Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

TFInterpy: A high-performance spatial interpolation Python package

Zhiwen Chen, Baorong Zhong [*]

School of Computer Science, Yangtze University, Jingzhou, 434000, China

A R T I C L E   I N F O

Article history:
Received 6 April 2022
Received in revised form 15 July 2022
Accepted 6 October 2022

MSC:
65D05
86A32

Keywords:
Python
TensorFlow
Interpolation
Kriging

A B S T R A C T

Interpolation algorithms are essential tools for spatial analysis. The Kriging method satisfies the best linear unbiased estimation and is widely used in scenarios where high accuracy is required. But the program running time may be unacceptably long when using the Kriging method for large dataset. To solve the problem, we developed TFInterpy based on the TensorFlow framework. This Python package provides an open-source, cross-platform, easy-to-use API for interpolation algorithms and achieves significant speedups when applied to large-scale tasks.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00084
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License, version 3 (GPLv3)
Code versioning system used	git
Software code languages, tools, and services used	Python3
Compilation requirements, operating environments & dependencies	Python3, TensorFlow, Keras, NumPy, Pandas, Scipy, VTK, PyQT5
If available Link to developer documentation/manual	
Support email for questions	orchenz@qq.com

1. Motivation and significance

Spatial analysis can only collect a few sample points in most cases. In order to build a visual model for the target region, it is necessary to use spatial interpolation techniques to estimate the relevant attribute values of unsampled locations. Inverse distance weighted (IDW) and Kriging methods are widely used in environmental science [1], geological exploration [2], hydrogeology [3], and other fields. IDW method is the most simple interpolation method, considering only the distance between sample points and the target point, and the contribution of the weight of the sample points to the target point is inversely proportional to the distance. IDW can also be regarded as one of radial basis function methods with hyperbolic kernel. The Kriging method is regarded as the optimal unbiased estimation, which can minimize the variance [4]. Compared to IDW, the Kriging method considers more complex spatial correlations. It uses a variogram function to fit the correlations between the distance and the difference in attributes, which results in higher interpolation accuracy than other algorithms. The Kriging method contains a series of variants derived from Ordinary Kriging (OK), e.g., Simple Kriging (SK), Indicator Kriging (IK), and Co-Kriging (CK). In general, these algorithms require solving a set of linear equations, known as the Kriging equation system. Solving the Kriging equation system involves matrix multiplication and matrix inversion operations, and it is preceded by calculating the value of the variogram function. As a result, the program can be very time-consuming when the number of target points to be estimated is large, especially in 3D modeling, where the number of target points can reach millions or more. Therefore, improving the execution speed has been a hot research topic for the Kriging algorithm.

* Corresponding author.
E-mail address: barry@yangtzeu.edu.cn (Baorong Zhong).

https://doi.org/10.1016/j.softx.2022.101229

Regarding improving the speed of the interpolation procedure, there have been many studies proposing solutions. Ferreira et al. [5] developed Kriging with parallelizable computation using OpenMP programming. Hu and Shu [6] built a distributed memory system based on MPI and implemented parallel processes for Ordinary Kriging (OK). Cheng [7], Gutiérrez de Ravé et al. [8], Liu [9] successively proposed Kriging based on CUDA programming for GPU acceleration. Misra et al. [10] implemented a distributed running OK based on the Spark framework. All the above schemes led to a significant improvement in running speed, but unfortunately, none of these studies have made their source code open source. In addition, the above solutions require high programming skills, which, together with the complexity of the Kriging algorithm itself, make reproducing these solutions very difficult.

The Python language has a well-established software ecosystem and many other advantages, making it increasingly used in scientific computing and engineering applications. The Kriging algorithm is an essential tool for spatial analysis, and there are Python packages that provide APIs for it, including GeostatsPy [11] and PyKrige [12]. The goal of GeostatsPy is to implement the algorithms in the GSLIB [13] library using the Python language. Currently, GeostatsPy contains only the Kriging algorithm for two-dimensional space, and the numerical operations depend on the executable provided by GSLIB. PyKrige provides various Kriging algorithms, including interfaces for 3D interpolation, but does not provide tools for 3D visualization.

In order to solve the above problem, we developed TFInterpy (TensorFlow-based Interpolation Python package). TFInterpy contains TensorFlow-based implementations of IDW, OK, and SK (Simple Kriging), which implies that the interpolation algorithms can be implemented in parallel by TensorFlow and can use the computing power of GPU devices. TFInterpy also provides the conventional version of these algorithms. TFInterpy is a package built on top of many highly portable Python packages, so TFInterpy also features high portability, allowing users to create cross-platform applications based on it. Simplicity and ease of use are one of the design goals of TFInterpy. It provides default functions for calculating the variogram function, which can lower the threshold of use for non-specialists. Meanwhile, TFInterpy also provides basic 2D and 3D data visualization solutions to allow the user understand the interpolation results instantly. In addition, the GUI sub-package under TFInterpy provides a GUI tool built based on the other core modules.

2. Software description

TFInterpy is designed for local interpolation considering only neighboring sample points, but global interpolation can also be achieved by adjusting the neighborhood search size. TFInterpy provides various functions that may be used in the spatial analysis process, including data loading, interpolation calculation, data visualization, and data export. The coupling between the modules is kept low, so users do not need to install all the dependencies if users only need to use partial functions. For example, installing TensorFlow and Keras [14] (TensorFlow backend) is unnecessary when using only the conventional implementation of the interpolation algorithm. Unlike the function interfaces provided in PyKrige and GeostatsPy, the interpolation functions in TFInterpy do not use the coordinates and dimensions of the structured grid as parameters. To maintain maximum flexibility, TFInterpy uses the target point coordinates as parameters. Scatter, structured grid and unstructured grid can pass the target point coordinates to the function interface to get the interpolation results.

2.1. Software architecture

TFInterpy relies on many excellent open-source Python packages. NumPy [15] for data operations; Pandas [16] for data import and export; KD tree in SciPy [17] for neighborhood search, least squares for fitting variogram functions; TensorFlow for implementing parallelizable interpolation algorithm; Matplotlib [18] for 2D plotting; VTK [19] for 3D visualization and PyQT5 [20] for building GUI tools. Table 1 lists the main components of TFInterpy, the corresponding functional descriptions, and the required third-party dependency libraries. Fig. 1 shows the dependencies between components within TFInterpy. The package shape in Fig. 1 represents a package or module in Python, and the package shape containing subitems represents a package; otherwise, it represents a module. The arrows in Fig. 1 indicate dependencies; for example, "krige" pointing to "utils" means "krige" depends on "utils".

2.2. Software functionalities

2.2.1. Interpolation

The Spatial interpolation can be expressed as the weighted combination of observed values Eq. (1), where n denotes the number of neighborhood points (i.e., neighborhood size), $\overline{Z}$ is the estimated result of the target point, Z_i is the attribute value of the ith neighborhood point, and λ_i denotes the weight of the influence of the ith neighborhood point (sample point). The weight calculation rule of IDW is as Eq. (2), where h_i denotes the Euclidean distance between the ith sample point and the target point, the exponent α indicates the degree of influence of the distance, and the higher α value will give lower weight. The value of α is usually taken as 2.

$$\overline{Z} = \sum_{i=1}^{n} \lambda_i Z_i \tag{1}$$

$$\lambda_i = \frac{1/h_i^{\alpha}}{\sum_{i=1}^{n}(1/h_i^{\alpha})} \tag{2}$$

Suppose there are target point x and n neighborhood points $\{x_1, x_2, \ldots, x_n\}$, the Ordinary Kriging equation system is as Eq. (3), where μ is a Lagrange multiplier introduced for the minimization of the error (i.e., Kriging variance) [4], and the formula for Ordinary Kriging variance is as Eq. (4). The calculation of the weighted combination of the OK method can refer to Eq. (1), and Z_i in Eq. (1) corresponds to the attribute value corresponding to the neighborhood point x_i. The Simple Kriging equation system is as Eq. (6), the formula for Simple Kriging variance is as Eq. (7), and the calculation of the weighted combination of the OK method can refer to Eq. (5). The m in Eq. (5) is the mathematical expectation of the regionalized variable $Z(x)$ [4], m takes the mean value of the sample points' attributes in practical application. The difference between Eq. (3) and (6) is that Eq. (3) constrains the sum of weights λ to be 1. The covariance $C(x, y)$ in Eq. (3), (4), (6) and (7) is got by calculating the variogram function.

$$\begin{bmatrix} C(x_1,x_1) & \cdots & C(x_1,x_n) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ C(x_n,x_1) & \cdots & C(x_n,x_n) & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \\ -\mu \end{bmatrix} = \begin{bmatrix} C(x,x_1) \\ \vdots \\ C(x,x_n) \\ 1 \end{bmatrix} \tag{3}$$

$$\sigma_{ok}^2 = C(x,x) - \sum_{i=1}^{n} \lambda_i C(x,x_i) + \mu \tag{4}$$

Table 1
Main components of TFInterpy.

Component	Type	Description	Dependencies
grid	module	Used for building rectilinear grid	numpy
idw	module	IDW algorithm	numpy, scipy
krige	module	SK and OK algorithms	numpy, scipy
variogram	module	Used for calculating the variogram function	numpy, scipy
variogramExp	module	Used for calculating experimental variogram function	numpy
variogramModel	module	Variogram function model	numpy
utils	module	Contains helper functions	numpy
tf	subpackage	TensorFlow implementations of the algorithms listed above	numpy, scipy, tensorflow
gslib	subpackage	Used for reading and writing GSLIB files	pandas
vtk	subpackage	Used for data rendering and model export based on VTK	numpy, vtk
gui	subpackage	GUI tool based on all the above features	matplotlib, numpy, pandas, PyQt5, scipy, vtk

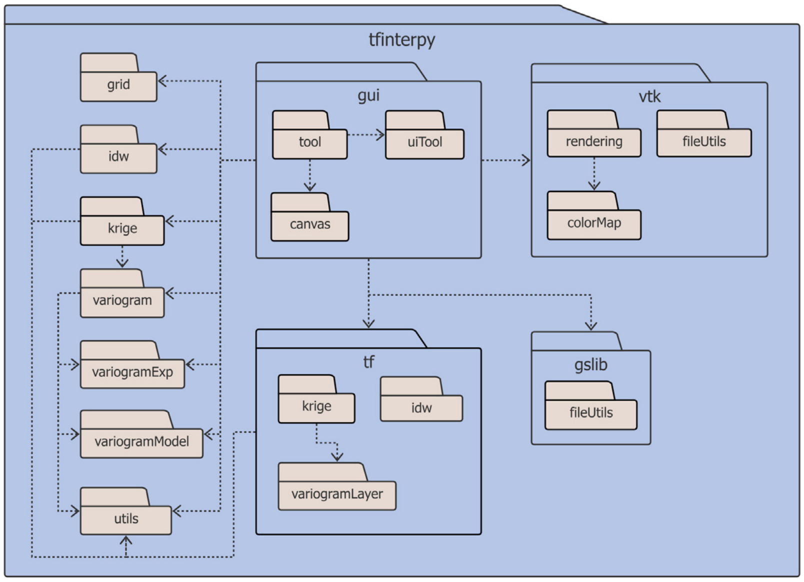


Fig. 1. Dependencies between components within TFInterpy.

$$\overline{Z} = \sum_{i=1}^{n} \lambda_i Z_i + m(1 - \sum_{i=1}^{n} \lambda_i) \tag{5}$$

$$\begin{bmatrix} C(x_1, x_1) & \cdots & C(x_1, x_n) \\ \vdots & \ddots & \vdots \\ C(x_n, x_1) & \cdots & C(x_n, x_n) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} C(x, x_1) \\ \vdots \\ C(x, x_n) \end{bmatrix} \tag{6}$$

$$\sigma_{sk}^2 = C(x, x) - \sum_{i=1}^{n} \lambda_i C(x, x_i) \tag{7}$$

Both IDW and Kriging methods are encapsulated as interpolator classes, and each interpolator class provides a function named "execute", which encapsulates the steps of neighborhood search, solving for weights, and the weighted combination of observed values. The minimum parameter requirement for calling the "execute" function is to provide a 2D array with each row representing the coordinates of a target point. Number of nearest neighbor points, variogram function, and other parameters are optional, and the function will return the interpolation results.

TensorFlow-based interpolator classes are named with "TF" in TFInterpy, including TFIDW, TFSK, and TFOK. These classes provide function interfaces consistent with the conventional implemented interpolator classes. The implementation principle of these classes is to use Keras' functional API to build the computational part as a TensorFlow data flow graph, which represents Keras' "Model" object. For example, Fig. 2 shows the structure of the data flow graph for TFSK. The "None" in the shape tuple means that the received data size is arbitrary, which corresponds to the "batch size" concept in TensorFlow. "Input layer 1" is the matrix on the left side of Eq. (6), "Input layer 2" is the vector on the right side of Eq. (6), and "Input layer 3" is the properties of the sample points.

2.2.2. Variogram function calculation

The variogram function maps location offset to semi-variance, the formula is defined in Eq. (8), $Z(x)$ denotes the property of the regionalized variable at position x, and h is the separation between sample points in both Euclidean distance and direction [4], which can be regarded as a vectorial argument. Under the second-order stationarity assumption in geostatistics, the covariance between two points is only related to the distance and direction [4], and $C(x_i, x_j)$ in Eq. (3) and (6) can be expressed as $C(h_{i,j})$, and $h_{i,j}$ denotes the location offset between x_i and x_j. As shown in Eq. (9), the covariance and variogram functions are equivalent under the second-order stationarity assumption [4], that means $C(h_{i,j})$ can express as $\gamma(h_{i,j})$.

$$\gamma(h) = \frac{1}{2} var(Z(x) - Z(x + h)) \tag{8}$$

$$\gamma(h) = C(0) - C(h) \tag{9}$$

The computation of the variogram function involves the search for experimental variogram function point pairs and the fitting of theoretical variogram function models, which involve many complex concepts [4]. TFInterpy provides corresponding functions

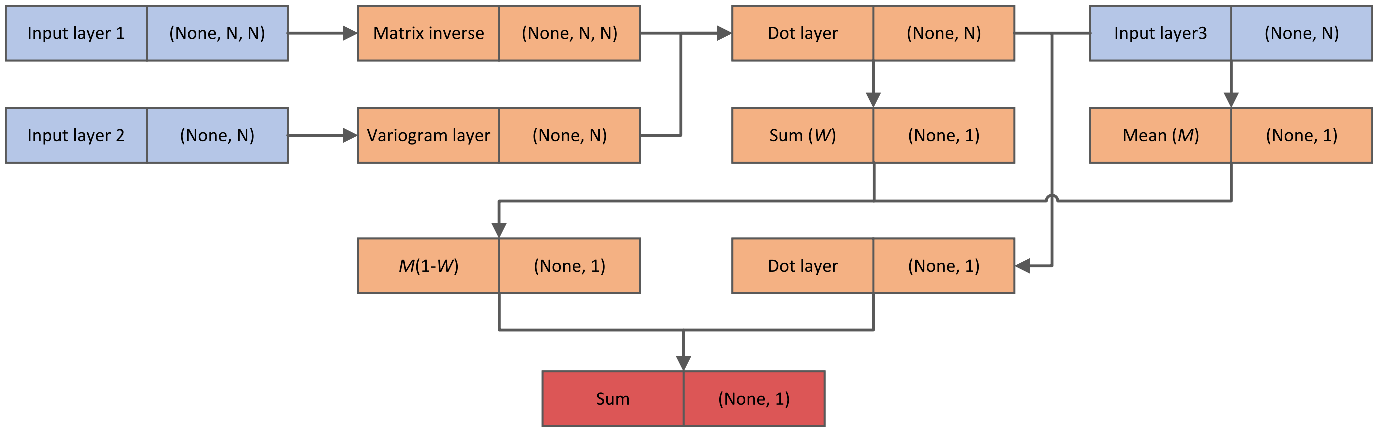


Fig. 2. Data flow graph of TFSK.

```
1  # Example of using IDW interpolator.
2  # Construct an IDW interpolator.
3  exe = IDW(samples, '3d')
4  # Perform leave-one-out cross-validation and print results.
5  print(exe.crossValidate(N))
6  # Perform interpolation.
7  grid.properties = exe.execute(grid.points(), N)
8
9  # Example of using TFIDW interpolator.
10 # Construct a TFIDW interpolator.
11 exe = TFIDW(samples, '3d')
12 # Specify the device used for computing as GPU 0.
13 with tf.device("/GPU:0"):
14     # Perform K-fold cross-validation (K=10) and print the results.
15     print(exe.crossValidateKFold(10, N))
16     # Perform interpolation.
17     grid.properties = exe.execute(grid.points(), N)
```

Fig. 3. Code snippet 1.

to handle these steps. However, these functions require many parameters to be set by the user, and non-experts cannot understand how to set them correctly and how these parameters will affect the interpolation. For this reason, TFInterpy additionally provides default functions for computing the variogram functions. This process infers the values of each parameter based on the spatial distribution characteristics of the scatter. The angle range representing the direction is divided into multiple portions, and the distance-variance pairs in the direction with the most pairs are used to fit the variogram function. This approach does not guarantee optimal modeling but allows users to judge how to adjust the parameters by the results of the first interpolation.

2.3. Sample code snippets analysis

The Code snippet in Fig. 3 shows how to use the IDW algorithm. The code in line 3 constructs a regular IDW interpolator using sample points; the code in line 5 performs leave-one-out cross-validation and prints the results; the code in line 7 calls the interpolator to estimate the grid points; the code in line 11 constructs an TFIDW interpolator based on the TensorFlow implementation and the code in line 13 specifies that TensorFlow uses the GPU device. In the code snippet shown in Fig. 3 and presented later, the variable “samples” is a 2D Numpy array with coordinates in the first three columns and property in the last column, variable “grid” is a Grid3D object representing a 3D rectilinear grid, and variable “N” indicates the number of nearest neighbor points.

The Code snippet in Fig. 4 shows how to use an OK interpolator with a default omnidirectional variogram function. The code in line 2 uses the sample points to calculate the nested variogram function; the code in line 4 constructs an OK interpolator using the sample points; the code in line 6 calculates the interpolation result and the Kriging variance; the code in line 8, 9 constructs a “vtkActor” object using the grid data, and specify the color map as “Rainbow”; the code in line 11 creates a new rendering window and renders the “vtkActor” object and the code in line 13 saves the data in VTK format.

The Code snippet in Fig. 5 shows how to use TFOK interpolator. The auxiliary function can convert the nested variogram function to “Layer” in TensorFlow, as in line 4. The *batch_size* in line 10 specifies the number of batches to be passed in for parallel operations. It should not set the *batch_size* too small; otherwise, the program will not get speedup or even slower than the conventional implementation. So the default value of *batch_size* is set to 10000.

3. Illustrative examples

This section shows examples of 2D and 3D scene interpolation. We have verified the GUI tool example on both Windows and Linux operating systems. Hardware environment: AMD Ryzen 7 5800H CPU, 16 GB RAM, NVIDIA GeForce RTX 3050 Laptop GPU.

3.1. Example of interpolation of elevation data for a 2D scene

The following example shows the results of a 2D interpolation using a submap of a large-scale high-resolution digital elevation model [21]. A TFSK is executed by randomly selecting 100 sample points on the original data and constructing a 2D grid consistent with the original map size (100 × 100) for interpolation. Fig. 6(a) shows the subplot of the original elevation data, the black dots are sample points, Fig. 6(b) shows the estimate result, Fig. 6(c) shows the absolute error between the estimates and the original data, and Fig. 6(d) shows the corresponding Kriging variance.

3.2. Example of 3D interpolation in GUI tool

In this example, the results of rectilinear grid 3D interpolation are shown with data from the SGeMS [22] tutorial example. The interface’s tools on the left sidebar allow users to import, interpolate, display, and export data. Users can check the “use tensorflow” checkbox to decide whether to use the algorithm implemented in TensorFlow. Fig. 7 shows the fitted plots of the variogram function in each direction, Fig. 8 shows the results of performing leave-one-out cross-validation using TFOK, and Fig. 9 shows the results of performing interpolation using TFOK.

```python
# Calculate a nested variogram function.
nestVariogram, _ = calculateOmnidirectionalVariogram3D(samples)
# Construct an OK interpolator.
exe = OK(samples, '3d')
# Perform interpolation.
grid.properties, grid.sigma = exe.execute(grid.points(), N, nestVariogram)
# Construct a vtkActor object using the grid data, and specify the color map as "rainbow".
actor = createGridActor(*grid.dim, grid.x, grid.y, grid.z, grid.properties,
    [samples[:, 3].min(), samples[:, 3].max()], colorMap.Rainbow)
# Create a new rendering window and render the actor.
rendering(actor)
# Saves the data in VTK format.
saveVTKGrid('savedData/grid.vtk', actor.GetMapper().GetInput())
```

Fig. 4. Code snippet 2.

```python
# Calculate a nested variogram function.
nestVariogram, variogramBuilders = calculateOmnidirectionalVariogram3D(samples)
# Construct the corresponding nested variogram function Layer.
nestVariogramLayer = getNestVariogramLayer(variogramBuilders, nestVariogram.unitVectors)
# Construct a TFOK interpolator.
exe = TFOK(samples, '3d')
# Specify the device used for computing as GPU 0.
with tf.device("/GPU:0"):
    # Perform interpolation.
    grid.properties, grid.sigma = exe.execute(grid.points(), N, nestVariogramLayer, batch_size=10000)
```

Fig. 5. Code snippet 3.

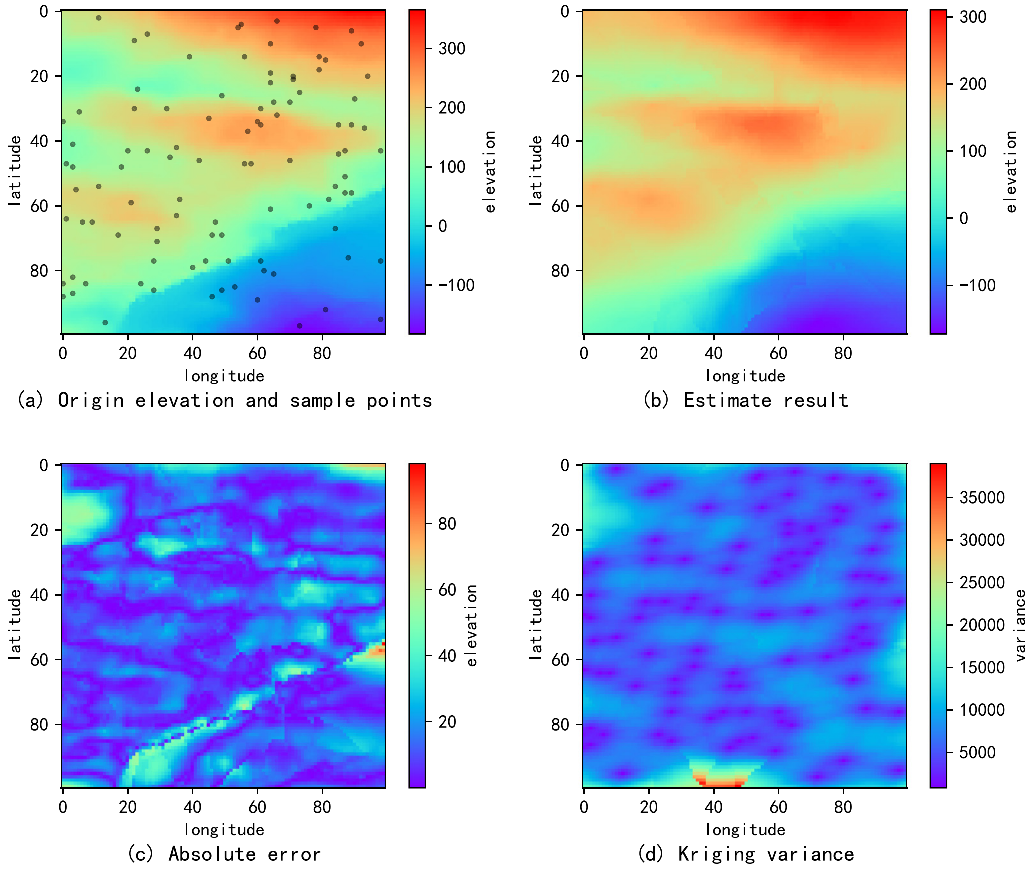
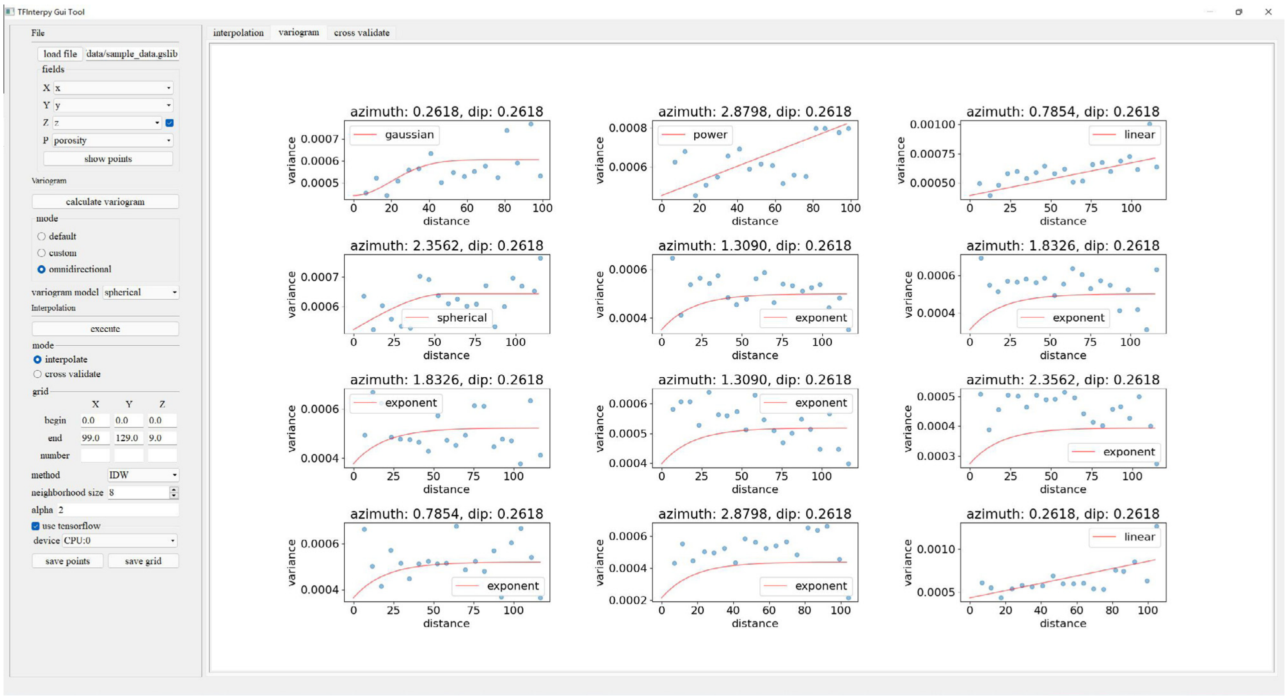


Fig. 6. Performing 2D interpolation using TFSK.

3.3. Performance comparison

We use different granularities of the grid to compare the execution time of OK in different implementations including GeostatsPy, PyKrige and TFInterpy. The results are shown in Table 2, "TFInterpy-OK" indicates the conventional implementation of OK in TFInterpy, "TFInterpy-TFOK" indicates the TensorFlow-based implementation of OK in TFInterpy. It can be seen that the performance advantage of TFInterpy-TFOK becomes more and more obvious as the number of target points grows. Compared to GeostatsPy, TFOK (CPU used) can achieve more than 20 times performance improvement, and it can achieve up to 10 times performance improvement compared to PyKrige. The TFOK (GPU used) is surprisingly slower than the TFOK (CPU used) due to the long time-consuming data copying from computer memory


TFInterpy Gui Tool

File

load file data/sample_data.gslib

fields

X x

Y y

Z z

P porosity

show points

Variogram

calculate variogram

mode

default

custom

[✓ omnidirectional]

variogram model spherical

Interpolation

execute

mode

[✓ interpolate]

cross validate

grid

	X	Y	Z
begin	0.0	0.0	0.0
end	99.0	129.0	9.0
number			

method IDW

neighborhood size 8

alpha 2

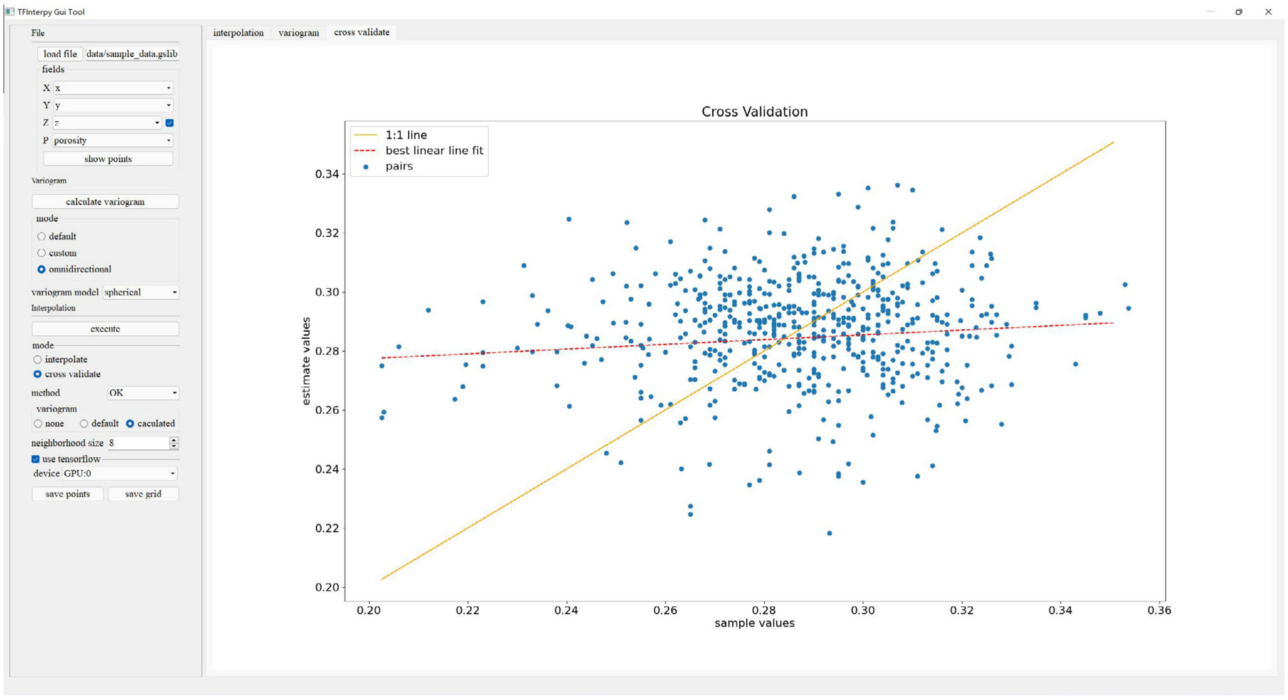
[✓ use tensorflow]

device CPU:0

save points save grid

interpolation
variogram
cross validate
azimuth: 0.2618, dip: 0.2618
gaussian
azimuth: 2.8798, dip: 0.2618
power
azimuth: 0.7854, dip: 0.2618
linear
azimuth: 2.3562, dip: 0.2618
spherical
azimuth: 1.3090, dip: 0.2618
exponent
azimuth: 1.8326, dip: 0.2618
exponent
azimuth: 1.8326, dip: 0.2618
exponent
azimuth: 1.3090, dip: 0.2618
exponent
azimuth: 2.3562, dip: 0.2618
exponent
azimuth: 0.7854, dip: 0.2618
exponent
azimuth: 2.8798, dip: 0.2618
exponent
azimuth: 0.2618, dip: 0.2618
linear
variance
distance

Fig. 7. Variogram function fitting results.


TFInterpy Gui Tool

File

load file data/sample_data.gslib

fields

X x

Y y

Z z

P porosity

show points

Variogram

calculate variogram

mode

default

custom

[✓ omnidirectional]

variogram model spherical

Interpolation

execute

mode

interpolate

[✓ cross validate]

method OK

variogram

none default [✓ caculated]

neighborhood size 8

[✓ use tensorflow]

device GPU:0

save points save grid

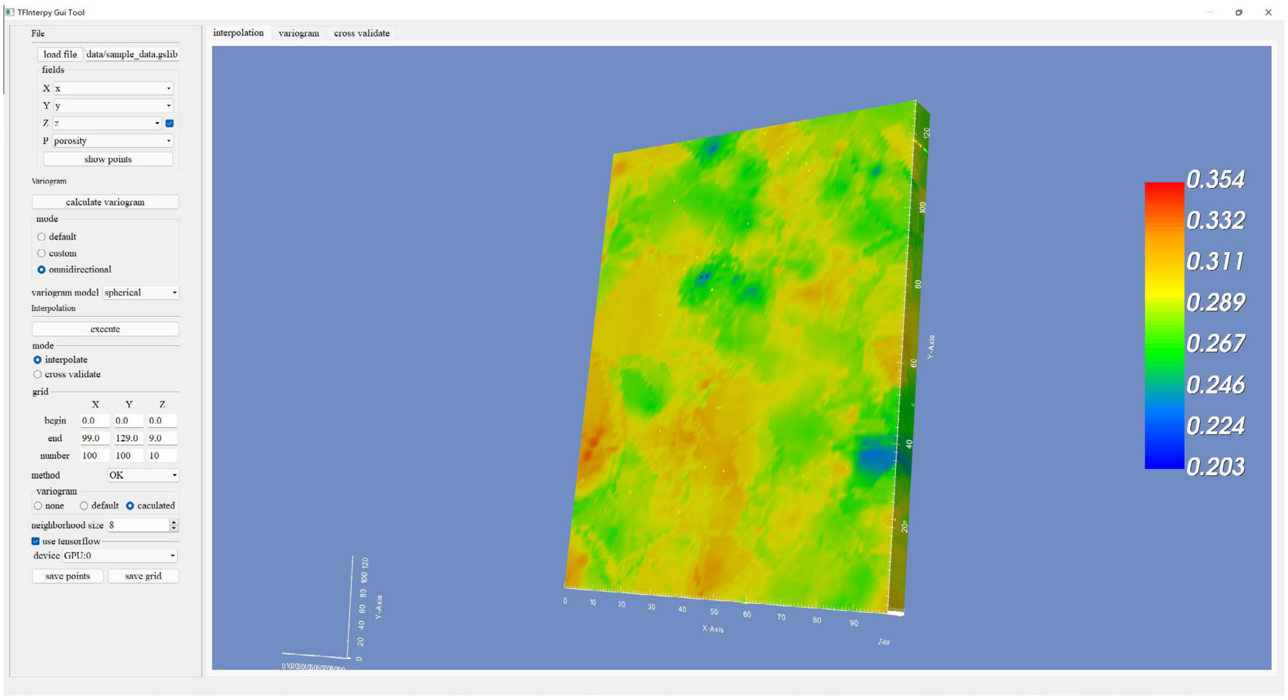
Fig. 8. Leave-one-out cross-validation results of TFOK.

to graphics card memory and the similar computing performance of the CPU and GPU in the experimental environment.

4. Impact

In previous research, accelerating interpolation algorithms has revolved around parallelizing execution, utilizing the computing power of GPU, distributed computing, and often using programming techniques with low portability. TFInterpy is developed base on the Python language and the TensorFlow framework, offering a completely different solution. We can use CPU, GPU, or even TPU devices simply by specifying the device name, and it has the potential for distributed computing. In addition, TFInterpy is highly portable and scalable.

TFInterpy is an open-source, cross-platform package that offers higher computational efficiency and better visualization support than similar Python libraries available today. TFInterpy is flexible, defining a default algorithm execution process while keeping parameters customizable. It provides users with a concise API, which significantly lowers the threshold for non-specialists



data/sample_data.gslib

fields: X x; Y y; Z z [✓]; P porosity

mode: ( ) default ( ) custom [✓ omnidirectional]

variogram model: spherical

mode: [✓ interpolate] ( ) cross validate

grid	X	Y	Z
begin	0.0	0.0	0.0
end	99.0	129.0	9.0
number	100	100	10

method: OK

variogram: ( ) none ( ) default [✓ caculated]

neighborhood size: 8

[✓] use tensorflow

device: GPU:0

Fig. 9. TFOK's 3D interpolation results.

Table 2
The elapsed time of each interpolator applies to different grid sizes (unit: second).

Grid size	Interpolator				
	GeostatsPy-OK	PyKrige-OK	TFInterpy-OK	TFInterpy-TFOK (GPU)	TFInterpy-TFOK (CPU)
$1 \times 10^4 = 100 \times 100$	23.977	1.258	0.828	2.070	0.979
$1 \times 10^5 \approx 317 \times 317$	230.299	12.264	8.140	6.239	2.067
$1 \times 10^6 = 1000 \times 1000$	2011.351	121.711	82.397	45.737	11.683
$1 \times 10^7 \approx 3163 \times 3163$	2784.843	1250.980	849.974	452.567	112.331

to use. Among the potential users, geologists can best appreciate the performance gains brought by TFInterpy, as their spatial analysis tasks often require processing larger datasets, which are more likely to take advantage of TensorFlow's parallel computing.

5. Conclusions

Spatial interpolation is essential in several disciplines, and the Kriging method can achieve high interpolation accuracy but with high computational cost. Although many previous studies have described how to reduce the program running time, there is still no open-source solution. Therefore, we propose the TFInterpy, which implements algorithm acceleration based on TensorFlow. This paper introduces various features of TFInterpy, which provide necessary and auxiliary API support for spatial interpolation, and provide significant performance improvements over existing Python packages. In future work, more interpolation algorithms will add to TFInterpy, and we will investigate the feasibility of distributed execution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This research has been funded by the National Science and Technology Major Project of China in the early stages under grant number 2016ZX05055.

References

[1] Metahni S, Coudert L, Gloaguen E, Guemiza K, Mercier G, Blais J-F. Comparison of different interpolation methods and sequential Gaussian simulation to estimate volumes of soil contaminated by As, Cr, Cu, PCP and dioxins/furans. Environ Pollut 2019;252:409–19. http://dx.doi.org/10.1016/j.envpol.2019.05.122, URL https://www.sciencedirect.com/science/article/pii/S0269749119306311.

[2] Khaba L, Griffiths JA. Calculation of reservoir capacity loss due to sediment deposition in the 'Muela reservoir, Northern Lesotho. Int Soil Water Conserv Res 2017;5(2):130–40. http://dx.doi.org/10.1016/j.iswcr.2017.05.005, URL https://www.sciencedirect.com/science/article/pii/S2095633916300041.

[3] Belkhiri L, Tiri A, Mouni L. Spatial distribution of the groundwater quality using kriging and Co-kriging interpolations. Groundw Sustain Dev 2020;11:100473. http://dx.doi.org/10.1016/j.gsd.2020.100473, URL https://www.sciencedirect.com/science/article/pii/S2352801X20300461.

[4] Oliver M, Webster R. A tutorial guide to geostatistics: Computing and modelling variograms and kriging. Catena 2014;113:56–69. http://dx.doi.org/10.1016/j.catena.2013.09.006, URL https://www.sciencedirect.com/science/article/pii/S0341816213002385.

[5] Ferreira A, Ponte F, Varella L. Parallelization of KT3D application of GSLIB library using coprocessor Intel Xeon Phi with OpenMP. In: 76th EAGE Conference and Exhibition 2014. 2014, (1):European Association of Geoscientists & Engineers; 2014, p. 1–5. http://dx.doi.org/10.

3997/2214-4609.20141291, URL https://www.earthdoc.org/content/papers/10.3997/2214-4609.20141291.

[6] Hu H, Shu H. An improved coarse-grained parallel algorithm for computational acceleration of ordinary Kriging interpolation. Comput Geosci 2015;78:44–52. http://dx.doi.org/10.1016/j.cageo.2015.02.011, URL https://www.sciencedirect.com/science/article/pii/S0098300415000333.

[7] Cheng T. Accelerating universal Kriging interpolation algorithm using CUDA-enabled GPU. Comput Geosci 2013;54:178–83. http://dx.doi.org/10.1016/j.cageo.2012.11.013, URL https://www.sciencedirect.com/science/article/pii/S0098300412003901.

[8] Gutiérrez de Ravé E, Jiménez-Hornero F, Ariza-Villaverde A, Gómez-López J. Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm. Comput Geosci 2014;64:1–6. http://dx.doi.org/10.1016/j.cageo.2013.11.004, URL https://www.sciencedirect.com/science/article/pii/S0098300413002914.

[9] Liu Y. Research and development of 3D Kriging interpolation algorithm based on gpu (Master's thesis), University of Electronic Science and Technology of China; 2021.

[10] Misra C, Bhattacharya S, Ghosh SK. A fast scalable distributed kriging algorithm using spark framework. Int J Data Sci Anal 2020;10(3):249–64.

[11] Pyrcz M, Jo H, Kupenko A, Liu W, Gigliotti A, Salomaki T, et al. GeostatsPy. 2021, https://github.com/GeostatsGuy/GeostatsPy.

[12] Murphy B, Müller S, Yurchak R. GeoStat-framework/PyKrige: v1.6.1. Zenodo; 2021, http://dx.doi.org/10.5281/zenodo.5380342.

[13] Deutsch CV, Journel AG. Gslib. Geostatistical software library and user's guide. Vol. 369. New York: Oxford University Press; 1998.

[14] Chollet F, et al. Keras. 2015, https://keras.io.

[15] Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62.

[16] McKinney W, et al. Pandas: a foundational Python library for data analysis and statistics. In: Python for high performance and scientific computing. Vol. 14. (9):Seattle; 2011, p. 1–9.

[17] Virtanen P, Gommers R, Oliphant T, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods 2020;17:1–12. http://dx.doi.org/10.1038/s41592-019-0686-2.

[18] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(03):90–5.

[19] Hanwell MD, Martin KM, Chaudhary A, Avila LS. The visualization toolkit (VTK): Rewriting the rendering code for modern graphics cards. SoftwareX 2015;1–2:9–12. http://dx.doi.org/10.1016/j.softx.2015.04.001, URL https://www.sciencedirect.com/science/article/pii/S2352711015000035.

[20] Riverbank Computing Limited. PyQT5. 2022, https://www.riverbankcomputing.com/software/pyqt. [Accessed 4 February 2022].

[21] NOAA National Geophysical Data Center. Tatitlek, alaska 8/15 arc-second MHHW coastal digital elevation model. NOAA National Centers for Environmental Information; 2011, https://www.ncei.noaa.gov/metadata/geoportal/rest/metadata/item/gov.noaa.ngdc.mgg.dem:700/html. [Accessed 14 January 2022].

[22] Remy N. S-GeMS: The stanford geostatistical modeling software: A tool for new algorithms development. In: Leuangthong O, Deutsch CV, editors. Geostatistics banff 2004. Dordrecht: Springer Netherlands; 2005, p. 865–71. http://dx.doi.org/10.1007/978-1-4020-3610-1_89.