs. *British Journal of Ophthalmology*, *98*(2), 159–161. https://doi.org/10.1136/bjophthalmol-2013-304446

Serreta Oliván, A., & Playán Jubillar, E. (1993). *MÉTODOS DE INTERPOLACIÓN DE ALTIMETRÍA PARA LA GENERACIÓN DE MALLAS TOPOGRÁFICAS REGULARES EN PARCELAS DE RIEGO POR SUPERFICIE* (pp. 173–187).

Simmons, B. (2017). *Circumcircle Circumscribed Circle*. 19-Jul. Oct 28, 2021

Statistics How To. (n.d.-a). *Mean Squared Error: Definition and Example*. Retrieved May 17, 2022, from https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error

Statistics How To. (n.d.-b). *RMSE: Root Mean Square Error*. Retrieved May 17, 2022, from https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error

Tobi, A. L. M., Omar, S. A., Yehia, Z., Al-Ojaili, S., Hashim, A., & Orhan, O. (2018). Cost viability of 3D printed house in UK. *IOP Conference Series: Materials Science and Engineering*, *319*(1). https://doi.org/10.1088/1757-899X/319/1/012061

Whittlesey, M. (2020). Spherical Geometry and its Applications. *CRC Press*, 120.

Wright, G. B. (2003). Radial Basis Function Interpolation : Numerical and Analytical Developments. *University of Colorado*.

Yoo, S. S. (2015). 3D-printed biological organs: Medical potential and patenting opportunity. *Expert Opinion on Therapeutic Patents*, *25*(5), 507–511. https://doi.org/10.1517/13543776.2015.1019466

8 ÍNDEX DE FIGURES

9 ANNEX

9.1 ALTRES RESULTATS

9.1.1 CANONS SUBMARINS

Input Points Data = 4320

Validation Points Data = 432

Output Points Data = 272 484

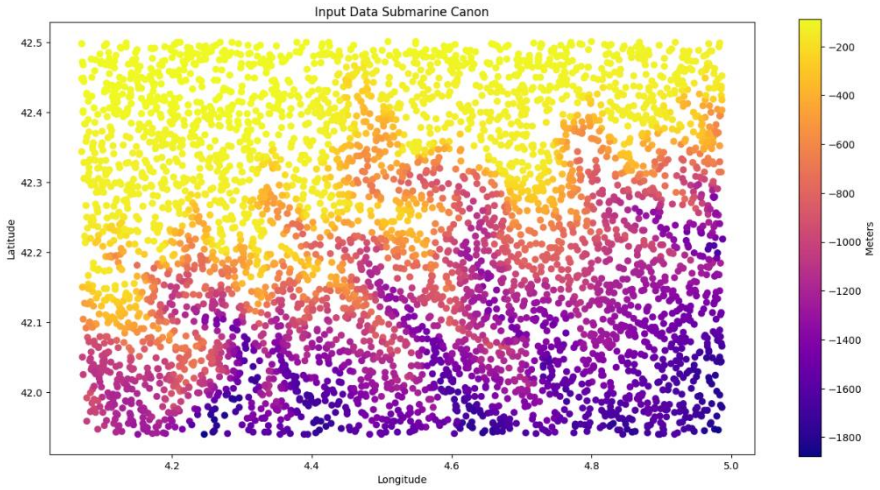


Fig. 37 Representació visual del Data Set dels Canons Submarins.

IDW

	IDW Radial Distances (Km)					
	1.00	2.50	5.00	10.0	15.0	20.0
Absolute Error	NaN	43.8010	60.5574	79.8936	87.9803	93.3535
Relative Error	NaN	4.4342	6.9887	9.8385	10.9309	11.8451
$\|\|z\ Real - \ z\ Interp\|\|$	NaN	910.3877	1258.6621	1660.5591	1828.6365	1940.3173

Taula 7. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

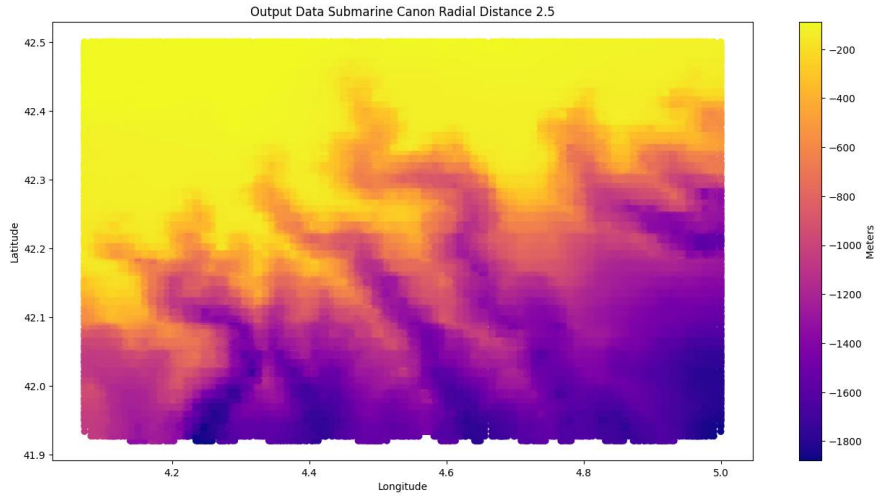


Fig. 38 Resultats de IDW amb Coordenades Geogràfiques i Radi 2.50 Km.

RBF

	RBF β					
	2.50	5.00	10.0	25.0	50.0	75.0
Absolute Error	27729.596	12810.184	21022.898	113.4269	48.1130	36.1209
Relative Error	2351.8594	1254.4651	1487.2614	10.5290	4.5333	3.4686
$\|\|z\ Real - \ z\ Interp\|\|$	576348.83	266254.67	436952.74	2357.5358	1000.0117	750.7602

Taula 8. Diferents Errors calculats en les interpolacions RBF, segons el paràmetre β.

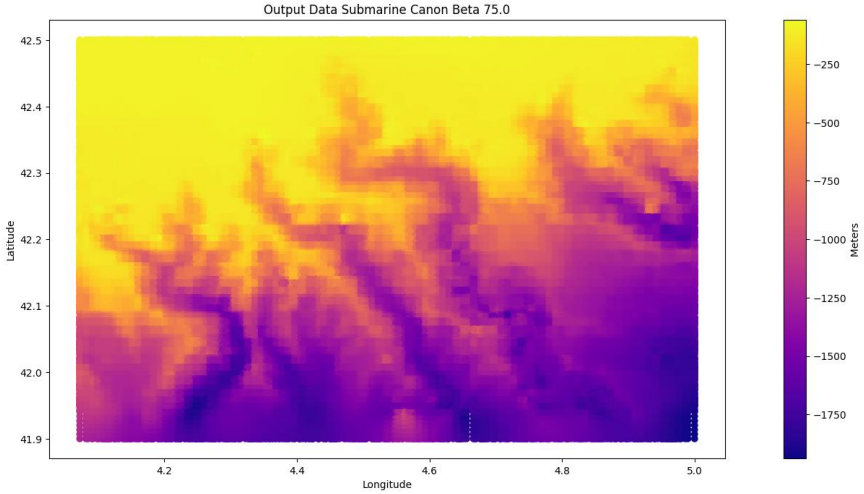


Fig. 39 Resultats de RBF amb un β de 75.

KRIGING

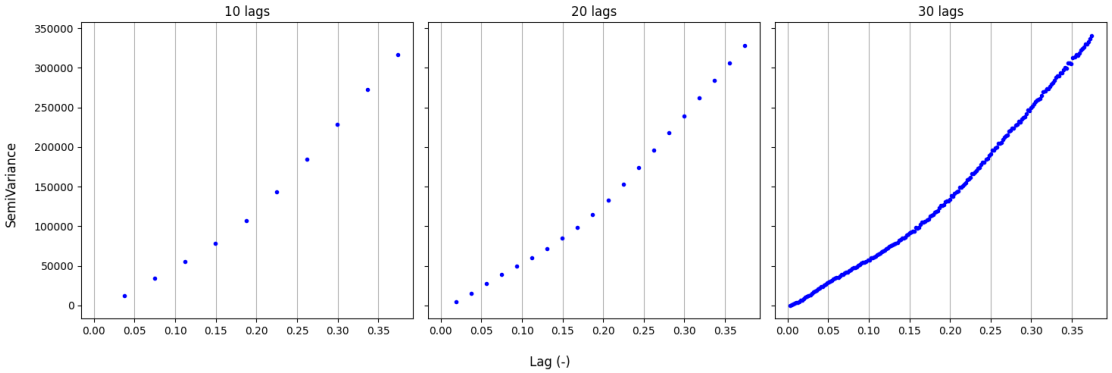


Fig. 40 SemiVariograma Empíric amb diferents valors de lags.

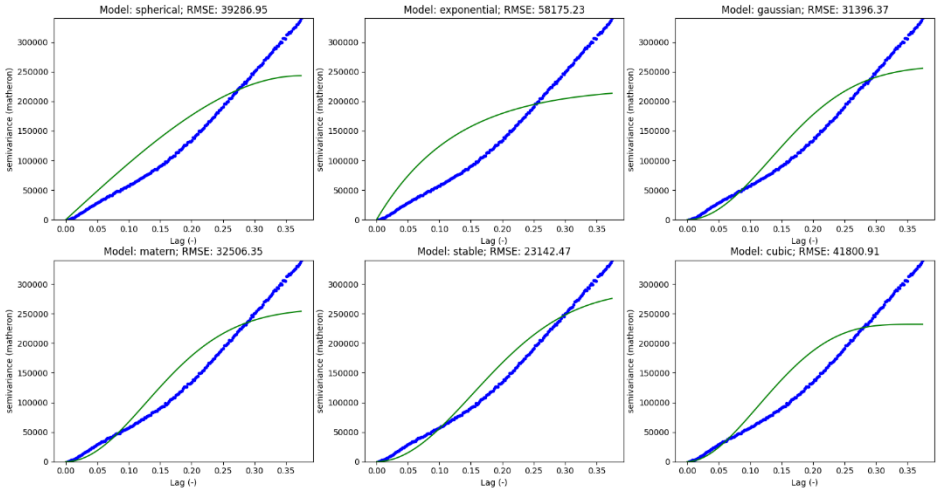


Fig. 41 SemiVariogrames empírics i teòrics amb el càlcul del RMSE corresponent.

	Kriging					
	Spherical	Exponential	Gaussian	Matern	Stable	Cubic
Absolute Error	30.6809	30.7142	84.9336	78.2171	92.7064	28.2086
Relative Error	2.4662	2.4693	7.9515	7.3791	8.5358	2.3270
$\|\|z\ Real - \ z\ Interp\|\|$	637.6914	638.3839x	1765.3134	1625.7129	1926.8683	586.3067

Taula 9. Diferents Errors calculats en les interpolacions Kriging Ordinari, segons els diferents models.

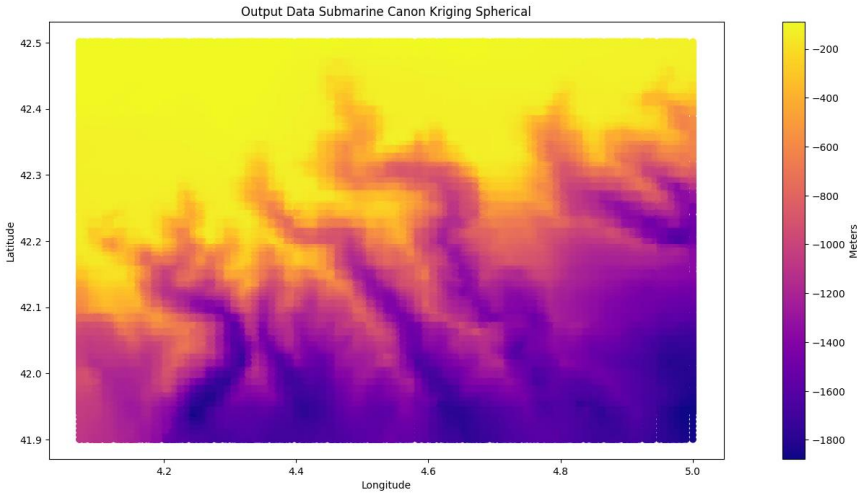


Fig. 42 Resultat de la interpolació per Kriging Esfèric.

9.1.2 VILANOVA I LA GELTRÚ

Input Points Data = 1221

Validation Points Data = 123

Output Points Data = 68644

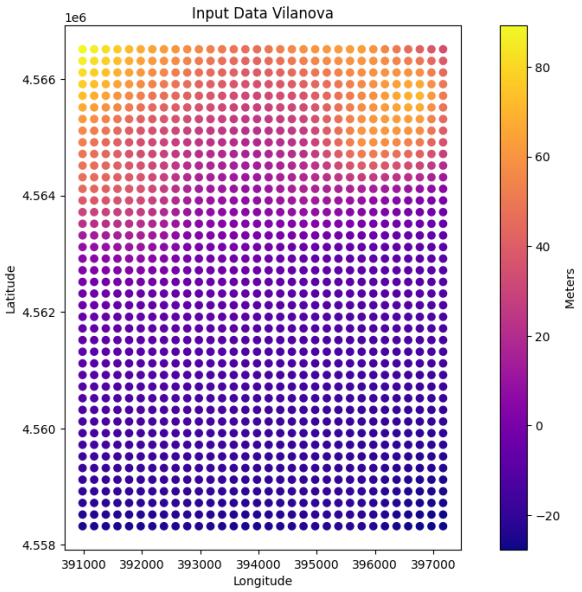


Fig. 43 Representació visual del Data Set de Vilanova i la Geltrú.

IDW

	IDW Radial Distances (m)					
	175	200	225	250	275	300
Absolute Error	NaN	0.7910	0.7910	0.7910	0.7910	0.7175
Relative Error	NaN	1.6585	1.6585	1.6585	1.6585	2.0759
$\|\|z\ Real - \ z\ Interp\|\|$	NaN	8.7729	8.7729	8.7729	8.7729	7.9583

Taula 10. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

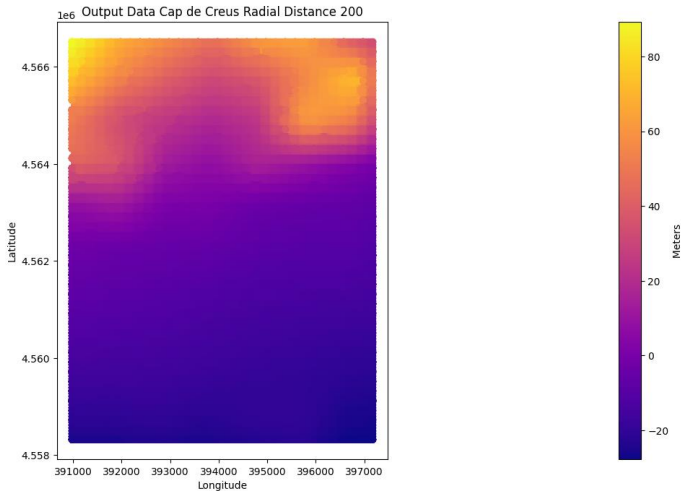


Fig. 44 Resultats de IDW amb Coordenades UTM i Radi 200 metres.

RBF

	RBF β					
	0.10	0.25	0.50	0.75	1.00	2.50
Absolute Error	0.2649	0.2707	0.2726	0.2733	0.2736	0.2742
Relative Error	0.4544	0.4675	0.4720	0.4735	0.4742	0.4756
$\|\|z\ Real - \ z\ Interp\|\|$	2.9388	3.0025	3.0242	3.0315	3.0351	3.0417

Taula 11. Diferents Errors calculats en les interpolacions RBF, segons el paràmetre β.

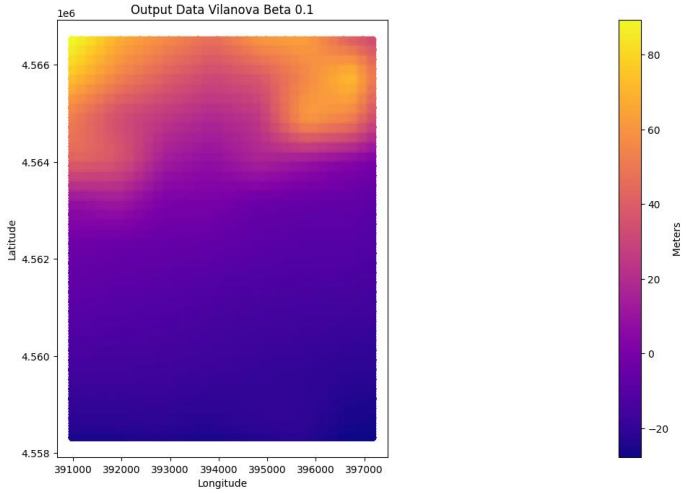


Fig. 45 Resultats de RBF amb un β de 0,10.

KRIGING

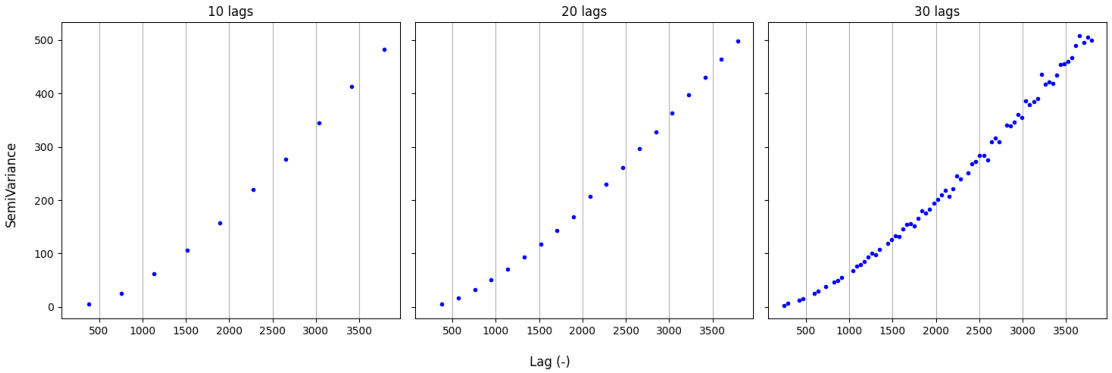


Fig. 46 SemiVariograma Empíric amb diferents valors de lags.

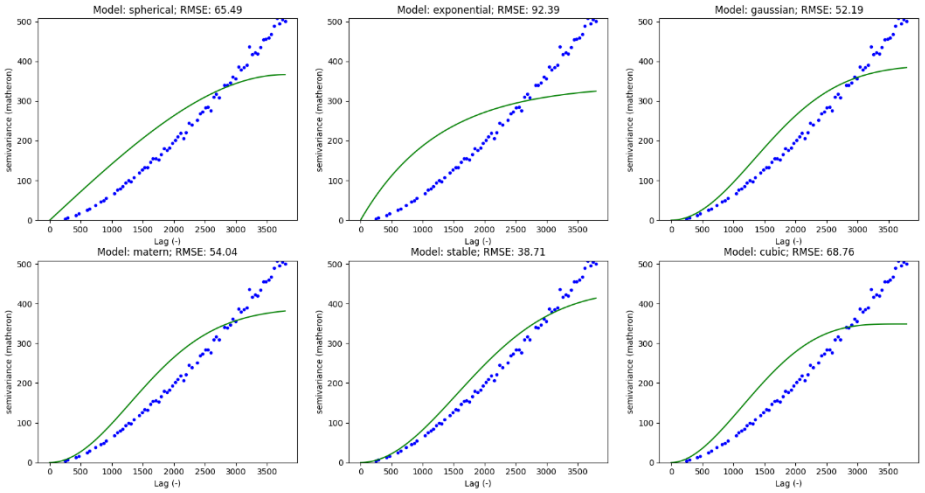
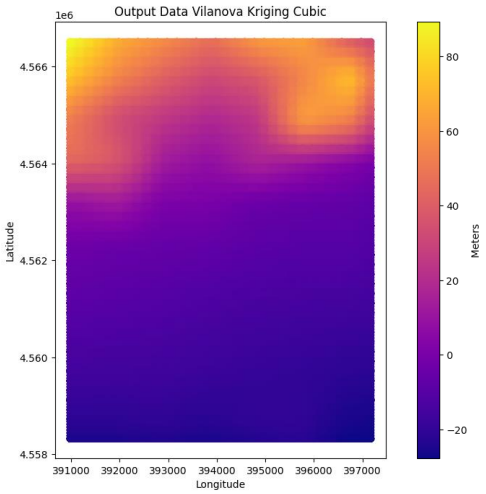


Fig. 47 SemiVariogrames empírics i teòrics amb el càlcul del RMSE corresponent.

	Kriging					
	Spherical	Exponential	Gaussian	Matern	Stable	Cubic
Absolute Error	0.3515	0.3793	0.2281	0.1970	0.2464	0.1177
Relative Error	0.6789	0.7810	0.3177	0.2766	0.3419	0.2134
$\|\|z\ Real - \ z\ Interp\|\|$	3.8992	4.2076	2.5299	2.1854	2.7327	1.3060

Taula 12. Diferents Errors calculats en les interpolacions Kriging Ordinari, segons els diferents models.



.

Fig. 48 Resultat de la interpolació per Kriging Cubic.

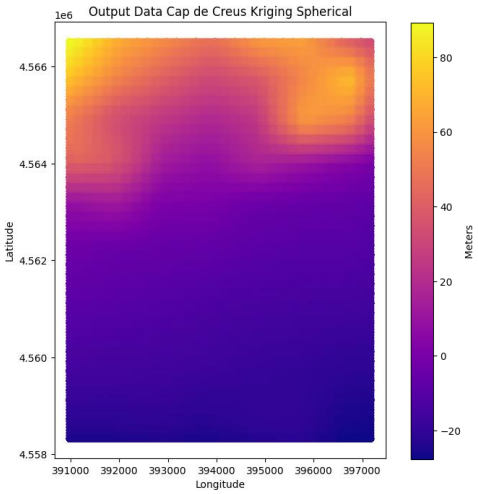


Fig. 49 Resultat de la interpolació per Kriging Esfèric.

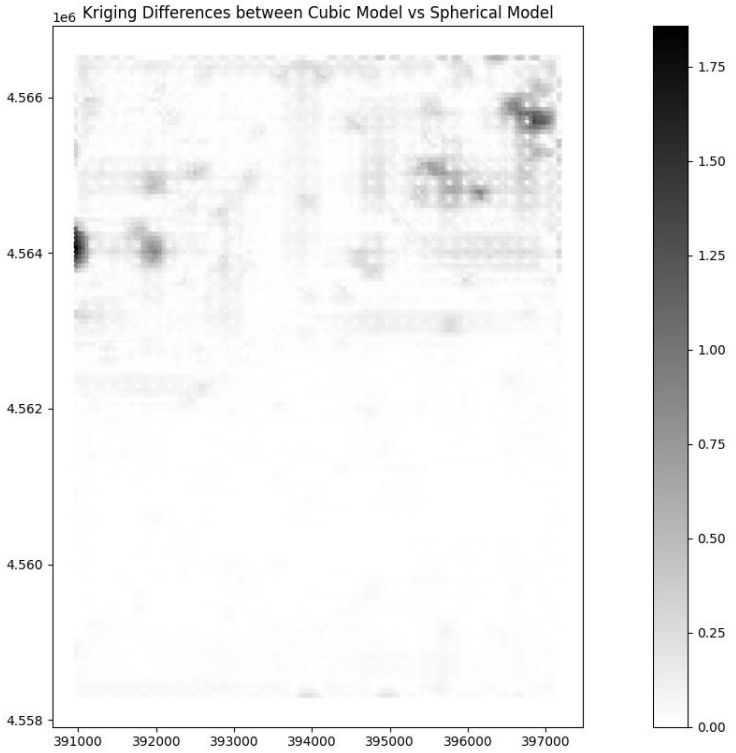


Fig. 50 Comparativa entre les interpolacions de la Fig. 48 Resultat de la interpolació per Kriging Cubic.Fig. 48 i de la Fig. 49, on veiem que la diferencia és mínima.

9.1.3 CAP DE CREUS

Input Points Data = 2854

Validation Points Data = 286

Output Points Data = 160000

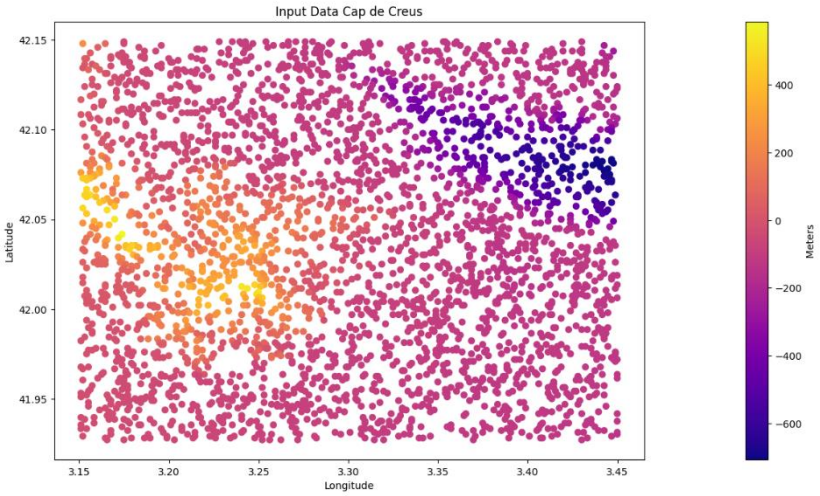


Fig. 51 Representació visual del Data Set de Cap de Creus.

IDW

	IDW Radial Distances (Km)					
	0.50	0.75	1.00	2.50	5.00	10.0
Absolute Error	NaN	18.4942	19.7605	25.8372	33.0762	40.9095
Relative Error	NaN	2.6628	2.8783	4.1495	6.8330	10.3767
$\|\|z\ Real - \ z\ Interp\|\|$	NaN	312.7653	334.1804	436.9475	559.3696	691.8435

Taula 13. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

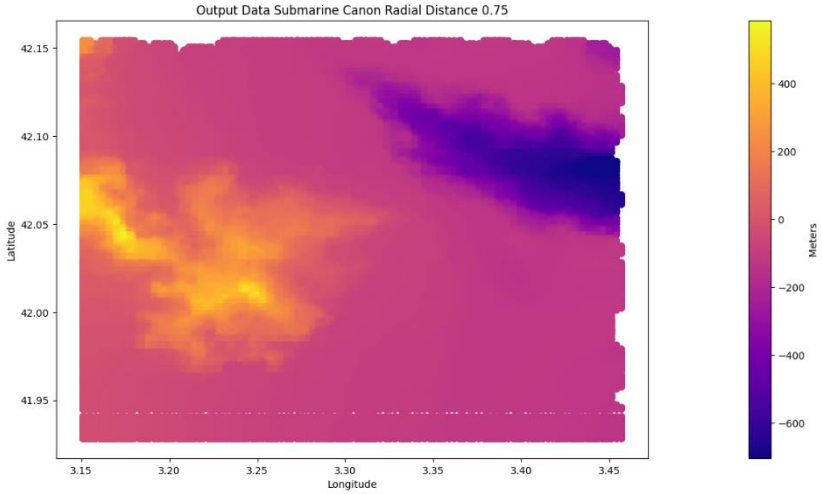


Fig. 52 Resultats de IDW amb Coordenades Geogràfiques i Radi 0.75 Km.

RBF

	RBF β					
	2.50	5.00	10.0	25.0	50.0	75.0
Absolute Error	2755.2682	3078.5692	1406.1776	2272.5138	55.5511	34.1628
Relative Error	403.3842	491.6155	231.1720	305.3757	8.8751	4.8934
$\|\|z\ Real - \ z\ Interp\|\|$	46595.813	52063.329	23780.622	38431.696	939.4545	577.7464

Taula 14. Diferents Errors calculats en les interpolacions RBF, segons el paràmetre β.

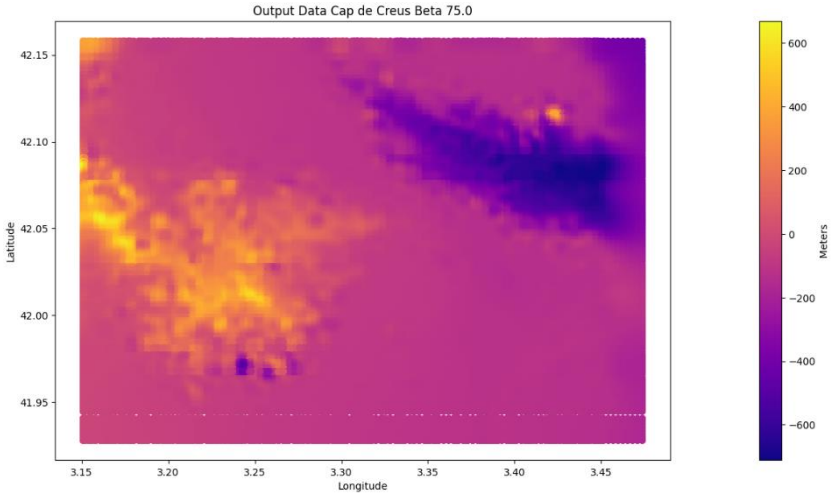


Fig. 53 Resultats de RBF amb un β de 75.

KRIGING

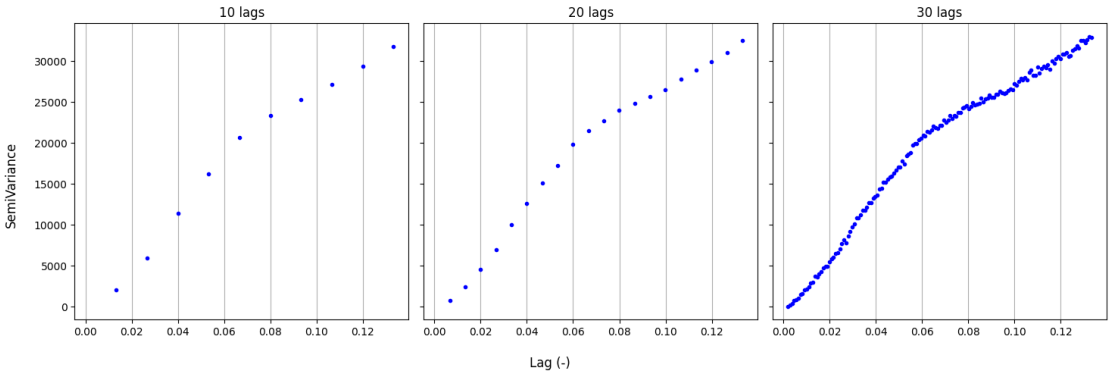


Fig. 54 SemiVariograma Empíric amb diferents valors de lags.

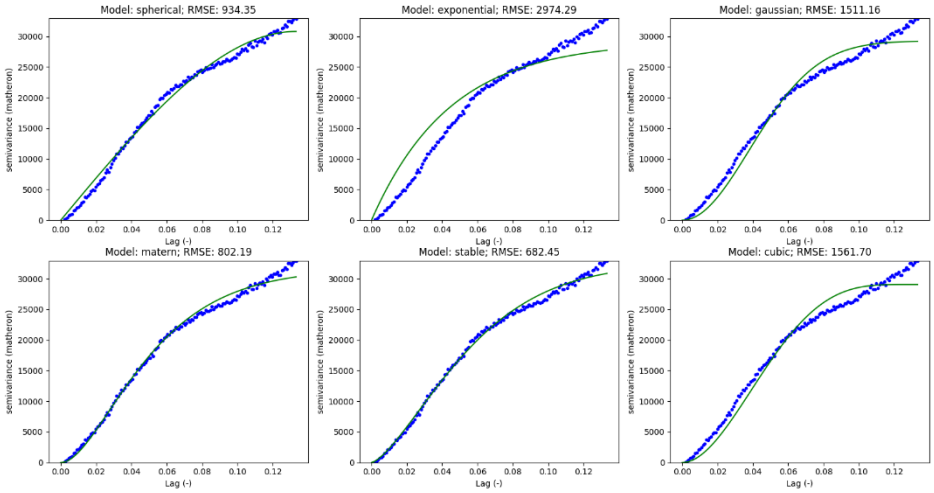


Fig. 55 SemiVariogrames empírics i teòrics amb el càlcul del RMSE corresponent.

	Kriging					
	Spherical	Exponential	Gaussian	Matern	Stable	Cubic
Absolute Error	16.6053	16.63903	45.9315	15.8419	15.7010	15.8485
Relative Error	2.3900	2.3951	6.5934	2.6950	2.3860	2.6928
$\|z\ Real - \ z\ Interp\|$	280.8226	281.3915	776.7734	267.9111	265.5290	268.0232

Taula 15. Diferents Errors calculats en les interpolacions Kriging Ordinari, segons els diferents models.

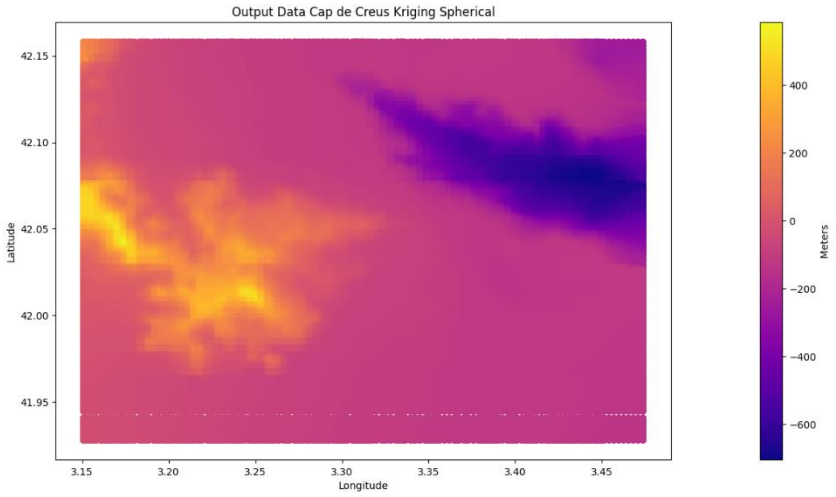


Fig. 56 Resultat de la interpolació per Kriging Esfèric.

9.1.4 GIJÓN

Input Points Data = 561

Validation Points Data = 57

Output Points Data = 127 449

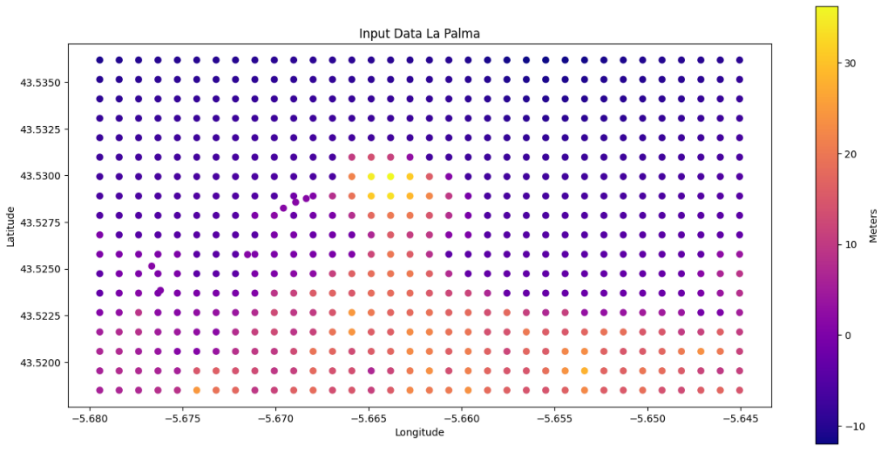


Fig. 57 Representació visual del Data Set de la platja de Gijón.

IDW

	IDW Radial Distances (Km)					
	0.10	0.25	0.50	1.00	2.50	5.00
Absolute Error	2.6766	3.6371	4.6975	5.5016	6.0899	6.1058
Relative Error	11.8573	18.3008	23.1113	29.0691	42.9566	43.2768
$\|\|z\ Real - \ z\ Interp\|\|$	20.2085	27.4597	35.4653	41.5364	45.9781	46.0979

Taula 16. Diferents Errors calculats en les interpolacions IDW, segons el paràmetre Radial Distances.

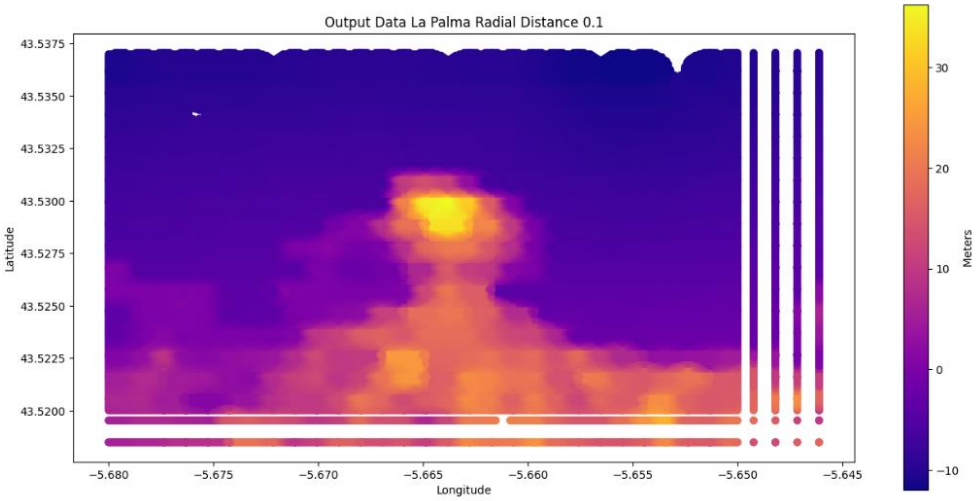


Fig. 58 Resultats de IDW amb Coordenades Geogràfiques i Radi 0.10 Km.

RBF

	RBF β					
	25.0	50.0	75.0	100	150	200
Absolute Error	519.7808	1130.1413	104.2365	40.6662	5.3967	4.0545
Relative Error	6036.0833	20998.649	1310.4851	811.7448	30.0879	19.9041
$\|\|z\ Real - \ z\ Interp\|\|$	3924.2593	8532.3801	786.9687	307.0236	40.7445	30.6111

Taula 17. Diferents Errors calculats en les interpolacions RBF, segons el paràmetre β.

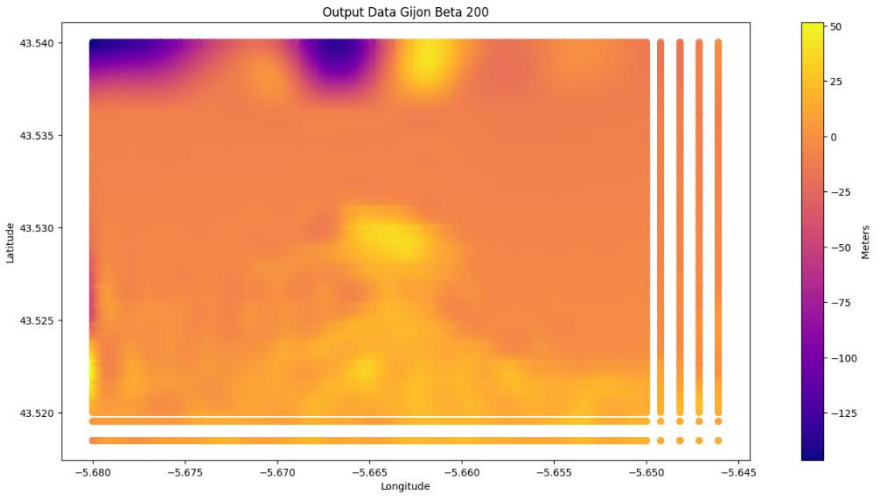


Fig. 59 Resultats de RBF amb un β de 200.

KRIGING

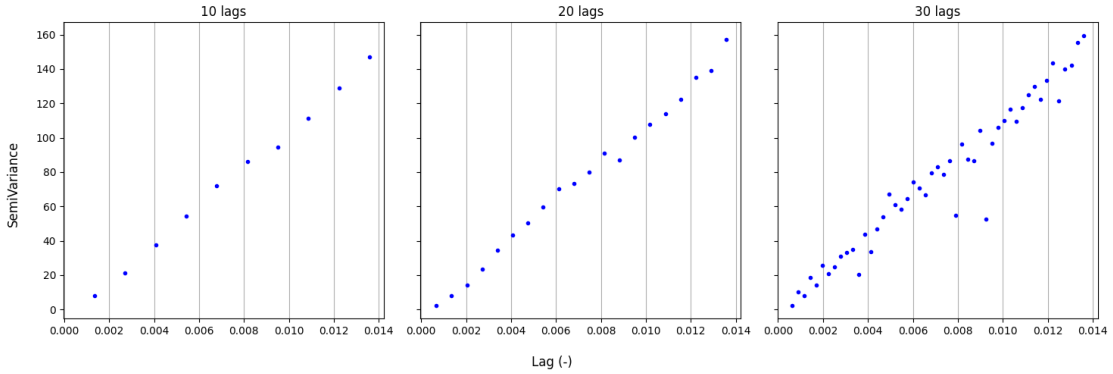


Fig. 60 SemiVariograma Empíric amb diferents valors de lags.

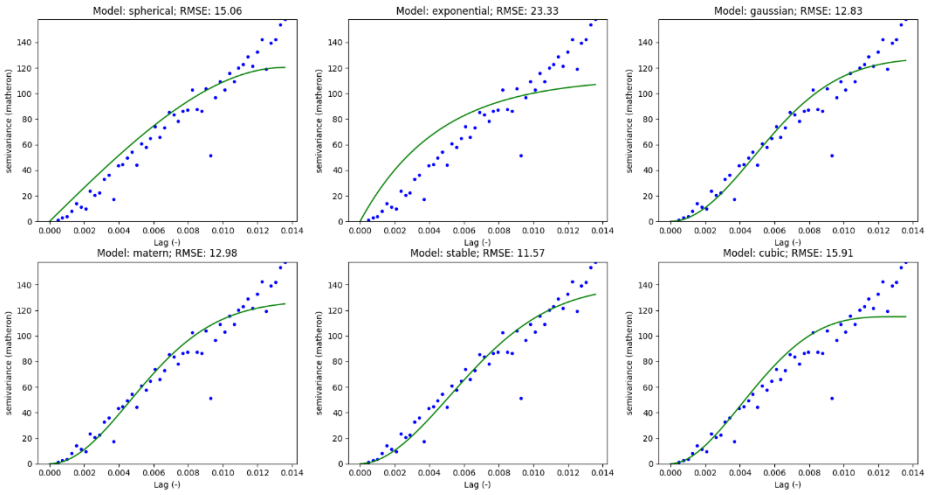


Fig. 61 SemiVariogrames empírics i teòrics amb el càlcul del RMSE corresponent.

	Kriging					
	Spherical	Exponential	Gaussian	Matern	Stable	Cubic
Absolute Error	2.8492	2.8802	3.6019	3.4312	3.7541	2.6898
Relative Error	12.9267	13.1875	21.7695	19.5155	23.7098	12.5136
$\|\|z\ Real - \ z\ Interp\|\|$	21.5113	21.7456	27.1938	25.9050	28.3433	20.3075

Taula 18. Diferents Errors calculats en les interpolacions Kriging Ordinari, segons els diferents models.

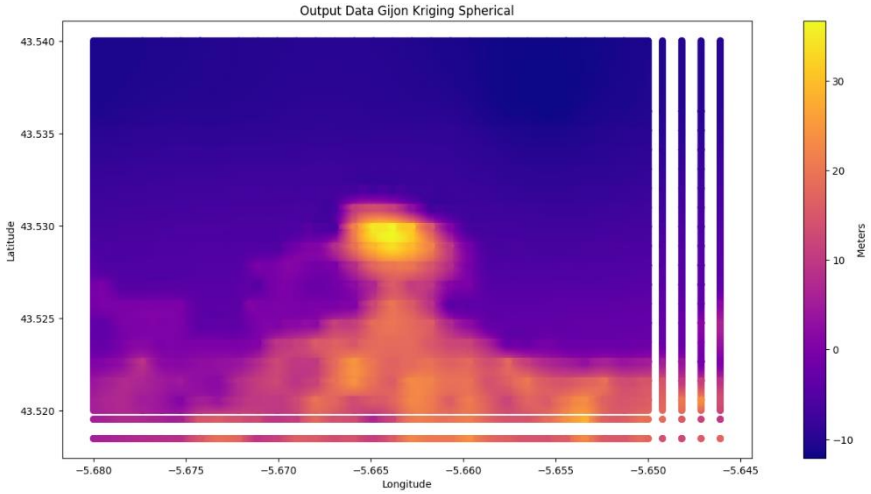


Fig. 62 Resultat de la interpolació per Kriging Esfèric.

9.2 CODI PYTHON

El codi estarà penjat al següent repositoris de GitHub,

https://github.com/PolBanosCastello/Topographic-and-Bathymetric-Interpolation-Models.git

el qual s'anirà actualitzant fins optimitzar-lo el màxim possible, ja que seguiré treballant amb el codi fins a deixar-lo llest per a l'ús públic de qualsevol usuari.

Però per a l'avaluació del treball deixaré el codi en els següents apartats.

9.2.1 IDW_multiprocessing.py

```
import time
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from rich import print as rprint
import numpy as np
import os
from tqdm.auto import tqdm
import math
from math import radians
from concurrent import futures
from rich.progress import Progress
import multiprocessing as mp


def cm_to_inch(value):
    return value/2.54


plt.rcParams["figure.figsize"] = [cm_to_inch(40), cm_to_inch(20)]


class IDW:
    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    max_worker = mp.cpu_count() * 2

    @staticmethod
    def _multiprocess_index_handler(index, handler, args):
        result = handler(*args)  # call the handler
        return index, result  # add index to the result

    def multiprocess(self, arg_list, handler, max_workers=20, text: str = "progress..."):
        index = 0  # thread index
        ctx = mp.get_context('spawn')
        with futures.ProcessPoolExecutor(max_workers=max_workers, mp_context=ctx) as executor:
            processes = []  # empty thread list
            results = []  # empty list of thread results
            for args in arg_list:
                # submit tasks to the executor and append the tasks to the thread list
                processes.append(executor.submit(self._multiprocess_index_handler, index, handler,
args))
                index += 1

            with Progress() as progress:  # Use Progress() to show a nice progress bar
                task = progress.add_task(text, total=index)
                for future in futures.as_completed(processes):
                    future_result = future.result()  # result of the handler
                    results.append(future_result)
                    progress.update(task, advance=1)

            # sort the results by the index added by __threadify_index_handler
            sorted_results = sorted(results, key=lambda a: a[0])

            final_results = []  # create a new array without indexes
            for result in sorted_results:
                final_results.append(result[1])
            return final_results

    def __init__(self, data_frame, df_validation, resolution_factor=10, reduction_scale=1):
        self.df = pd.DataFrame(data_frame.values, columns=['X', 'Y', 'Z'])
        self.reduction_scale = reduction_scale

        self.df['X'] = self.df['X'] * self.reduction_scale
        self.df['Y'] = self.df['Y'] * self.reduction_scale
```

```
        self.num_points = len(self.df)
        self.resolution_factor = resolution_factor
        self.df_validation = df_validation
        self.df_validation['X'] = self.df_validation['X'] * self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] * self.reduction_scale

        x_min, x_max = int(np.round(min(self.df['X']))), int(np.round(max(self.df['X'])))
        y_min, y_max = int(np.round(min(self.df['Y']))), int(np.round(max(self.df['Y'])))

        num_resolution = ((x_max - x_min) * self.resolution_factor)
        x = np.linspace(x_min, x_max, num=int(num_resolution))
        y = np.linspace(y_min, y_max, num=int(num_resolution))

        for a in range(0, len(self.df_validation['X'])):
            x = np.append(x, self.df_validation['X'][a])
            y = np.append(y, self.df_validation['Y'][a])

        self.df['X'] = self.df['X'] / self.reduction_scale
        self.df['Y'] = self.df['Y'] / self.reduction_scale

        self.df_validation['X'] = self.df_validation['X'] / self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] / self.reduction_scale

        x = x / self.reduction_scale
        y = y / self.reduction_scale

        rprint(f'Min-Max X:  {round(min(x), 2)}  -  {round(max(x), 4)}')
        rprint(f'Min-Max Y:  {round(min(y), 2)}  -  {round(max(y), 4)}')
        rprint(f'Size X:     {x.size}    -   Size Y:     {y.size}')

        self.xx, self.yy = np.meshgrid(x, y)
        self.zz = np.empty(self.xx.shape)
        self.zz[:] = np.nan

    def calculate_list_dist(self, xx_ij, yy_ij, type_coordinates, earth_radius=6371):
        list_dist = []
        if type_coordinates == 'GPS':
            for a in range(0, len(self.df['X'])):
                d = earth_radius * np.arccos((np.cos(radians(90-yy_ij)) * np.cos(radians(90-
self.df['Y'][a])) +
                                              np.sin(radians(90-yy_ij)) * np.sin(radians(90-
self.df['Y'][a])) *
                                              np.cos(radians(xx_ij-self.df['X'][a]))))
                if d == 90:
                    rprint(d)
                if d != 0:
                    b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a], d]
                    list_dist.append(b)
                    rprint(b)
                input()

        if type_coordinates == 'UTM':
            for a in range(0, len(self.df['X'])):
                if math.dist((self.df['X'][a], self.df['Y'][a]), (xx_ij, yy_ij)) != 0:
                    b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a],
                         math.dist((self.df['X'][a], self.df['Y'][a]), (xx_ij, yy_ij))]
                    list_dist.append(b)
        df_dist = pd.DataFrame(list_dist, columns=['X', 'Y', 'Z', 'distance'])
        df_dist_by_dist = df_dist.sort_values('distance')
        return df_dist_by_dist

    def execute_method_no_multiprocessing(self,
                                          distance,
                                          file_name,
                                          type_coordinates,
                                          earth_radius=6371,
                                          show_prints=False):

        global dist_0_value
        rprint(f'Execute Inverse Distance Weight Interpolation with Radial Distance  = {distance}')

        rprint(f'Process of IDW method with Radial Distance = {distance} ...')
        for i in tqdm(range(0, self.zz.shape[0])):
            for j in range(0, self.zz.shape[1]):
                if show_prints:
                    print(i, j)
                # t = time.time()
                # rprint('[bold]XX - YY', i, '-', j, self.xx[i, j], self.yy[i, j])
                list_coord_and_dist = []
                list_dist = []

                if type_coordinates == 'GPS':
                    for a in range(0, len(self.df['X'])):
                        d = earth_radius * np.arccos(
                            (np.cos(radians(90 - self.yy[i, j])) * np.cos(radians(90 -
self.df['Y'][a])) +
```

```
                            np.sin(radians(90 - self.yy[i, j])) * np.sin(radians(90 -
self.df['Y'][a])) *
                            np.cos(radians(self.xx[i, j] - self.df['X'][a])))
                        if d != 0:
                            b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a], self.xx[i, j],
self.yy[i, j], d]
                            list_dist.append(b)

                if type_coordinates == 'UTM':
                    for a in range(0, len(self.df['X'])):
                        if math.dist((self.df['X'][a], self.df['Y'][a]), (self.xx[i, j], self.yy[i,
j])) != 0:
                            b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a],
                                 math.dist((self.df['X'][a], self.df['Y'][a]), (self.xx[i, j],
self.yy[i, j]))]
                            list_dist.append(b)

                list_dist = sorted(list_dist)
                list_coord_and_dist = sorted(list_coord_and_dist)

                if show_prints:
                    rprint(f'Radi = {distance}:', len(list_dist))
                    rprint(f'Head list distances: {list_dist[:5]}')

                df_dist = pd.DataFrame(list_dist, columns=['X', 'Y', 'Z', 'distance'])
                df_dist_by_dist = pd.DataFrame(df_dist.sort_values('distance').values,
                                               columns=['X', 'Y', 'Z', 'distance'])
                rprint(df_dist_by_dist)
                input()

                if show_prints:
                    rprint(df_dist_by_dist)

                vect_w = []
                vect_z = []

                dist_0 = False
                for a in range(0, len(df_dist_by_dist)):
                    d = (math.dist((self.df['X'][a], self.df['Y'][a]), (self.xx[i, j], self.yy[i,
j])) ** 2)

                    if d == 0:
                        dist_0 = True

                    else:

                        w = 1 / (df_dist_by_dist['distance'][a] ** 2)
                        vect_w.append(w)
                        vect_z.append(df_dist_by_dist['Z'][a])

                if dist_0:
                    if show_prints:
                        print(f'[red]Radi = {distance}:', 'Diatance 0', '-->',
                              self.xx[i, j], self.yy[i, j], dist_0_value)
                    self.zz[i, j] = dist_0_value
                    time.sleep(1.5)

                else:

                    if len(vect_w) == 0:
                        if show_prints:
                            rprint(f'[yellow]Radi = {distance}:', 'nan')
                        self.zz[i, j] = float("nan")
                        time.sleep(1.5)

                    else:
                        if show_prints:
                            rprint('Before Normalization: Sum W', sum(vect_w), type(sum(vect_w)))
                        vect_w = (vect_w / sum(vect_w))
                        if show_prints:
                            rprint('After Normalization: Sum W', sum(vect_w), type(sum(vect_w)))
                        vect_wz = np.array(vect_w) * np.array(vect_z)
                        self.zz[i, j] = sum(vect_wz)

                if show_prints:
                    a = 1
                    print(a)

        rprint('Interpolation Finish.\nTransform Mesh into DataFrame')

        list_p = []

        for a in range(0, self.zz.shape[0]):
            for b in range(0, self.zz.shape[1]):
                P = [self.xx[a, b], self.yy[a, b], self.zz[a, b]]
                list_p.append(P)

        self.df_new = pd.DataFrame(list_p, columns=['X', 'Y', 'Z'])
```

```
        self.save_method(file_name, dist=distance)

        rprint(f'Finish Radial Distance  = {distance}')

        return self.df_new

    def execute_method(self, distance, file_name, show_prints=False):
        rprint(f'Execute Inverse Distance Weight Interpolation with Radial Distance  = {distance}')

        argument_list = []
        list = np.arange(0, self.max_worker + 1)
        index = 1
        for i in list:
            if i != list[-1]:
                inici = int(self.zz.shape[0] / self.max_worker) * (index - 1)
                final = int(self.zz.shape[0] / self.max_worker) * index
                index += 1
            else:
                inici = int(self.zz.shape[0] / self.max_worker) * self.max_worker
                final = self.zz.shape[0]

            myargs = [distance, file_name, 'UTM', inici, final]
            argument_list.append(myargs)

        rprint(f'Multiprocessing of IDW method with Radial Distance = {distance} ...')

        results = self.multiprocess(argument_list, self.execute_method_multiprocessing,
max_workers=self.max_worker)
        list_res = []
        for result in results:
            list_res += result

        df_lists = pd.DataFrame(list_res, columns=['Z'])
        df_lists.to_csv(f'{self.point_folder}IDW_optim_method.csv',
                        index=False, header=True)

        zz = np.array(list_res).reshape(self.zz.shape[1], self.zz.shape[0])
        rprint(zz)

        rprint('Interpolation Finish.\nTransform Mesh into DataFrame')

        list_p = []

        for a in range(0, self.zz.shape[0]):
            for b in range(0, self.zz.shape[1]):
                P = [self.xx[a, b], self.yy[a, b], zz[a, b]]
                list_p.append(P)

        self.df_new = pd.DataFrame(list_p, columns=['X', 'Y', 'Z'])
        self.save_method(file_name, dist=distance)

        rprint(f'Finish Radial Distance  = {distance}')

        return self.df_new

    def execute_method_multiprocessing(self,
                                       distance,
                                       file_name,
                                       type_coordinates,
                                       start,
                                       end,
                                       earth_radius=6371,
                                       show_prints=False):

        rprint(f'[green]Start Multiprocessing Interpolation with Radial Distance {distance} -->
{start} - {end} -- {self.zz.shape[0]}')

        list_results = []
        list_results_only = []
        for i in tqdm(range(start, end)):
            for j in range(0, self.zz.shape[1]):
                if show_prints:
                    print(i, j)
                list_res_dist = []
                list_dist = []

                if type_coordinates == 'GPS':
                    for a in range(0, len(self.df['X'])):

                        alfa = (np.cos(radians(90 - self.yy[i, j])) * np.cos(radians(90 -
self.df['Y'][a])) +
                                np.sin(radians(90 - self.yy[i, j])) * np.sin(radians(90 -
self.df['Y'][a])) *
                                np.cos(radians(self.xx[i, j] - self.df['X'][a])))
                        d = earth_radius * np.arccos(alfa)

                        if d <= distance:
```

```
                            b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a], self.xx[i, j],
self.yy[i, j], d]
                            list_dist.append(b)
                            list_res_dist.append(d)

                if type_coordinates == 'UTM':

                    for a in range(0, len(self.df['X'])):
                        d = math.dist((self.df['X'][a], self.df['Y'][a]), (self.xx[i, j], self.yy[i,
j]))
                        if d <= distance:
                            b = [self.df['X'][a], self.df['Y'][a], self.df['Z'][a], self.xx[i, j],
self.yy[i, j], d]
                            list_res_dist.append(d)
                            list_dist.append(b)

                list_dist = sorted(list_dist)

                if show_prints:
                    rprint(f'Radi = {distance}:', len(list_dist))
                    rprint(f'Head list distances: {list_dist[:5]}')

                df_dist = pd.DataFrame(list_dist, columns=['X', 'Y', 'Z', 'Mesh XX', 'Mesh YY',
'distance'])
                df_dist_by_dist = pd.DataFrame(df_dist.sort_values('distance').values,
                                               columns=['X', 'Y', 'Z', 'Mesh XX', 'Mesh YY',
'distance'])

                vect_w = []
                vect_z = []
                dist_0 = False

                for a in range(0, len(df_dist_by_dist)):
                    d = df_dist_by_dist['distance'][a]

                    if d == 0:
                        dist_0 = True
                        if show_prints:
                            rprint(f'[red]Radi = {distance}:', 'Diatance 0', '-->',
                                   self.xx[i, j], self.yy[i, j], df_dist_by_dist['Z'][a])
                        list_results.append((i, j, df_dist_by_dist['Z'][a], 'd = 0'))
                        list_results_only.append(df_dist_by_dist['Z'][a])
                    else:
                        if not dist_0:
                            w = 1 / (df_dist_by_dist['distance'][a] ** 2)
                            vect_w.append(w)
                            vect_z.append(df_dist_by_dist['Z'][a])

                if not dist_0:

                    if len(vect_w) == 0:
                        if show_prints:
                            rprint(f'[yellwo]Radi = {distance}:', 'nan')
                        list_results.append((i, j, 'nan', 'len(vect) = 0'))
                        list_results_only.append('nan')

                    else:
                        if show_prints:
                            rprint('Before Normalization: Sum W', sum(vect_w), type(sum(vect_w)))
                        vect_w = (vect_w / sum(vect_w))
                        if show_prints:
                            rprint('After Normalization: Sum W', sum(vect_w), type(sum(vect_w)))
                        vect_wz = np.array(vect_w) * np.array(vect_z)
                        if show_prints:
                            rprint(f'[blue]Radi = {distance}:', sum(vect_wz))
                        list_results.append((i, j, sum(vect_wz), 'calcule w'))
                        list_results_only.append(sum(vect_wz))


                if show_prints:
                    a = 1
                    print(a)

        rprint(f'[red]Finish Multiprocessing Interpolation with Radial Distance {distance} -->
{start} - {end} -- {self.zz.shape[0]}. {len(list_results)}')

        return list_results_only

    def save_method(self, name, dist):


self.df_new.to_csv(f'{self.point_folder}IDW_optim_method_{name}_Dist_{dist}_{self.num_points}_points.
csv',
                           index=False, header=False)

    def load_last_method(self, name, dist):

        self.df_new = pd.read_csv(
```

```
            f'{self.point_folder}IDW_optim_method_{name}_Dist_{dist}_{self.num_points}_points.csv',
            header=None)
        self.df_new.columns = ['X', 'Y', 'Z']

        return self.df_new

    def calculate_error(self, name_save, method='MSE', show_plot=False):

        def find_nearest(df_predict, value_x, value_y, value_z, min_z_validate):
            if value_z == 0:
                value_z = 0.00001

            df_predict_x_array = df_predict.X.to_numpy()
            df_predict_y_array = df_predict.Y.to_numpy()

            idx_x = (min(np.abs(df_predict_x_array - value_x)))
            idx_y = (min(np.abs(df_predict_y_array - value_y)))

            for x_val in df_predict_x_array:
                if np.abs(x_val - value_x) == idx_x:
                    x = x_val

            for y_val in df_predict_y_array:
                if np.abs(y_val - value_y) == idx_y:
                    y = y_val

            df_aprox = df_predict[(df_predict['X'] == x) & (df_predict['Y'] == y)]
            array_aprox = df_aprox.to_numpy()

            escal_z = abs(min_z_validate) * 1.2
            val_z_escal = (value_z + escal_z)
            int_z_escal = (array_aprox[0][2] + escal_z)
            e_a_z_escal = abs(val_z_escal - int_z_escal)
            e_r_z_escal = (e_a_z_escal / abs(val_z_escal)) * 100

            lst = [value_x, value_y, value_z,
                   array_aprox[0][0], array_aprox[0][1], array_aprox[0][2],
                   abs(value_x - array_aprox[0][0]), abs(value_y - array_aprox[0][1]), e_a_z_escal,
                   ((abs(value_x - array_aprox[0][0]) / abs(value_x)) * 100),
                   ((abs(value_y - array_aprox[0][1]) / abs(value_y)) * 100),
                   e_r_z_escal,
                   [value_z, array_aprox[0][2], (abs(value_z - array_aprox[0][2])),
                    ((abs(value_z - array_aprox[0][2]) / abs(value_z)) * 100)]]
            return lst

        comparative = []
        for i in tqdm(range(0, len(self.df_validation.values))):
            comparative.append(find_nearest(self.df_new, self.df_validation.X[i],
self.df_validation.Y[i],
                                            self.df_validation.Z[i],
min_z_validate=min(self.df_validation.Z)))
        df_comparative = pd.DataFrame(comparative,
                                      columns=['X_Real', 'Y_Real', 'Z_Real',
                                               'X_Interp', 'Y_Interp', 'Z_Interp',
                                               'X_Error_Absolute', 'Y_Error_Absolute',
'Z_Error_Absolute',
                                               'X_Error_Relative', 'Y_Error_Relative',
'Z_Error_Relative',
                                               'List_Real_Zs'])


df_comparative.to_csv(f'{self.point_folder}df_comparative_{name_save}_{self.num_points}_points.csv',
                      index=False)
        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Absolute'], cmap='plasma')
        plt.axis('scaled')
        plt.colorbar(sct)
        plt.title(f'{name_save} Absolute Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Absolute_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Relative'], cmap='plasma')
        plt.axis('scaled')
        col_bar = plt.colorbar(sct)
        col_bar.ax.get_yaxis().labelpad = 15
        col_bar.ax.set_ylabel('  %  ', rotation=270)
        plt.title(f'{name_save} Relative Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Relative_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        if method == 'MSE':
            ''' 1/n * sum((y - y')**2) '''
            e_abs_x = 0
```

```
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((e_abs_x / len(df_comparative)), (e_abs_y / len(df_comparative)),
                    (e_abs_z / len(df_comparative))), \
                   ((e_rel_x / len(df_comparative)), (e_rel_y / len(df_comparative)),
                    (e_rel_z / len(df_comparative))), df_comparative

        if method == 'RMSE':
            ''' sqrt(1/n * sum((y - y')**2)) '''
            e_abs_x = 0
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((math.sqrt(e_abs_x / len(df_comparative))), (math.sqrt(e_abs_y /
len(df_comparative))),
                    (math.sqrt(e_abs_z / len(df_comparative)))), \
                   ((math.sqrt(e_rel_x / len(df_comparative))), (math.sqrt(e_rel_y /
len(df_comparative))),
                    (math.sqrt(e_rel_z / len(df_comparative)))), df_comparative


if __name__ == "__main__":

    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    point_folder_dataset = f'./Points/'
    if not os.path.exists(point_folder_dataset):
        os.mkdir(point_folder_dataset)
    else:
        pass

    FILE_NAME = 'NAME_OF_DATASET'
    RES_FACT =
    SCALE_FACT = 1
    list_dist = [0.50, 1.00, 2.50, 5.00, 10.0, 15.0, 20.0]
    ERROR_METHOD = 'RMSE'
    DO_INTERPOLATE = True
    CALCULATE_ERROR = True
    SHOW_PLOTS = False

    df = pd.read_csv(f'{point_folder_dataset}{FILE_NAME}.csv', sep=';')
    rprint(df)

    plt.scatter(df['X'], df['Y'], c=df['Z'], cmap='plasma')
    plt.title('Input Data NAME_OF_DATASET')
    plt.ylabel('Latitude')
    plt.xlabel('Longitude')
    col_bar = plt.colorbar()
    col_bar.ax.set_ylabel('  Meters  ')
    plt.axis('scaled')
    plt.savefig(f'{point_folder}Plot_Input_Data.png')
    plt.show()

    num_of_points = int(len(df) / 1.10)
    df_subset = df.sample(n=num_of_points, random_state=1)
```

```
    df_drop = pd.DataFrame(df.drop(df_subset.index).values, columns=['X', 'Y', 'Z'])

    fit_IDW = IDW(df_subset, df_drop, resolution_factor=RES_FACT, reduction_scale=SCALE_FACT)

    # ddff = fit_IDW.execute_method_no_multiprocessing(distance=100, file_name=FILE_NAME,
type_coordinates='UTM',
    #                                                  show_prints=True)

    if DO_INTERPOLATE:
        dfs = []
        rprint(' ----------------------   Execute Method   ----------------------')

        for NUM_DIST in list_dist:
            rprint(f'Number of Subset points are {len(df_subset)}, '
                   f'and the number of Validation points are {len(df_drop)}')
            rprint()
            rprint('  -----   Variables   ----- ')
            rprint('File Name:              ', FILE_NAME)
            rprint('Resolution Factor:      ', RES_FACT)
            rprint('Factor Scale:           ', SCALE_FACT)
            rprint('Distance to the firsts: ', NUM_DIST)

            df = fit_IDW.execute_method(distance=NUM_DIST, file_name=FILE_NAME)
            dfs.append(df)

        rprint(dfs)

    if CALCULATE_ERROR:
        rprint(' ----------------------  Calculate Errors  ----------------------')
        dic_errors = {
            "IDW Distances": {}
        }

        for NUM_DIST in list_dist:
            rprint('Calculating Errors of Dist', NUM_DIST)

            ddff = fit_IDW.load_last_method(FILE_NAME, dist=NUM_DIST)

            plt.scatter(ddff['X'], ddff['Y'], c=ddff['Z'], cmap='plasma')
            plt.title(f'Output Data NAME_OF_DATASET Radial Distance {NUM_DIST}')
            plt.ylabel('Latitude')
            plt.xlabel('Longitude')
            col_bar = plt.colorbar()
            col_bar.ax.set_ylabel('  Meters  ')
            plt.axis('scaled')

plt.savefig(f'{point_folder}Plot_Output_Data_{NAME_OF_DATASET}_Radial_Distance_{NUM_DIST}.png')
                        if SHOW_PLOTS:
                                plt.show()
                        plt.close()

            abs_error, rel_error, df_error = fit_IDW.calculate_error(method=ERROR_METHOD,

name_save=f'IDW_Dist_{NUM_DIST}')
            print(f'Error Absolut XYZ {ERROR_METHOD} IDW Dist = {NUM_DIST}:       ', abs_error[0],
abs_error[1],
                  abs_error[2])
            print(f'Error Relatiu XYZ {ERROR_METHOD} IDW Dist = {NUM_DIST}:       ', rel_error[0],
rel_error[1],
                  rel_error[2])

            vec_norm_real = np.linalg.norm(df_error['Z_Real'])
            vec_norm_interpolate = np.linalg.norm(df_error['Z_Interp'])

            print('La Norma del Vector Z_Real:               ', round(vec_norm_real, 4))
            print('La Norma del Vector Z_Interp:             ', round(vec_norm_interpolate, 4))
            print('Diferencia de Normas Real - Interp:       ', round((vec_norm_real -
vec_norm_interpolate), 4))
            print('(Norma Real - Norma Interp)/(Norma Real):  ',
                  round(((vec_norm_real - vec_norm_interpolate)/vec_norm_real), 4))
            print('La Norma(Vector Z Reals - Vector Z Interp:   ',
                  round((np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 4))

            dic = {
                "Absolute Error": {
                    "X": np.round(abs_error[0], 7),
                    "Y": np.round(abs_error[1], 7),
                    "Z": np.round(abs_error[2], 7),
                },
                "Relative Error": {
                    "X": np.round(rel_error[0], 7),
                    "Y": np.round(rel_error[1], 7),
                    "Z": np.round(rel_error[2], 7)
                },
                "Norma Real": np.round(vec_norm_real, 7),
                "Norma Interp": np.round(vec_norm_interpolate, 7),
                "(Norma Real) - (Interp)": np.round((vec_norm_real - vec_norm_interpolate), 7),
                "(Norma Real - Norma Interp)/(Norma Real)":
```

```
        np.round(((vec_norm_real - vec_norm_interpolate) / vec_norm_real), 7),
    "Norma(Vector Reals - Vector Interp)":
        np.round((np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 7)
}

dic_to_print = {
    "Absolute Error": np.round(abs_error[2], 7),
    "Relative Error": np.round(rel_error[2], 7),
    "Norma(Vector Reals - Vector Interp)": np.round(
        (np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 7)
}

dic_errors['IDW Distances'][NUM_DIST] = dic_to_print

plt_z = ddff.pivot_table(index='X', columns='Y', values='Z').T.values
X_unique = np.sort(ddff.X.unique())
Y_unique = np.sort(ddff.Y.unique())
plt_x, plt_y = np.meshgrid(X_unique, Y_unique)

rprint(plt_x.shape)
rprint(plt_y.shape)
rprint(plt_z.shape)

if plt_x.shape == plt_y.shape == plt_z.shape:
    try:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
        cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
        ax.clabel(cs, fontsize=8)
        plt.axis('scaled')
        plt.colorbar(ct)
        plt.title(f'IDW Distances {NUM_DIST} Simulate')
        plt.savefig(f'{point_folder}Plot_IDW_Distances_{NUM_DIST}.png')
                            if SHOW_PLOTS:
                                    plt.show()
                            plt.close()
    except:
        rprint(f'Error in plot: IDW Distances {NUM_DIST} Simulate')

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
    cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
    ax.clabel(cs, fontsize=8)
    plt.scatter(df_drop['X'], df_drop['Y'], s=12, c='red')
    plt.axis('scaled')
    plt.colorbar(ct)
    plt.title(f'IDW Distances {NUM_DIST} Simulate vs Validate')
    plt.savefig(f'{point_folder}Plot_IDW_Distances_{NUM_DIST}_simulate_vs_validate.png')
                        if SHOW_PLOTS:
                                plt.show()
                        plt.close()

    # Fig SubPlot Errors
    fig, (ax1, ax2) = plt.subplots(1, 2)
    ct = ax1.contour(plt_x, plt_y, plt_z, colors='k')
    sct = ax1.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
                      c=df_error['Z_Error_Absolute'], cmap='plasma')
    ax1.clabel(ct, fontsize=8)
    ax1.title.set_text('Absolute Error')
    ax1.axis('scaled')
    divider1 = make_axes_locatable(ax1)
    cax1 = divider1.append_axes("right", size="5%", pad=0.1)
    col_bar = plt.colorbar(sct, ax=ax1, cax=cax1)
    col_bar.ax.get_yaxis().labelpad = 15
    col_bar.ax.set_ylabel('  meters  ')
    ct = ax2.contour(plt_x, plt_y, plt_z, colors='k')
    sct = ax2.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
                      c=df_error['Z_Error_Relative'], cmap='plasma')
    ax2.clabel(ct, fontsize=8)
    ax2.title.set_text('Relative Error')
    ax2.axis('scaled')
    divider2 = make_axes_locatable(ax2)
    cax2 = divider2.append_axes("right", size="5%", pad=0.1)
    col_bar = plt.colorbar(sct, ax=ax2, cax=cax2)
    col_bar.ax.get_yaxis().labelpad = 15
    fig.suptitle(f'Errors Distribution of IDW Distance = {NUM_DIST}', fontsize=16)
    plt.savefig(f'{point_folder}Plot_Errors_Distribution_IDW_Distance_{NUM_DIST}.png')
                        if SHOW_PLOTS:
                                plt.show()
                        plt.close()

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    X, Y, Z = axes3d.get_test_data(0.05)
    ct3d = ax.contour(plt_x, plt_y, plt_z, levels=[0])
    ax.clabel(ct3d, fontsize=9, inline=1)
```

```
        ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Interp'],
                   label='Interpolate Values')
        ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Real'],
                   label='Real Values')
        plt.legend(loc="best")
        plt.title(f'3D Compare Real - Interpolate IDW Distance = {NUM_DIST}')
        plt.savefig(f'{point_folder}Plot_Errors_3D_Compare_IDW_Distance_{NUM_DIST}.png')
                        if SHOW_PLOTS:
                                plt.show()
                        plt.close()

rprint(dic_errors)
```

9.2.2 RBF_multiprocessing.py

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from rich import print as rprint
import numpy as np
import os
from tqdm.auto import tqdm
import math
from concurrent import futures
from rich.progress import Progress
import multiprocessing as mp


def cm_to_inch(value):
    return value/2.54


plt.rcParams["figure.figsize"] = [cm_to_inch(40), cm_to_inch(20)]


class RBI:
    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    max_worker = mp.cpu_count() * 2

    @staticmethod
    def _multiprocess_index_handler(index, handler, args):
        """
        This function adds the index to the return of the handler function. Useful to sort the
results of a
        multi-threaded operation
        :param index: index to be returned
        :param handler: function handler to be called
        :param args: list with arguments of the function handler
        :return: tuple with (index, xxx) where xxx is whatever the handler function returned
        """
        result = handler(*args)  # call the handler
        return index, result  # add index to the result

    def multiprocess(self, arg_list, handler, max_workers=20, text: str = "progress..."):
        """
        Splits a repetitive task into several processes
        :param arg_list: each element in the list will crate a thread and its contents passed to the
handler
        :param handler: function to be invoked by every thread
        :param max_workers: Max processes to be launched at once
        :return: a list with the results (ordered as arg_list)
        :param text: text to be displayed in the progress bar
        """
        index = 0  # thread index
        ctx = mp.get_context('spawn')
        with futures.ProcessPoolExecutor(max_workers=max_workers, mp_context=ctx) as executor:
            processes = []  # empty thread list
            results = []  # empty list of thread results
            for args in arg_list:
                # submit tasks to the executor and append the tasks to the thread list
                processes.append(executor.submit(self._multiprocess_index_handler, index, handler,
args))
                index += 1

            with Progress() as progress:  # Use Progress() to show a nice progress bar
                task = progress.add_task(text, total=index)
                for future in futures.as_completed(processes):
                    future_result = future.result()  # result of the handler
                    results.append(future_result)
                    progress.update(task, advance=1)

            # sort the results by the index added by __threadify_index_handler
            sorted_results = sorted(results, key=lambda a: a[0])

            final_results = []  # create a new array without indexes
            for result in sorted_results:
                final_results.append(result[1])
            return final_results

    def __init__(self, data_frame, df_validation, resolution_factor=10, reduction_scale=1):
        self.df = pd.DataFrame(data_frame.values, columns=['X', 'Y', 'Z'])
        self.reduction_scale = reduction_scale

        self.df['X'] = self.df['X'] * self.reduction_scale
        self.df['Y'] = self.df['Y'] * self.reduction_scale
```

```
        self.num_points = len(self.df)
        self.resolution_factor = resolution_factor
        self.df_validation = df_validation
        self.df_validation['X'] = self.df_validation['X'] * self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] * self.reduction_scale

        x_min, x_max = int(np.round(min(self.df['X']))), int(np.round(max(self.df['X'])))
        y_min, y_max = int(np.round(min(self.df['Y']))), int(np.round(max(self.df['Y'])))

        num_resolution = ((x_max - x_min) * self.resolution_factor)
        x = np.linspace(x_min, x_max, num=int(num_resolution))
        y = np.linspace(y_min, y_max, num=int(num_resolution))

        for a in range(0, len(self.df_validation['X'])):
            x = np.append(x, self.df_validation['X'][a])
            y = np.append(y, self.df_validation['Y'][a])

        self.df['X'] = self.df['X'] / self.reduction_scale
        self.df['Y'] = self.df['Y'] / self.reduction_scale

        self.df_validation['X'] = self.df_validation['X'] / self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] / self.reduction_scale

        x = x / self.reduction_scale
        y = y / self.reduction_scale

        rprint(f'Min-Max X:  {round(min(x), 2)}  -  {round(max(x), 4)}')
        rprint(f'Min-Max Y:  {round(min(y), 2)}  -  {round(max(y), 4)}')
        rprint(f'Size X: {x.size}   -   Size Y: {y.size}')

        self.xx, self.yy = np.meshgrid(x, y)
        self.zz = np.empty(self.xx.shape)
        self.zz[:] = np.nan

        plt.scatter(self.xx, self.yy)
        plt.show()

    def k_ij(self, x_i, y_i, x_j, y_j, equation, beta):

        if equation == 'exp':
            return math.exp(-(beta * np.linalg.norm(math.dist((x_i, y_i), (x_j, y_j)))) ** 2)
        if equation == 'multi_sqrt':
            if math.dist((x_i, y_i), (x_j, y_j)) != np.linalg.norm(math.dist((x_i, y_i), (x_j,
y_j))):
                print(math.dist((x_i, y_i), (x_j, y_j)), '---', np.linalg.norm(math.dist((x_i, y_i),
(x_j, y_j))))
            return np.sqrt(1 + ((beta * math.dist((x_i, y_i), (x_j, y_j))) ** 2))

    def calculate_k_matrix(self, points, beta, kernel_function):
        k_matrix = []
        z_vector = []

        for i in range(0, len(points['X'])):
            k_vector = []
            z_vector.append([points['Z'][i]])
            t_sum = 0
            for j in range(0, len(points['X'])):

                k_vector.append(self.k_ij(points['X'][i], points['Y'][i], points['X'][j],
points['Y'][j],
                                          beta=beta, equation=kernel_function))

            k_matrix.append(k_vector)

        return np.matrix(k_matrix), np.matrix(z_vector), np.linalg.solve(np.matrix(k_matrix),
np.matrix(z_vector))

    def execute_method_no_multiprocessing(self, beta, kernel_function, file_name):
        rprint(f'Execute Radial Basis Function Interpolation with Beta  = {beta}')

        M_k, V_z, V_w = self.calculate_k_matrix(self.df, beta=beta, kernel_function=kernel_function)
        rprint(f'Process of RBF method with Beta = {beta} ...')
        for i in tqdm(range(0, self.zz.shape[0])):
            for j in range(0, self.zz.shape[1]):
                sum_wk = [V_w[a] * self.k_ij(self.xx[i, j], self.yy[i, j], self.df['X'][a],
self.df['Y'][a],
                                             beta=beta, equation=kernel_function)
                          for a in range(0, len(self.df['X']))]
                self.zz[i, j] = sum(sum_wk)

        rprint('Interpolation Finish.\nTransform Mesh into DataFrame')

        list_p = []
        for a in range(0, self.zz.shape[0]):
            for b in range(0, self.zz.shape[1]):
                P = [self.xx[a, b], self.yy[a, b], self.zz[a, b]]
                list_p.append(P)
```

```
        self.df_new = pd.DataFrame(list_p, columns=['X', 'Y', 'Z'])
        self.save_method(file_name, beta=beta)

        rprint(f'Finish Beta  = {beta}')

        return self.df_new

    def execute_method(self, beta, kernel_function, file_name, show_prints=False):
        rprint(f'Execute Radial Basis Function Interpolation with Beta  = {beta}')

        self.M_k, self.V_z, self.V_w = self.calculate_k_matrix(self.df, beta=beta,
kernel_function=kernel_function)

        argument_list = []
        list = np.arange(0, self.max_worker + 1)
        index = 1
        for i in list:
            if i != list[-1]:
                print(i)
                inici = int(self.zz.shape[0] / self.max_worker) * (index - 1)
                final = int(self.zz.shape[0] / self.max_worker) * index
                print(inici, final, '\n')
                index += 1
            else:
                print(i)
                inici = int(self.zz.shape[0] / self.max_worker) * self.max_worker
                final = self.zz.shape[0]
                print(inici, final, '\n')
            myargs = [beta, kernel_function, file_name, inici, final]
            argument_list.append(myargs)

        if show_prints:
            rprint(argument_list)

        rprint(f'Multiprocessing of RBF method with Beta = {beta} ...')

        results = self.multiprocess(argument_list, self.execute_method_multiprocessing,
max_workers=self.max_worker)

        if show_prints:
            rprint(results)

        list_res = []
        for result in results:
            list_res += result

        if show_prints:
            rprint(list_res, len(list_res))

        zz = np.array(list_res).reshape(self.zz.shape[1], self.zz.shape[0])

        rprint('Interpolation Finish.\nTransform Mesh into DataFrame')

        list_p = []
        for a in range(0, self.zz.shape[0]):
            for b in range(0, self.zz.shape[1]):
                P = [self.xx[a, b], self.yy[a, b], zz[a, b]]
                list_p.append(P)

        self.df_new = pd.DataFrame(list_p, columns=['X', 'Y', 'Z'])
        self.save_method(file_name, beta=beta)

        rprint(f'Finish Beta  = {beta}')

        return self.df_new

    def execute_method_multiprocessing(self, beta, kernel_function, file_name, start, end,
show_prints=False):

        rprint(f'[green]Start Multiprocessing Interpolation with Beta {beta} --> {start} - {end} --
{self.zz.shape[0]}')

        list_results = []
        for i in tqdm(range(start, end)):
            for j in range(0, self.zz.shape[1]):
                if show_prints:
                    print(i, j)
                sum_wk = [self.V_w[a] * self.k_ij(self.xx[i, j], self.yy[i, j], self.df['X'][a],
self.df['Y'][a],
                                                  beta=beta, equation=kernel_function)
                          for a in range(0, len(self.df['X']))]
                list_results.append(sum(sum_wk))

        rprint(f'[yellow]Finish Multiprocessing Interpolation with Beta {beta} --> {start} - {end} --
{self.zz.shape[0]}')

        return list_results
```

```
    def save_method(self, name, beta):


self.df_new.to_csv(f'{self.point_folder}RBI_optim_method_{name}_Beta_{beta}_{self.num_points}_points.
csv',
                           index=False, header=False)

    def load_last_method(self, name, beta):

        self.df_new = pd.read_csv(
            f'{self.point_folder}RBI_optim_method_{name}_Beta_{beta}_{self.num_points}_points.csv',
            header=None)
        self.df_new.columns = ['X', 'Y', 'Z']

        return self.df_new

    def calculate_error(self, name_save, method='MSE', show_plot=False):

        def find_nearest(df_predict, value_x, value_y, value_z, min_z_validate):
            if value_z == 0:
                value_z = 0.00001

            df_predict_x_array = df_predict.X.to_numpy()
            df_predict_y_array = df_predict.Y.to_numpy()

            idx_x = (min(np.abs(df_predict_x_array - value_x)))
            idx_y = (min(np.abs(df_predict_y_array - value_y)))

            for x_val in df_predict_x_array:
                if np.abs(x_val - value_x) == idx_x:
                    x = x_val

            for y_val in df_predict_y_array:
                if np.abs(y_val - value_y) == idx_y:
                    y = y_val

            df_aprox = df_predict[(df_predict['X'] == x) & (df_predict['Y'] == y)]
            array_aprox = df_aprox.to_numpy()

            escal_z = abs(min_z_validate) * 1.2
            val_z_escal = (value_z + escal_z)
            int_z_escal = (array_aprox[0][2] + escal_z)
            e_a_z_escal = abs(val_z_escal - int_z_escal)
            e_r_z_escal = (e_a_z_escal / abs(val_z_escal)) * 100

            lst = [value_x, value_y, value_z,
                   array_aprox[0][0], array_aprox[0][1], array_aprox[0][2],
                   abs(value_x - array_aprox[0][0]), abs(value_y - array_aprox[0][1]), e_a_z_escal,
                   ((abs(value_x - array_aprox[0][0])/abs(value_x))*100),
                   ((abs(value_y - array_aprox[0][1])/abs(value_y))*100),
                   e_r_z_escal,
                   [value_z, array_aprox[0][2], (abs(value_z - array_aprox[0][2])),
                    ((abs(value_z - array_aprox[0][2])/abs(value_z))*100)]]
            return lst

        comparative = []
        for i in tqdm(range(0, len(self.df_validation.values))):
            comparative.append(find_nearest(self.df_new, self.df_validation.X[i],
self.df_validation.Y[i],
                                            self.df_validation.Z[i],
min_z_validate=min(self.df_validation.Z)))

        df_comparative = pd.DataFrame(comparative,
                                      columns=['X_Real', 'Y_Real', 'Z_Real',
                                               'X_Interp', 'Y_Interp', 'Z_Interp',
                                               'X_Error_Absolute', 'Y_Error_Absolute',
'Z_Error_Absolute',
                                               'X_Error_Relative', 'Y_Error_Relative',
'Z_Error_Relative',
                                               'List_Real_Zs'])

df_comparative.to_csv(f'{self.point_folder}df_comparative_{name_save}_{self.num_points}_points.csv',
                              index=False)

        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Absolute'], cmap='plasma')
        plt.axis('scaled')
        plt.colorbar(sct)
        plt.title(f'{name_save} Absolute Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Absolute_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Relative'], cmap='plasma')
        plt.axis('scaled')
```

```
        col_bar = plt.colorbar(sct)
        col_bar.ax.get_yaxis().labelpad = 15
        col_bar.ax.set_ylabel('  %  ', rotation=270)
        plt.title(f'{name_save} Relative Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Relative_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        if method == 'MSE':
            ''' 1/n * sum((y - y')**2) '''
            e_abs_x = 0
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((e_abs_x / len(df_comparative)), (e_abs_y / len(df_comparative)), (e_abs_z /
len(df_comparative))),\
                   ((e_rel_x / len(df_comparative)), (e_rel_y / len(df_comparative)),
                    (e_rel_z / len(df_comparative))), df_comparative

        if method == 'RMSE':
            ''' sqrt(1/n * sum((y - y')**2)) '''
            e_abs_x = 0
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((math.sqrt(e_abs_x / len(df_comparative))), (math.sqrt(e_abs_y /
len(df_comparative))),
                    (math.sqrt(e_abs_z / len(df_comparative)))), \
                   ((math.sqrt(e_rel_x / len(df_comparative))), (math.sqrt(e_rel_y /
len(df_comparative))),
                    (math.sqrt(e_rel_z / len(df_comparative)))), df_comparative


if __name__ == "__main__":

    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    point_folder_dataset = f'./Points/'
    if not os.path.exists(point_folder_dataset):
        os.mkdir(point_folder_dataset)
    else:
        pass

    FILE_NAME = 'NAME_OF_DATASET'
    RES_FACT = 1
    SCALE_FACT = 1
    list_beta = [0.10, 0.25, 0.50, 0.75, 1.00, 2.50, 5.00, 10.0, 25.0, 50.0, 75.0]
    ERROR_METHOD = 'RMSE'
    KERNEL_FUNC = 'multi_sqrt'
    DO_INTERPOLATE = True
    CALCULATE_ERROR = True
    SHOW_PLOTS = False

    df = pd.read_csv(f'{point_folder_dataset}{FILE_NAME}.csv', sep=';')
    rprint(df)
```

```
    plt.scatter(df['X'], df['Y'], c=df['Z'], cmap='plasma')
    plt.title('Input Data NAME_OF_DATASET')
    plt.ylabel('Latitude')
    plt.xlabel('Longitude')
    col_bar = plt.colorbar()
    col_bar.ax.set_ylabel('  Meters  ')
    plt.axis('scaled')
    plt.savefig(f'{point_folder}Plot_Input_Data.png')
    if SHOW_PLOTS:
        plt.show()
    plt.close()

    num_of_points = int(len(df) / 1.10)
    df_subset = df.sample(n=num_of_points, random_state=1)
    df_drop = pd.DataFrame(df.drop(df_subset.index).values, columns=['X', 'Y', 'Z'])

    fit_RBI = RBI(df_subset, df_drop, resolution_factor=RES_FACT, reduction_scale=SCALE_FACT)

    # ddff = fit_RBI.execute_method(beta=BETA, file_name=FILE_NAME)

    if DO_INTERPOLATE:
        dfs = []
        rprint(' ----------------------   Execute Method   ----------------------')

        for BETA in list_beta:
            rprint(f'Number of Subset points are {len(df_subset)}, '
                   f'and the number of Validation points are {len(df_drop)}')
            rprint()
            rprint('  -----   Variables   ----- ')
            rprint('File Name:              ', FILE_NAME)
            rprint('Resolution Factor:      ', RES_FACT)
            rprint('Factor Scale:           ', SCALE_FACT)
            rprint('Constant Beta:          ', BETA)

            df = fit_RBI.execute_method(beta=BETA, kernel_function=KERNEL_FUNC, file_name=FILE_NAME)
            dfs.append(df)

        rprint(dfs)

    if CALCULATE_ERROR:
        rprint(' ----------------------  Calculate Errors  ----------------------')
        dic_errors = {
            "RBF Beta": {}
        }

        for BETA in list_beta:
            rprint('Calculating Errors of Beta', BETA)

            ddff = fit_RBI.load_last_method(FILE_NAME, beta=BETA)

            plt.scatter(ddff['X'], ddff['Y'], c=ddff['Z'], cmap='plasma')
            plt.title(f'Output Data NAME_OF_DATASET Beta {BETA}')
            plt.ylabel('Latitude')
            plt.xlabel('Longitude')
            col_bar = plt.colorbar()
            col_bar.ax.set_ylabel('  Meters  ')
            plt.axis('scaled')
            plt.savefig(f'{point_folder}Plot_Output_Data_{NAME_OF_DATASET}_Beta_{BETA}.png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

            abs_error, rel_error, df_error = fit_RBI.calculate_error(method=ERROR_METHOD,
name_save=f'RBF_Beta_{BETA}')
            print(f'Error Absolut XYZ {ERROR_METHOD} RBF Beta = {BETA}:       ', abs_error[0],
abs_error[1],
                  abs_error[2])
            print(f'Error Relatiu XYZ {ERROR_METHOD} RBF Beta = {BETA}:       ', rel_error[0],
rel_error[1],
                  rel_error[2])

            vec_norm_real = np.linalg.norm(df_error['Z_Real'])
            vec_norm_interpolate = np.linalg.norm(df_error['Z_Interp'])

            print('La Norma del Vector Z_Real:                ', round(vec_norm_real, 4))
            print('La Norma del Vector Z_Interp:              ', round(vec_norm_interpolate, 4))
            print('Diferencia de Normas Real - Interp:        ', round((vec_norm_real -
vec_norm_interpolate), 4))
            print('(Norma Real - Norma Interp)/(Norma Real):  ',
                  round(((vec_norm_real - vec_norm_interpolate)/vec_norm_real), 4))

            dic = {
                "Absolute Error": {
                    "X": np.round(abs_error[0], 7),
                    "Y": np.round(abs_error[1], 7),
                    "Z": np.round(abs_error[2], 7),
                },
```

```
    "Relative Error": {
        "X": np.round(rel_error[0], 7),
        "Y": np.round(rel_error[1], 7),
        "Z": np.round(rel_error[2], 7)
    },
    "Norma Real": np.round(vec_norm_real, 7),
    "Norma Interp": np.round(vec_norm_interpolate, 7),
    "(Norma Real) - (Interp)": np.round((vec_norm_real - vec_norm_interpolate), 7),
    "(Norma Real - Norma Interp)/(Norma Real)": np.round(
        ((vec_norm_real - vec_norm_interpolate) / vec_norm_real), 7),
    "Norma(Vector Reals - Vector Interp)": np.round(
        (np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 7)
}

dic_to_print = {
    "Absolute Error": np.round(abs_error[2], 7),
    "Relative Error": np.round(rel_error[2], 7),
    "Norma(Vector Reals - Vector Interp)": np.round(
        (np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 7)
}

# rprint(dic)
dic_errors["RBF Beta"][BETA] = dic_to_print
# rprint(dic_errors)

plt_z = ddff.pivot_table(index='X', columns='Y', values='Z').T.values
X_unique = np.sort(ddff.X.unique())
Y_unique = np.sort(ddff.Y.unique())
plt_x, plt_y = np.meshgrid(X_unique, Y_unique)

fig = plt.figure()
ax = fig.add_subplot(111)
ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
ax.clabel(cs, fontsize=8)
plt.axis('scaled')
plt.colorbar(ct)
plt.title(f'RBF Beta {BETA} Simulate')
plt.savefig(f'{point_folder}Plot_RBF_Beta_{BETA}.png')
if SHOW_PLOTS:
    plt.show()
plt.close()

fig = plt.figure()
ax = fig.add_subplot(111)
ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
ax.clabel(cs, fontsize=8)
plt.scatter(df_drop['X'], df_drop['Y'], s=12, c='red')
plt.axis('scaled')
plt.colorbar(ct)
plt.title(f'RBF Beta {BETA} Simulate vs Validate')
plt.savefig(f'{point_folder}Plot_RBF_Beta_{BETA}_simulate_vs_validate.png')
if SHOW_PLOTS:
    plt.show()
plt.close()

# Fig SubPlot Errors
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(cm_to_inch(40), cm_to_inch(20)))
ct = ax1.contour(plt_x, plt_y, plt_z, colors='k')
sct = ax1.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
                  c=df_error['Z_Error_Absolute'], cmap='plasma')  # 'jet') # 'YlOrRd')
ax1.clabel(ct, fontsize=8)
ax1.title.set_text('Absolute Error')
ax1.axis('scaled')
divider1 = make_axes_locatable(ax1)
cax1 = divider1.append_axes("right", size="5%", pad=0.1)
col_bar = plt.colorbar(sct, ax=ax1, cax=cax1)  # shrink=0.8)
col_bar.ax.get_yaxis().labelpad = 15
col_bar.ax.set_ylabel('  meters  ')
ct = ax2.contour(plt_x, plt_y, plt_z, colors='k')
sct = ax2.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
                  c=df_error['Z_Error_Relative'], cmap='plasma')  # 'jet') # 'YlOrRd')
ax2.clabel(ct, fontsize=8)
ax2.title.set_text('Relative Error')
ax2.axis('scaled')
divider2 = make_axes_locatable(ax2)
cax2 = divider2.append_axes("right", size="5%", pad=0.1)
col_bar = plt.colorbar(sct, ax=ax2, cax=cax2)  # shrink=0.8)
col_bar.ax.get_yaxis().labelpad = 15
col_bar.ax.set_ylabel('  %  ', rotation=270)
fig.suptitle(f'Errors Distribution of RBF Beta = {BETA}', fontsize=16)
plt.savefig(f'{point_folder}Plot_Errors_Distribution_RBF_Beta_{BETA}.png')
if SHOW_PLOTS:
    plt.show()
plt.close()

fig = plt.figure()
```

```
    ax = fig.add_subplot(111, projection='3d')
    X, Y, Z = axes3d.get_test_data(0.05)
    ct3d = ax.contour(plt_x, plt_y, plt_z, levels=[0])
    ax.clabel(ct3d, fontsize=9, inline=1)
    ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Interp'],
               label='Interpolate Values')
    ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Real'],
               label='Real Values')
    plt.legend(loc="best")
    plt.title(f'3D Compare Real - Interpolate RBF Beta = {BETA}')
    plt.savefig(f'{point_folder}Plot_Errors_3D_Compare_RBF_Beta_{BETA}.png')
    if SHOW_PLOTS:
        plt.show()
    plt.close()

rprint(dic_errors)
```

9.2.3 Kriging_multiprocessing.py

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from rich import print as rprint
import numpy as np
import skgstat as skg
import os
from tqdm.auto import tqdm
import math


def cm_to_inch(value):
    return value/2.54


plt.rcParams["figure.figsize"] = [cm_to_inch(40), cm_to_inch(20)]


class Kriging:
    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    def __init__(self, data_frame, df_validation, resolution_factor=50, reduction_scale=1):
        self.points = data_frame
        self.reduction_scale = reduction_scale
        self.df_validation = df_validation

        self.num_points = len(self.points)
        self.resolution_factor = resolution_factor

        self.df_validation.to_csv(f'{self.point_folder}df_validation_{self.num_points}_points.csv',
                                  index=False, header=False)
        self.points.to_csv(f'{self.point_folder}df_points_{self.num_points}_points.csv',
                           index=False, header=False)

        self.V_df = skg.Variogram(self.points[['X', 'Y']].values,
                                  self.points.Z.values, maxlag='median', normalize=False)

    def semivariogram_plot(self, show_plot=True):

        fig, (ax_1, ax_2, ax_3) = plt.subplots(1, 3, figsize=(15, 5), sharey=True, sharex=True)
        x = np.linspace(0, self.V_df.maxlag, 100)
        manual_lags = (6, 12, 18)
        # plot each variogram

        self.V_df.bin_func = 'even'
        self.V_df.n_lags = 10
        ax_1.plot(self.V_df.bins, self.V_df.experimental, '.b')
        ax_1.grid(which='major', axis='x')
        ax_1.set_title('10 lags')

        self.V_df.n_lags = 20
        ax_2.plot(self.V_df.bins, self.V_df.experimental, '.b')
        ax_2.grid(which='major', axis='x')
        ax_2.set_title('20 lags')

        self.V_df.bin_func = 'scott'
        ax_3.set_xlabel('Lag (-)')
        ax_3.plot(self.V_df.bins, self.V_df.experimental, '.b')
        ax_3.grid(which='major', axis='x')
        ax_3.set_title('Scott rule lags')

        plt.tight_layout()
        plt.savefig(f'{self.point_folder}Plot_Kriging_SemiVariogramas_1.png')
        if show_plot:
            plt.show()
        plt.close()

    def semivariogram_plot_errors(self, show_plot=True):

        self.V_df.bin_func = 'scott'
        fig, _a = plt.subplots(2, 3, sharex=False, sharey=False)
        axes = _a.flatten()
        for i, model in enumerate(('spherical', 'exponential', 'gaussian', 'matern', 'stable',
'cubic')):
            self.V_df.model = model
            self.V_df.plot(axes=axes[i], hist=False, show=False, grid=False)
            axes[i].set_title('Model: %s; RMSE: %.2f' % (model, self.V_df.rmse))
            axes[i].set_ylim(0, max(self.V_df.experimental))
        plt.savefig(f'{self.point_folder}Plot_Kriging_SemiVariogramas.png')
        if show_plot:
            plt.show()
```

```
        plt.close()

    def interpolate_df(self, V, ax, df):
        self.points['X'] = self.points['X'] * self.reduction_scale
        self.points['Y'] = self.points['Y'] * self.reduction_scale

        self.df_validation['X'] = self.df_validation['X'] * self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] * self.reduction_scale

        x_min, x_max = int(np.round(min(df['X']))), int(np.round(max(df['X'])))
        y_min, y_max = int(np.round(min(df['Y']))), int(np.round(max(df['Y'])))

        num_resolution = ((x_max - x_min) * self.resolution_factor)
        x = np.linspace(x_min, x_max, num=int(num_resolution))
        y = np.linspace(y_min, y_max, num=int(num_resolution))

        for a in range(0, len(self.df_validation['X'])):

            x = np.append(x, self.df_validation['X'][a])
            y = np.append(y, self.df_validation['Y'][a])

        self.points['X'] = self.points['X'] / self.reduction_scale
        self.points['Y'] = self.points['Y'] / self.reduction_scale

        self.df_validation['X'] = self.df_validation['X'] / self.reduction_scale
        self.df_validation['Y'] = self.df_validation['Y'] / self.reduction_scale

        x = x / self.reduction_scale
        y = y / self.reduction_scale

        rprint(f'Min-Max X:  {round(min(x), 2)}  -  {round(max(x), 4)}')
        rprint(f'Min-Max Y:  {round(min(y), 2)}  -  {round(max(y), 4)}')
        rprint(f'Size X: {x.size}   -   Size Y: {y.size}')

        xx, yy = np.meshgrid(x, y)
        zz = np.empty(xx.shape)
        zz[:] = np.nan

        ok = skg.OrdinaryKriging(V, min_points=3, max_points=15, mode='exact')
        zz = ok.transform(xx.flatten(), yy.flatten()).reshape(xx.shape)

        art = ax.matshow(zz, origin='lower', cmap='plasma', vmin=V.values.min(), vmax=V.values.max())
        ax.set_title('%s model' % V.model.__name__)
        plt.colorbar(art, ax=ax)
        # plt.show()

        return xx, yy, zz

    def execute_method(self):

        self.fields = []
        fig, _a = plt.subplots(2, 3, figsize=(12, 10), sharex=True, sharey=True)
        axes = _a.flatten()
        for i, model in tqdm(enumerate(('spherical', 'exponential', 'gaussian', 'matern', 'stable',
'cubic'))):
            self.V_df.model = model
            self.xx, self.yy, zz = self.interpolate_df(self.V_df, axes[i], self.points)
            self.fields.append(zz)

        plt.show()

        self.df_fields = pd.DataFrame(
            {'spherical': self.fields[0].flatten(), 'exponential': self.fields[1].flatten(),
             'gaussian': self.fields[2].flatten(), 'matern': self.fields[3].flatten(),
             'stable': self.fields[4].flatten(), 'cubic': self.fields[5].flatten()}).describe()

        rprint(self.df_fields)

        self.df_fields.to_csv(f'{self.point_folder}df_fields_{self.num_points}_points.csv',
                              index=True, header=True)

        return self.df_fields

    def save_method(self, name):

        for i, model in enumerate(('spherical', 'exponential', 'gaussian', 'matern', 'stable',
'cubic')):
            zz = self.fields[i]

            list_p = []
            for a in range(0, zz.shape[0]):
                for b in range(0, zz.shape[1]):
                    P = [self.xx[a, b], self.yy[a, b], zz[a, b]]
                    list_p.append(P)
            df_save = pd.DataFrame(list_p)
            df_save.columns = ['X', 'Y', 'Z']
            df_save.to_csv(f'{self.point_folder}kriging_{model}_{name}_{self.num_points}_points.csv',
                           index=False, header=False)
```

```
    def load_last_method(self, model, name):

        df_load =
pd.read_csv(f'{self.point_folder}kriging_{model}_{name}_{self.num_points}_points.csv', header=None)
        df_load.columns = ['X', 'Y', 'Z']

        return df_load

    def calculate_error(self, df_model, name_save, method='MSE', show_plot=False):

        def find_nearest(df_predict, value_x, value_y, value_z, min_z_validate):
            global x, y
            df_predict_x_array = df_predict.X.to_numpy()
            df_predict_y_array = df_predict.Y.to_numpy()

            idx_x = (min(np.abs(df_predict_x_array - value_x)))
            idx_y = (min(np.abs(df_predict_y_array - value_y)))

            for i in df_predict_x_array:
                if np.abs(i - value_x) == idx_x:
                    x = i

            for i in df_predict_y_array:
                if np.abs(i - value_y) == idx_y:
                    y = i

            df_aprox = df_predict[(df_predict['X'] == x) & (df_predict['Y'] == y)]
            array_aprox = df_aprox.to_numpy()

            escal_z = abs(min_z_validate)*1.2
            val_z_escal = (value_z + escal_z)
            int_z_escal = (array_aprox[0][2] + escal_z)
            e_a_z_escal = abs(val_z_escal - int_z_escal)
            e_r_z_escal = (e_a_z_escal/abs(val_z_escal))*100

            lst = [value_x, value_y, value_z,
                   array_aprox[0][0], array_aprox[0][1], array_aprox[0][2],
                   abs(value_x - array_aprox[0][0]), abs(value_y - array_aprox[0][1]), e_a_z_escal,
                   ((abs(value_x - array_aprox[0][0])/abs(value_x))*100),
                   ((abs(value_y - array_aprox[0][1])/abs(value_y))*100),
                   e_r_z_escal,
                   [value_z, array_aprox[0][2], (abs(value_z - array_aprox[0][2])),
                    ((abs(value_z - array_aprox[0][2])/abs(value_z))*100)]]
            return lst
        comparative = []
        for i in tqdm(range(0, len(self.df_validation.values))):
            comparative.append(find_nearest(df_model, self.df_validation.X[i],
self.df_validation.Y[i],
                                            self.df_validation.Z[i],
min_z_validate=min(self.df_validation.Z)))

        df_comparative = pd.DataFrame(comparative,
                                      columns=['X_Real', 'Y_Real', 'Z_Real',
                                               'X_Interp', 'Y_Interp', 'Z_Interp',
                                               'X_Error_Absolute', 'Y_Error_Absolute',
'Z_Error_Absolute',
                                               'X_Error_Relative', 'Y_Error_Relative',
'Z_Error_Relative',
                                               'List_Real_Zs'])

df_comparative.to_csv(f'{self.point_folder}df_comparative_{name_save}_{self.num_points}_points.csv',
                              index=False)

        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Absolute'], cmap='plasma')
        plt.axis('scaled')
        col_bar = plt.colorbar(sct)
        plt.title(f'{name_save} Absolute Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Absolute_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        sct = plt.scatter(df_comparative['X_Real'], df_comparative['Y_Real'], s=12,
                          c=df_comparative['Z_Error_Relative'], cmap='plasma')
        plt.axis('scaled')
        col_bar = plt.colorbar(sct)
        col_bar.ax.get_yaxis().labelpad = 15
        col_bar.ax.set_ylabel('  %  ', rotation=270)
        plt.title(f'{name_save} Relative Error')
        plt.savefig(f'{self.point_folder}Plot_Error_Relative_{method}_{name_save}.png')
        if show_plot:
            plt.show()
        plt.close()

        if method == 'MSE':
            ''' 1/n * sum((y - y')**2) '''
```

```
            e_abs_x = 0
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((e_abs_x / len(df_comparative)), (e_abs_y / len(df_comparative)), (e_abs_z /
len(df_comparative))), \
                   ((e_rel_x / len(df_comparative)), (e_rel_y / len(df_comparative)),
                    (e_rel_z / len(df_comparative))), df_comparative

        if method == 'RMSE':
            ''' sqrt(1/n * sum((y - y')**2)) '''
            e_abs_x = 0
            e_abs_y = 0
            e_abs_z = 0

            e_rel_x = 0
            e_rel_y = 0
            e_rel_z = 0

            for i in tqdm(range(0, len(df_comparative))):
                # print(df_comparative.X_Dist[i], df_comparative.Y_Dist[i], df_comparative.Z_Dist[i])
                e_abs_x += df_comparative.X_Error_Absolute[i] ** 2
                e_abs_y += df_comparative.Y_Error_Absolute[i] ** 2
                e_abs_z += df_comparative.Z_Error_Absolute[i] ** 2

                e_rel_x += df_comparative.X_Error_Relative[i] ** 2
                e_rel_y += df_comparative.Y_Error_Relative[i] ** 2
                e_rel_z += df_comparative.Z_Error_Relative[i] ** 2

            return ((math.sqrt(e_abs_x / len(df_comparative))), (math.sqrt(e_abs_y /
len(df_comparative))),
                    (math.sqrt(e_abs_z / len(df_comparative)))), \
                   ((math.sqrt(e_rel_x / len(df_comparative))), (math.sqrt(e_rel_y /
len(df_comparative))),
                    (math.sqrt(e_rel_z / len(df_comparative)))), \
                   df_comparative


if __name__ == "__main__":

    point_folder = f'./Points/Subset_NAME_OF_DATASET/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    point_folder_dataset = f'./Points/'
    if not os.path.exists(point_folder_dataset):
        os.mkdir(point_folder_dataset)
    else:
        pass

    FILE_NAME = 'NAME_OF_DATASET'
    RES_FACT = 1
    SCALE_FACT = 1
    ERROR_METHOD = 'RMSE'
    DO_INTERPOLATE = True
    CALCULATE_ERROR = True
    SHOW_PLOTS = False

    df = pd.read_csv(f'{point_folder_dataset}{FILE_NAME}.csv', sep=';')
    print(len(df))

    plt.scatter(df['X'], df['Y'], c=df['Z'], cmap='plasma')
    plt.title('Input Data NAME_OF_DATASET')
    plt.ylabel('Latitude')
    plt.xlabel('Longitude')
    col_bar = plt.colorbar()
    col_bar.ax.set_ylabel('  Meters  ')
    plt.axis('scaled')
    plt.savefig(f'{point_folder}Plot_Input_Data.png')
    if SHOW_PLOTS:
        plt.show()
    plt.close()
```

```
    num_of_points = int(len(df) / 1.10)
    df_subset = df.sample(n=num_of_points, random_state=1)
    df_drop = pd.DataFrame(df.drop(df_subset.index).values, columns=['X', 'Y', 'Z'])

    krig_fit = Kriging(df_subset, df_drop, resolution_factor=RES_FACT, reduction_scale=SCALE_FACT)
    krig_fit.semivariogram_plot()
    krig_fit.semivariogram_plot_errors()

    if DO_INTERPOLATE:
        dfs = []
        rprint(' ----------------------   Execute Method   ----------------------')

        rprint(f'Number of Subset points are {len(df_subset)}, '
               f'and the number of Validation points are {len(df_drop)}')

        rprint('  -----   Variables   -----  ')
        rprint('File Name:              ', FILE_NAME)
        rprint('Resolution Factor:      ', RES_FACT)
        rprint('Factor Scale:           ', SCALE_FACT)

        dfs = krig_fit.execute_method()
        krig_fit.save_method(FILE_NAME)

    if CALCULATE_ERROR:
        rprint(' ----------------------  Calculate Errors  ----------------------')
        dic_errors = {
            "Kriging Method": {}
        }

        for i, CONCRETE_MODEL in enumerate(('spherical', 'exponential', 'gaussian', 'matern',
'stable', 'cubic')):
            print(i, CONCRETE_MODEL.capitalize())
            rprint(f'Open {CONCRETE_MODEL.capitalize()} Model')
            ddff = krig_fit.load_last_method(CONCRETE_MODEL, FILE_NAME)

            plt.scatter(ddff['X'], ddff['Y'], c=ddff['Z'], cmap='plasma')
            plt.title(f'Output Data NAME_OF_DATASET Kriging {CONCRETE_MODEL.capitalize()}')
            plt.ylabel('Latitude')
            plt.xlabel('Longitude')
            col_bar = plt.colorbar()
            col_bar.ax.set_ylabel('  Meters  ')
            plt.axis('scaled')

plt.savefig(f'{point_folder}Plot_Output_Data_{NAME_OF_DATASET}_Kriging_{CONCRETE_MODEL.capitalize()}.
png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

            abs_error, rel_error, df_error = krig_fit.calculate_error(ddff, method=ERROR_METHOD,

name_save=f'Kriging_{CONCRETE_MODEL.capitalize()}')
            print(f'Error Absolut XYZ {ERROR_METHOD} Kriging {CONCRETE_MODEL.capitalize()}:       ',
abs_error[0], abs_error[1], abs_error[2])
            print(f'Error Relatiu XYZ {ERROR_METHOD} Kriging {CONCRETE_MODEL.capitalize()}:       ',
rel_error[0], rel_error[1], rel_error[2])

            vec_norm_real = np.linalg.norm(df_error['Z_Real'])
            vec_norm_interpolate = np.linalg.norm(df_error['Z_Interp'])

            print('La Norma del Vector Z_Real:                  ', round(vec_norm_real, 4))
            print('La Norma del Vector Z_Interp:                ', round(vec_norm_interpolate, 4))
            print('Diferencia de Normas Real - Interp:          ', round((vec_norm_real -
vec_norm_interpolate), 4))
            print('(Norma Real - Norma Interp)/(Norma Real):    ',
                  round(((vec_norm_real - vec_norm_interpolate) / vec_norm_real), 4))
            print('La Norma(Vector Z Reals - Vector Z Interp:   ',
                  round((np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 4))

            dic = {
                "Absolute Error": {
                    "X": np.round(abs_error[0], 7),
                    "Y": np.round(abs_error[1], 7),
                    "Z": np.round(abs_error[2], 7),
                },
                "Relative Error": {
                    "X": np.round(rel_error[0], 7),
                    "Y": np.round(rel_error[1], 7),
                    "Z": np.round(rel_error[2], 7)
                },
                "Norma Real": np.round(vec_norm_real, 7),
                "Norma Interp": np.round(vec_norm_interpolate, 7),
                "(Norma Real) - (Interp)": np.round((vec_norm_real - vec_norm_interpolate), 7),
                "(Norma Real - Norma Interp)/(Norma Real)": np.round(((vec_norm_real -
vec_norm_interpolate) / vec_norm_real), 7),
                "Norma(Vector Reals - Vector Interp)": np.round((np.linalg.norm((df_error['Z_Real'] -
df_error['Z_Interp']))), 7)
```

```
            }

            dic_to_print = {
                "Absolute Error": np.round(abs_error[2], 7),
                "Relative Error": np.round(rel_error[2], 7),
                "Norma(Vector Reals - Vector Interp)": np.round(
                    (np.linalg.norm((df_error['Z_Real'] - df_error['Z_Interp']))), 7)
            }

            # rprint(dic)
            dic_errors['Kriging Method'][CONCRETE_MODEL.capitalize()] = dic_to_print

            plt_z = ddff.pivot_table(index='X', columns='Y', values='Z').T.values
            X_unique = np.sort(ddff.X.unique())
            Y_unique = np.sort(ddff.Y.unique())
            plt_x, plt_y = np.meshgrid(X_unique, Y_unique)

            fig = plt.figure()
            ax = fig.add_subplot(111)
            ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
            cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
            ax.clabel(cs, fontsize=8)
            plt.axis('scaled')
            plt.colorbar(ct)
            plt.title(f'Kriging {CONCRETE_MODEL.capitalize()} Simulate')
            plt.savefig(f'{point_folder}Plot_Kriging_{CONCRETE_MODEL.capitalize()}.png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

            fig = plt.figure()
            ax = fig.add_subplot(111)
            ct = plt.contourf(plt_x, plt_y, plt_z, cmap='plasma')
            cs = plt.contour(plt_x, plt_y, plt_z, levels=[0])
            ax.clabel(cs, fontsize=8)
            plt.scatter(df_drop['X'], df_drop['Y'], s=12, c='red')
            plt.axis('scaled')
            plt.colorbar(ct)
            plt.title(f'Kriging {CONCRETE_MODEL.capitalize()}: Simulate vs Validate')

plt.savefig(f'{point_folder}Plot_Kriging_{CONCRETE_MODEL.capitalize()}_simulate_vs_validate.png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

            # Fig SubPlot Errors
            fig, (ax1, ax2) = plt.subplots(1, 2)
            ct = ax1.contour(plt_x, plt_y, plt_z, colors='k')
            sct = ax1.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
c=df_error['Z_Error_Absolute'], cmap='plasma')
            ax1.clabel(ct, fontsize=8)
            ax1.title.set_text('Absolute Error')
            ax1.axis('scaled')
            divider1 = make_axes_locatable(ax1)
            cax1 = divider1.append_axes("right", size="5%", pad=0.1)
            col_bar = plt.colorbar(sct, ax=ax1, cax=cax1)  # shrink=0.8)
            col_bar.ax.get_yaxis().labelpad = 15
            col_bar.ax.set_ylabel('  meters  ')
            ct = ax2.contour(plt_x, plt_y, plt_z, colors='k')
            sct = ax2.scatter(df_error['X_Real'], df_error['Y_Real'], s=12,
c=df_error['Z_Error_Relative'], cmap='plasma')
            ax2.clabel(ct, fontsize=8)
            ax2.title.set_text('Relative Error')
            ax2.axis('scaled')
            divider2 = make_axes_locatable(ax2)
            cax2 = divider2.append_axes("right", size="5%", pad=0.1)
            col_bar = plt.colorbar(sct, ax=ax2, cax=cax2)  # shrink=0.8)
            col_bar.ax.get_yaxis().labelpad = 15
            col_bar.ax.set_ylabel('  %  ', rotation=270)
            fig.suptitle(f'Errors Distribution of Kriging {CONCRETE_MODEL.capitalize()}',
fontsize=16)

plt.savefig(f'{point_folder}Plot_Errors_Distribution_Kriging_{CONCRETE_MODEL.capitalize()}.png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            X, Y, Z = axes3d.get_test_data(0.05)
            ct3d = ax.contour(plt_x, plt_y, plt_z, levels=[0])
            ax.clabel(ct3d, fontsize=9, inline=1)
            ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Interp'],
                       label='Interpolate Values')
            ax.scatter(df_error['X_Real'], df_error['Y_Real'], df_error['Z_Real'],
                       label='Real Values')
            plt.legend(loc="best")
            plt.title(f'3D Compare Real - Interpolate Kriging {CONCRETE_MODEL.capitalize()}')
```

```
plt.savefig(f'{point_folder}Plot_Errors_3D_Compare_Kriging_{CONCRETE_MODEL.capitalize()}.png')
            if SHOW_PLOTS:
                plt.show()
            plt.close()

        rprint(dic_errors)
```

9.2.4 STL_Delaunay.py

```
import pandas as pd
import matplotlib.pyplot as plt
from rich import print as rprint
import numpy as np
import pyvista as pv
from tqdm.auto import tqdm
import os
import time


class Structure3D:
    point_folder = './Points/'
    if not os.path.exists(point_folder):
        os.mkdir(point_folder)
    else:
        pass

    def __init__(self, data_frame):
        self.df = data_frame

    def save_method(self, name, method):
        self.df.to_csv(f'{self.point_folder}{method}_DataFrame_{name}.csv', index=False)

    def load_last_method(self, name, method):
        self.df = pd.read_csv(f'{self.point_folder}{method}_DataFrame_{name}.csv')
        self.df.columns = ['X', 'Y', 'Z']
        return self.df

    def blocs_to_print(self):
        t = time.strftime("%Y%m%d-%H%M%S")

        self.df['X'] = self.df['X'] - min(self.df['X'])
        self.df['Y'] = self.df['Y'] - min(self.df['Y'])
        self.df['Z'] = (self.df['Z'] + abs(min(self.df['Z']))) / 10

        rprint('Values of X:    ', min(self.df['X']), '   -   ', max(self.df['X']))
        rprint('Values of Y:    ', min(self.df['Y']), '   -   ', max(self.df['Y']))
        rprint('Values of Z:    ', min(self.df['Z']), '   -   ', max(self.df['Z']))

        limit_in_x = max(self.df['X']) * 1/2
        limit_in_y = max(self.df['Y']) * 1/4

        rprint('Limit inter Bloc in X:', limit_in_x)
        rprint('Limit inter Bloc in Y:', limit_in_y)

        list_B_11 = []
        list_B_12 = []
        list_B_13 = []
        list_B_14 = []
        list_B_21 = []
        list_B_22 = []
        list_B_23 = []
        list_B_24 = []

        for data in self.df.values:

            x = data[0]
            y = data[1]
            z = data[2]

            if x < limit_in_x:
                if y < limit_in_y:
                    list_B_11.append([x, y, z])

                elif limit_in_y <= y < 2*limit_in_y:
                    list_B_12.append([x, y, z])

                elif 2*limit_in_y <= y < 3*limit_in_y:
                    list_B_13.append([x, y, z])

                elif y >= 3*limit_in_y:
                    list_B_14.append([x, y, z])

            elif x >= limit_in_x:
                if y < limit_in_y:
                    list_B_21.append([x, y, z])

                elif limit_in_y <= y < 2 * limit_in_y:
                    list_B_22.append([x, y, z])

                elif 2 * limit_in_y <= y < 3 * limit_in_y:
                    list_B_23.append([x, y, z])

                elif y >= 3 * limit_in_y:
                    list_B_24.append([x, y, z])
```

```
        rprint(list_B_11)
        df_B_11 = pd.DataFrame(list_B_11)
        rprint(df_B_11)
        df_B_11.columns = ['X', 'Y', 'Z']
        df_B_11.to_csv(f'{self.point_folder}df_B_11_{t}.csv', index=False)
        rprint('Bloc 1 1 X:   ', min(df_B_11['X']), max(df_B_11['X']))
        rprint('Bloc 1 1 Y:   ', min(df_B_11['Y']), max(df_B_11['Y']))
        rprint()

        df_B_12 = pd.DataFrame(list_B_12)
        df_B_12.columns = ['X', 'Y', 'Z']
        df_B_12.to_csv(f'{self.point_folder}df_B_12_{t}.csv', index=False)
        rprint('Bloc 1 2 X:   ', min(df_B_12['X']), max(df_B_12['X']))
        rprint('Bloc 1 2 Y:   ', min(df_B_12['Y']), max(df_B_12['Y']))
        rprint()

        df_B_13 = pd.DataFrame(list_B_13)
        df_B_13.columns = ['X', 'Y', 'Z']
        df_B_13.to_csv(f'{self.point_folder}df_B_13_{t}.csv', index=False)
        rprint('Bloc 1 3 X:   ', min(df_B_13['X']), max(df_B_13['X']))
        rprint('Bloc 1 3 Y:   ', min(df_B_13['Y']), max(df_B_13['Y']))
        rprint()

        df_B_14 = pd.DataFrame(list_B_14)
        df_B_14.columns = ['X', 'Y', 'Z']
        df_B_14.to_csv(f'{self.point_folder}df_B_14_{t}.csv', index=False)
        rprint('Bloc 1 4 X:   ', min(df_B_14['X']), max(df_B_14['X']))
        rprint('Bloc 1 4 Y:   ', min(df_B_14['Y']), max(df_B_14['Y']))
        rprint()

        df_B_21 = pd.DataFrame(list_B_21)
        df_B_21.columns = ['X', 'Y', 'Z']
        df_B_21.to_csv(f'{self.point_folder}df_B_21_{t}.csv', index=False)
        rprint('Bloc 2 1 X:   ', min(df_B_21['X']), max(df_B_21['X']))
        rprint('Bloc 2 1 Y:   ', min(df_B_21['Y']), max(df_B_21['Y']))
        rprint()

        df_B_22 = pd.DataFrame(list_B_22)
        df_B_22.columns = ['X', 'Y', 'Z']
        df_B_22.to_csv(f'{self.point_folder}df_B_22_{t}.csv', index=False)
        rprint('Bloc 2 2 X:   ', min(df_B_22['X']), max(df_B_22['X']))
        rprint('Bloc 2 2 Y:   ', min(df_B_22['Y']), max(df_B_22['Y']))
        rprint()

        df_B_23 = pd.DataFrame(list_B_23)
        df_B_23.columns = ['X', 'Y', 'Z']
        df_B_23.to_csv(f'{self.point_folder}df_B_23_{t}.csv', index=False)
        rprint('Bloc 2 3 X:   ', min(df_B_23['X']), max(df_B_23['X']))
        rprint('Bloc 2 3 Y:   ', min(df_B_23['Y']), max(df_B_23['Y']))
        rprint()

        df_B_24 = pd.DataFrame(list_B_24)
        df_B_24.columns = ['X', 'Y', 'Z']
        df_B_24.to_csv(f'{self.point_folder}df_B_24_{t}.csv', index=False)
        rprint('Bloc 2 4 X:   ', min(df_B_24['X']), max(df_B_24['X']))
        rprint('Bloc 2 4 Y:   ', min(df_B_24['Y']), max(df_B_24['Y']))
        rprint()

        return [df_B_11, df_B_12, df_B_13, df_B_14, df_B_21, df_B_22, df_B_23, df_B_24], t

    def blocs_to_print_2(self):
        t = time.strftime("%Y%m%d-%H%M%S")

        self.df['X'] = self.df['X'] - min(self.df['X'])

        self.df['Y'] = self.df['Y'] - min(self.df['Y'])

        self.df['Z'] = (self.df['Z'] + abs(min(self.df['Z']))) / 10

        rprint('Values of X:    ', min(self.df['X']), '   -   ', max(self.df['X']))
        rprint('Values of Y:    ', min(self.df['Y']), '   -   ', max(self.df['Y']))
        rprint('Values of Z:    ', min(self.df['Z']), '   -   ', max(self.df['Z']))

        limit_in_x = max(self.df['X']) * 1 / 2
        limit_in_y = max(self.df['Y']) * 1 / 3

        rprint('Limit inter Bloc in X:', limit_in_x)
        rprint('Limit inter Bloc in Y:', limit_in_y)

        list_B_11 = []
        list_B_12 = []
        list_B_13 = []
        # list_B_14 = []
        list_B_21 = []
        list_B_22 = []
        list_B_23 = []
        # list_B_24 = []
```

```
    for data in self.df.values:

        x = data[0]
        y = data[1]
        z = data[2]

        if x < limit_in_x:
            if y < limit_in_y:
                list_B_11.append([x, y, z])

            elif limit_in_y <= y < 2 * limit_in_y:
                list_B_12.append([x, y, z])

            elif 2 * limit_in_y <= y < 3 * limit_in_y:
                list_B_13.append([x, y, z])

            # elif y >= 3 * limit_in_y:
            #     list_B_14.append([x, y, z])

        elif x >= limit_in_x:
            if y < limit_in_y:
                list_B_21.append([x, y, z])

            elif limit_in_y <= y < 2 * limit_in_y:
                list_B_22.append([x, y, z])

            elif 2 * limit_in_y <= y < 3 * limit_in_y:
                list_B_23.append([x, y, z])

            # elif y >= 3 * limit_in_y:
            #     list_B_24.append([x, y, z])

    rprint(list_B_11)
    df_B_11 = pd.DataFrame(list_B_11)
    rprint(df_B_11)
    df_B_11.columns = ['X', 'Y', 'Z']
    df_B_11.to_csv(f'{self.point_folder}df_B_11_{t}.csv', index=False)
    rprint('Bloc 1 1 X:   ', min(df_B_11['X']), max(df_B_11['X']))
    rprint('Bloc 1 1 Y:   ', min(df_B_11['Y']), max(df_B_11['Y']))
    rprint()

    df_B_12 = pd.DataFrame(list_B_12)
    df_B_12.columns = ['X', 'Y', 'Z']
    df_B_12.to_csv(f'{self.point_folder}df_B_12_{t}.csv', index=False)
    rprint('Bloc 1 2 X:   ', min(df_B_12['X']), max(df_B_12['X']))
    rprint('Bloc 1 2 Y:   ', min(df_B_12['Y']), max(df_B_12['Y']))
    rprint()

    df_B_13 = pd.DataFrame(list_B_13)
    df_B_13.columns = ['X', 'Y', 'Z']
    df_B_13.to_csv(f'{self.point_folder}df_B_13_{t}.csv', index=False)
    rprint('Bloc 1 3 X:   ', min(df_B_13['X']), max(df_B_13['X']))
    rprint('Bloc 1 3 Y:   ', min(df_B_13['Y']), max(df_B_13['Y']))
    rprint()

    # df_B_14 = pd.DataFrame(list_B_14)
    # df_B_14.columns = ['X', 'Y', 'Z']
    # df_B_14.to_csv(f'{self.point_folder}df_B_14_{t}.csv', index=False)
    # rprint('Bloc 1 4 X:   ', min(df_B_14['X']), max(df_B_14['X']))
    # rprint('Bloc 1 4 Y:   ', min(df_B_14['Y']), max(df_B_14['Y']))
    # rprint()

    df_B_21 = pd.DataFrame(list_B_21)
    df_B_21.columns = ['X', 'Y', 'Z']
    df_B_21.to_csv(f'{self.point_folder}df_B_21_{t}.csv', index=False)
    rprint('Bloc 2 1 X:   ', min(df_B_21['X']), max(df_B_21['X']))
    rprint('Bloc 2 1 Y:   ', min(df_B_21['Y']), max(df_B_21['Y']))
    rprint()

    df_B_22 = pd.DataFrame(list_B_22)
    df_B_22.columns = ['X', 'Y', 'Z']
    df_B_22.to_csv(f'{self.point_folder}df_B_22_{t}.csv', index=False)
    rprint('Bloc 2 2 X:   ', min(df_B_22['X']), max(df_B_22['X']))
    rprint('Bloc 2 2 Y:   ', min(df_B_22['Y']), max(df_B_22['Y']))
    rprint()

    df_B_23 = pd.DataFrame(list_B_23)
    df_B_23.columns = ['X', 'Y', 'Z']
    df_B_23.to_csv(f'{self.point_folder}df_B_23_{t}.csv', index=False)
    rprint('Bloc 2 3 X:   ', min(df_B_23['X']), max(df_B_23['X']))
    rprint('Bloc 2 3 Y:   ', min(df_B_23['Y']), max(df_B_23['Y']))
    rprint()

    return [df_B_11, df_B_12, df_B_13, df_B_21, df_B_22, df_B_23], t

def make_stl(self, df_block, mesh_name_to_save, alfa_Delaunay_3D=1):
    rprint('Starting to generate a Delaunay STL ...')
    points = df_block.to_numpy()
```

```
        cloud = pv.PolyData(points)
        cloud.plot()

        volume = cloud.delaunay_2d()
        shell = volume.extract_geometry()
        shell.save(mesh_name_to_save + '.stl')
        rprint("The Delaunay STL it's done")
        shell.plot()


point_folder = './Points/'
if not os.path.exists(point_folder):
    os.mkdir(point_folder)
else:
    pass

NAME = 'NAME_OF_DATASET'
MODEL = 'METHOD'
NUM_POINTS = 'NUM_POINTS' + '_points'
STL_TOTAL = True
STL_BLOCS = False

df = pd.read_csv(f'{point_folder}{MODEL}_{NAME}_{NUM_POINTS}.csv', header=None)
df.columns = ['X', 'Y', 'Z']

rprint("File it's open to create a Delaunay STL")

df['Z'] = df['Z'] / 3000

structure_3d_fit = Structure3D(df)
'''STL en Blocs'''
if STL_BLOCS:
    blocks, time = structure_3d_fit.blocs_to_print_2()
    x = 1
    for block in tqdm(blocks):
        # print(x)
        # rprint(block, type(block))
        name = f'{x}'
        structure_3d_fit.make_stl(block, name, alfa_Delaunay_3D=1)
        x += 1

'''STL de tot el Bloc'''
if STL_TOTAL:
    structure_3d_fit.make_stl(df, f'{point_folder}{MODEL}_{NAME}_{NUM_POINTS}')
    # structure_3d_fit.from_df_to_txt(df, MODEL, NAME, NUM_POINTS)
```