1 PSOES

The **PSOES** code is written in native MATLAB syntax in the form of a general optimization algorithm. See Table 2 in the manuscript for a complete list of input parameters with definitions and default values (if applied).

In this documentation, usage of the code is demonstrated by solving the Rastrigin problem, a well-known non-convex multi-modal bench mark for various optimization algorithms:

$$f(x) = An + \sum_{i=1}^{n}[x_i^2 - Acos(2\pi x_i)] \quad \text{(D.1)}$$

where A is a real-valued parameter and n corresponds to the dimension of the function in eq. (D.1). The global minimum of the function is located at the origin with f(0,0)=0. However, it is surrounded with at least 8 local minimums in its immediate vicinity which makes its minimization challenging.

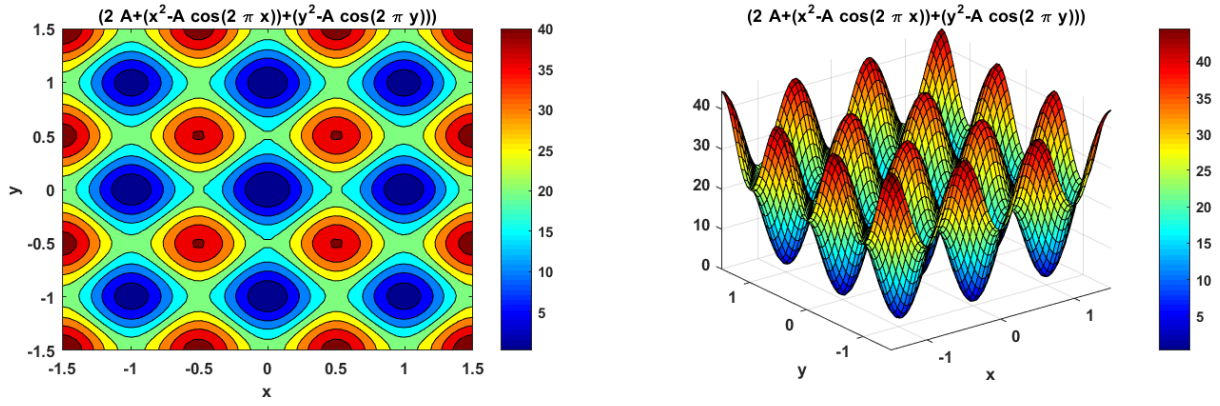


Figure 1- The Rastrigin Function

Minimization of the cost function (e.g. eq. D.1) requires a search space which is defined by the user. The PSOES algorithm would only probe the model space that is confined in the limits of this search space. The function **`SearchSpace()`** is used for producing the necessary input parameters with the correct syntax with respect to the main code. The limits of the search space (**`VarMin`** and **`VarMax`**) are hard coded into this function. Also, the dimension of the model space (**`nVar`**) should be entered therein.

In case of the Rastrigin function, the model space is a two-dimensional plane (e.g. x-y plane) and the search space is defined as a subset of this plane. Here, it is set as a square with side length 20 centered at the origin, that is, $-10 \leq x \leq 10$ and $-10 \leq y \leq 10$. Each particle is therefore a point (x_i, y_i) on this square. In general, for an **`nVar`** dimensional model space, the search space is defined as:

- **`VarMin`**: matrix of size **`1xnVar`**
- **`VarMax`**: matrix of size **`1xnVar`**

With the condition that:

$$\forall \begin{cases} x_i \in \boldsymbol{VarMin} \\ x_j \in \boldsymbol{VarMax} \end{cases} : \quad x_i < x_j \qquad i = j = 1,2, \dots, \boldsymbol{nVar}$$

Along with the **`CostFunction`**, **`VarMin`** and **`VarMax`** are the only required input parameters.

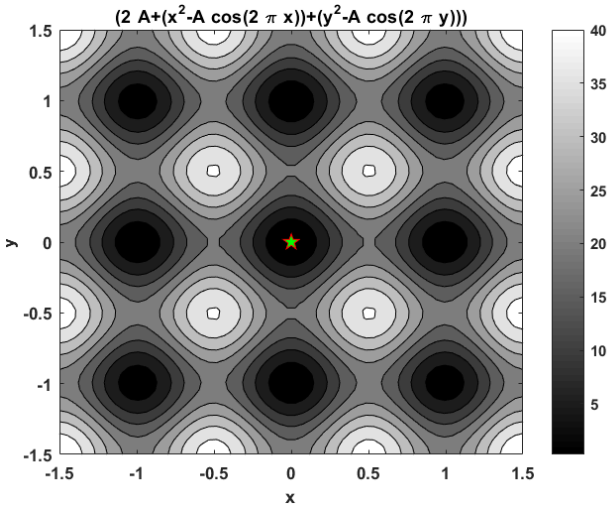
For producing the required input parameters, run:

```
[VarMin, VarMax, nVar, VarSize] = SearchSpace();
```

For the minimization of function CostFunction by **psoes** with predefined inputs as above and default parameters run:

sol = psoes(CostFunction,VarSize,VarMin,VarMax) variable **sol** is a struct holding the global best solution, and its cost (i.e. global best cost)	**(c.1)**
[sol, sol.History] = psoes(CostFunction,VarSize,VarMin,VarMax) Add an optional second output parameter **sol.History** to gets the convergence history of the algorithm.	**(c.2)**
[sol, sol.History, param] = psoes(CostFunction,VarSize,VarMin,VarMax) An optional third output parameter would also get the psoes parameters used in the minimization process.	**(c.3)**

The results of running code (c.3) on the Rastrigin function is presented in Fig. 2 where the star shows the final solution of the algorithm (global best).




```
sol =

Position: [-9.2174e-09 -6.7333e-09]

Cost: 2.4869e-14

ConvHis:[100x1 double]
```

Figure 2: The minimization of the Rastrigin function using PSOES with default parameters

The last output holds the used default parameters.

```
param =

                    c1: 1.5000
                    c2: 2
              c_minus: 0.9700
               c_plus: 1.7500
   ConstrictionFactor: 0
         CostFunction: [function_handle]
                MaxIt: 100
             mutation: 1
            MuteLimit: 5
                 nPop: 25
                 phi1: 2.0500
                 phi2: 2.0500
               VarMax: [10 10]
               VarMin: [-10 -10]
              VarSize: [1 2]
                    w: 1
```

To run the PSO with inertia weight (without mutation) and default parameters, run: `[sol_c4, sol_c4.ConvHis, paramc4] = psoes(CostFunction,VarSize,VarMin,VarMax,'Mutation',false);`	(c.4)
To run PSO with constriction factor and default parameters, run: `[sol_c5, sol_c5.ConvHis,paramc5] = psoes(CostFunction,VarSize,VarMin,VarMax,'Mutation',false, 'ConstrictionFactor',true);`	(c.5)

It should be noted that as a stochastic algorithm, the solutions of PSO upon repeating the minimization process vary due to the involvement of random numbers. The exact results that are presented above may not be reproduced. In order to determine the variability of solution for the three algorithms, (c.3), (c.4), and (c.5) have been repeated 100 times. The distribution of solutions is represented in Fig. 3.

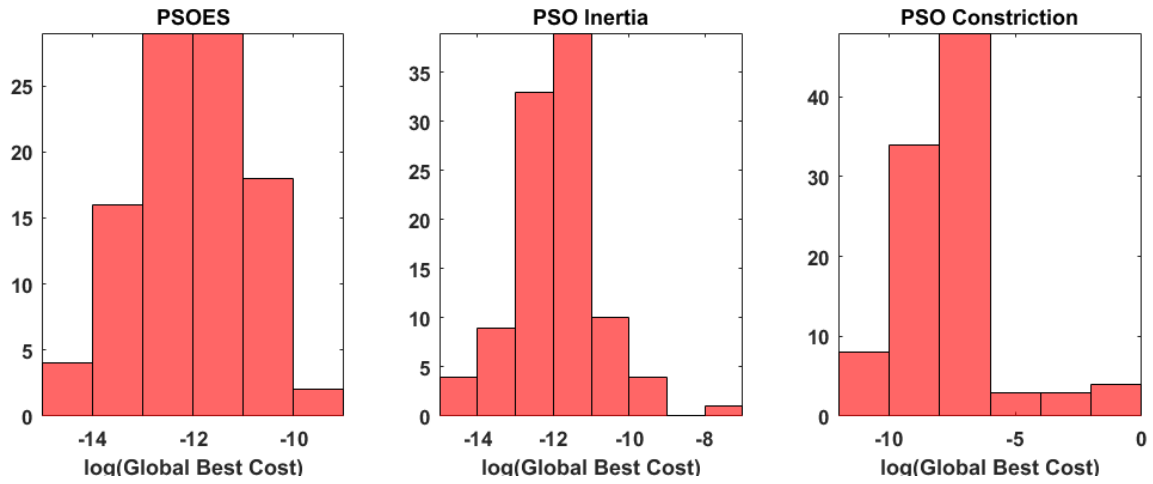


Figure 3: the distribution of the solutions when the runs are repeated 100 times. In this example, the best solution among the 100 runs for PSOES had a cost value of 0, for PSO with inertia 1e-14 and for PSO with constriction 1e-10.

2 Gravity Gradiometry Code: gradGrid3D()

function out=gradGrid3D(prism,geometry,varargin) calculates the gravity gradiometry tensor for either a DEM or a basin. The function uses right rectangular prisms with a linear density gradient to discretize the mass for which the data is to be calculated. See the accompanying paper (Jamasb et al. ***, ***) for formulas.

INPUT PARAMETERS (All Units are in SI)-> Distances in (m); density(kg/m3)

Input parameters of the Gradiometry code:

Input	Description
prism(Required)	a struct holding the coordinates of the prisms
geometry(Required)	a struct holding the lateral
Output (Optional)	Default output is a struct with six feilds for eachgravity gradiometry component - otherwise choose between: {'gxx','gyy','gzz','gxz','gzy','gxy','all'}
'Gamma',value (Optional Pair)	from the top and bottom densities (default) orit can be given directly.

Please note that the order of input follows MATLAB principles:

First enter the Required inputs, then optionals, and finally optional pairs.

out=gradGrid3D(prism,geometry) calculates the full tensor	**(c.6)**
out=gradGrid3D(prism, geometry,'gxx'); calculates only gxx component	**(c.7)**
out=gradGrid3D(prism, geometry,'Gamma',0); calculates the full tensor for a prism with constant density (i.e. Gamma=0)	**(c.8)**
out=gradGrid3D(prism, geometry,'gyz','Gamma',0.02); calculates the gyz component of for a prism with a linear density gradient of 0.02	**(c.9)**

3 A Synthetic Inverse Problem

In this section, a simple synthetic model is used to demonstrate the usefulness of the forward code in interface reconstruction. Also, the workflow of any general inversion using the PSOES algorithm is demonstrated.

Synthetic Model:

In order for the results of this section to be easily reproducible, a 2D (fairly complex) synthetic model is assumed:

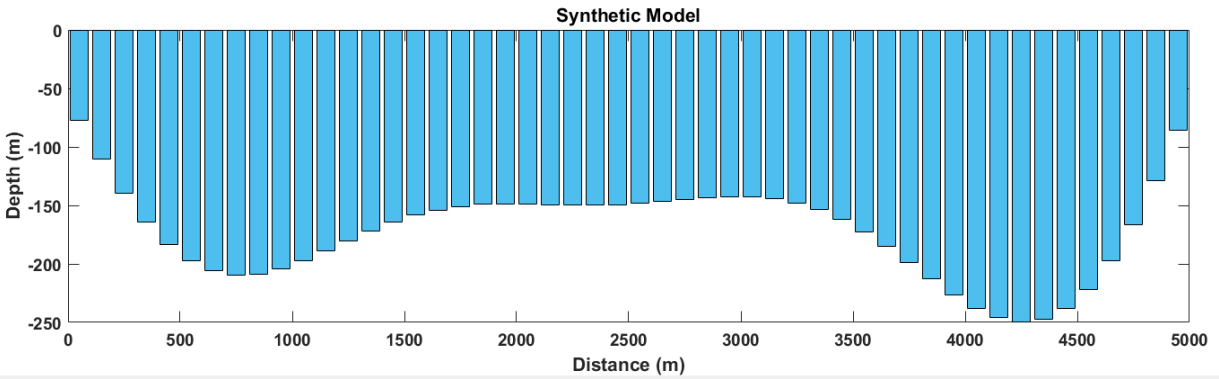


Figure 4: A synthetic basement model

Synthetic Data:

The synthetic gravity data are produced for the synthetic model presented in fig. 4. in the directory "ProduceSyntheticData". After calculation of the data, 3% of additive white Gaussian noise is added to the data. Also, the gradiometry data are calculated for the given model and are presented in fig. 6. Because, we are working on a 2D case, only 3 non-zeros and independent terms remain of the gradiometry tensor. A visual comparison between figures 4 and 5 could easily reveal that the gradiometry data represent the sharp changes (i.e. relatively high frequency effects) better that the gravity data.

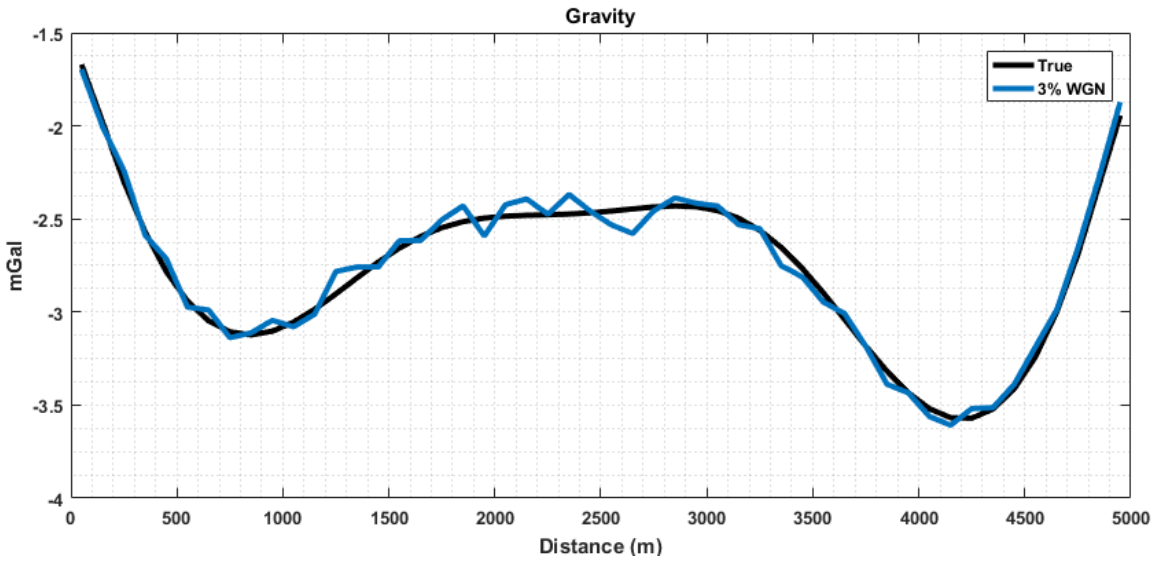


Figure 5: The gravity effect of the synthetic model in Fig. 4.

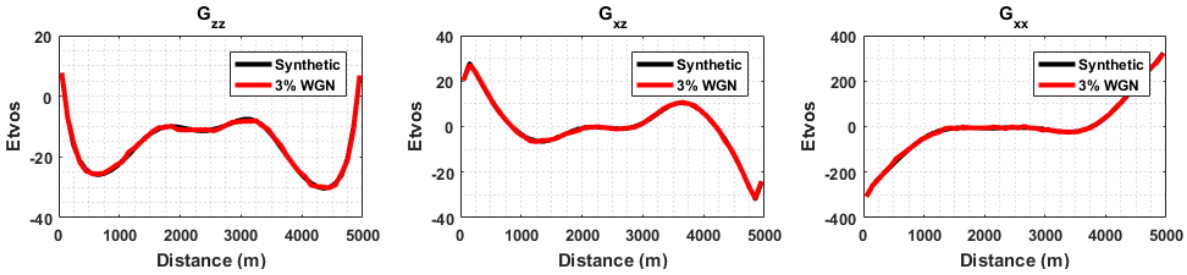


Figure 6: The gradiometry tensor components calculated for the synthetic model in Fig. 4

Inversion:

The details of performing a minimization problem using the psoes is presented in section 1 of this code documentation. Also, the mathematical details of forming the cost function and its considerations can be found in section 3 of the manuscript.

The presented inverse problem aims to invert the g_{zz} component of the gravity gradiometry tensor to reconstruct the depth-to-basement. The total number of unknowns is M (i.e. the number of prisms). See the **`main.m`** script in the main directory of the supplementary material for the inversion.

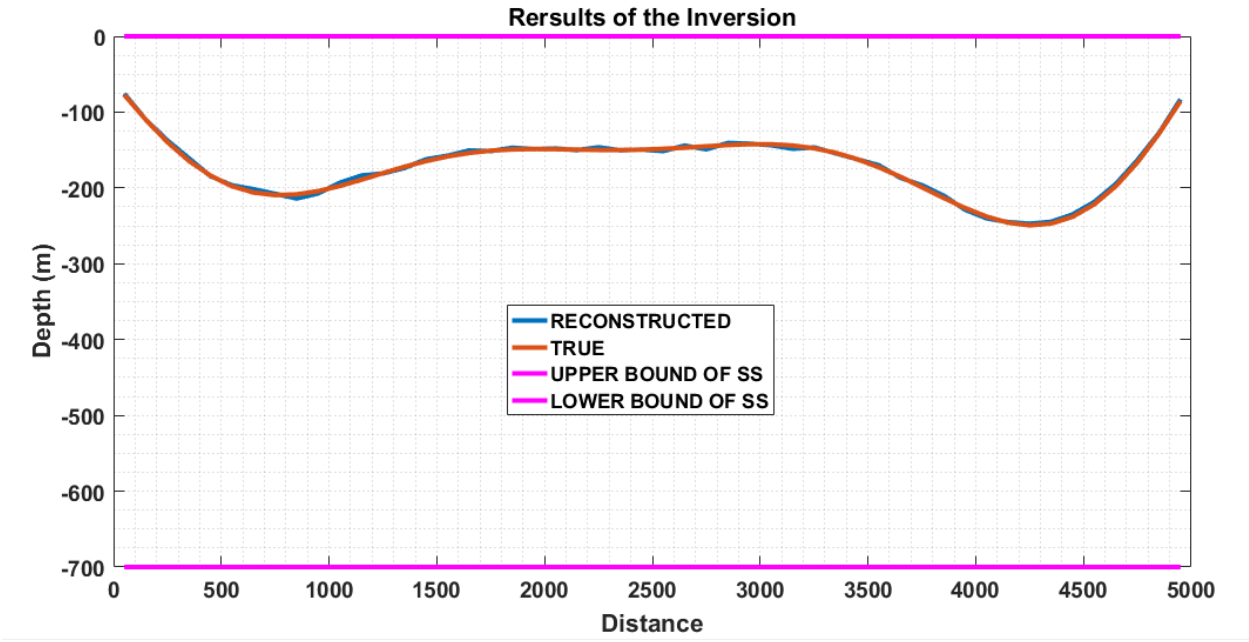


Figure 7: Results of the inversion using psoes with 30 particles and 500 iterations.

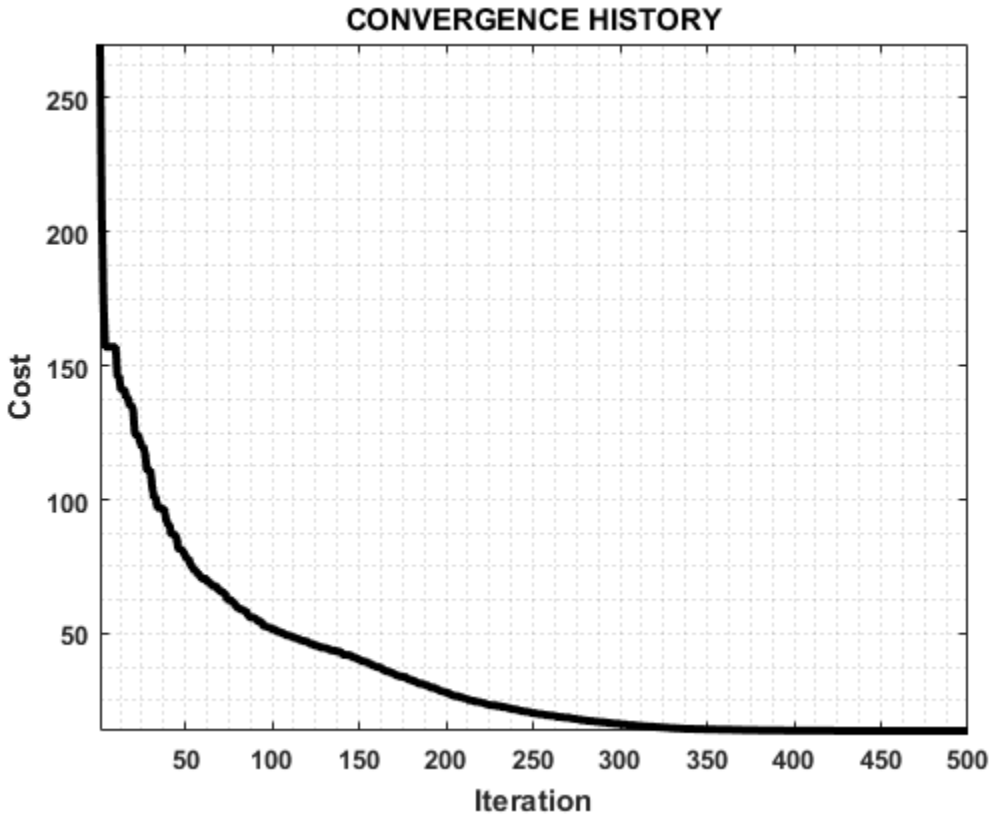


Figure 8: The convergence history of the inversion