The GA-cal software for the automatic calibration of soil constitutive laws: a tutorial and a user manual

Francisco J. Mendez¹*, Miguel A. Mendez², Antonio Pasculli¹
¹University G. D’Annunzio, Dept. of Engineering Geology (INGEO), Chieti-Pescara, Italy;
²von Karman Institute for Fluid Dynamics, EA Department, Sint-Genesius-Rode, Belgium
*Corresponding to: francisco.mendez@unich.it

Abstract

The calibration of an advanced constitutive law for soil is a challenging task. This work describes GA-cal, a Fortran software for automatically calibrating constitutive laws using Genetic Algorithms (GA) optimization. The proposed approach sets the calibration problem as a regression, and the GA optimization is used to adjust the model parameters so that a numerical model matches experimental data. This document provides a user guide and a simple tutorial. We showcase GA-cal on the calibration of the Sand Hypoplastic law proposed by von Wolffersdorff, with the oedometer and triaxial drained test data. The implemented subroutines can be easily extended to solve other regression or optimization problems, including different tests and constitutive models. The source code and the presented tutorial are freely available at https://github.com/FraJoMen/GA-cal.

Keywords Hypoplasticity Model Calibration, Genetic Algorithm Optimization, Nonlinear Regression

1. Introduction

This work describes GA-cal, a Fortran software for automatically calibrating constitutive laws using Genetic Algorithms (GA) optimization. The calibration consist in identifying the model parameters associated with a specific soil. The proposed approach consists of three ingredients: (1) a solver that simulates the laboratory tests of interest for a given set of constitutive model’s parameters, (2) a cost function measuring the performances of the model, i.e. how closely the model predicts the available data from the tests and (3) an optimization algorithm to look for the parameters that give the best prediction (that is a minimum of the cost function).

We here present the calibration of the Sand Hypoplastic law (SH) developed by von Wolffersdorff [12] using the oedometer (OE) and triaxial drained (TD) tests data. This approach was first presented in [8], developed in a Python code and used to calibrate six of the eight coefficients in the SH model. The Fortran code presented in this document optimizes all eight parameters of the SH model and is 30 times faster than the previous Python code. Besides the speed up gained by the porting into Fortran, the new version benefits from a more consistent use of matrix multiplications and an improved approach for generating new individuals. Moreover, its current architecture offers specific subroutines for each of the three ingredients previously described. This allows for extending the proposed approach far beyond the analysed test case, including other constitutive laws and other experimental tests.

This document is organized as follows. In section 2, we briefly recall, in the interest of self-consistency, the SH constitute model currently available in GA-Cal. Section 3 reports on the proposed methodology while section 4 describes the GA-cal code. Section 5 presents a tutorial with an example calibration.

2. The Sand Hypoplasticity (SH) model

The simulation of the OE and TD tests according to the SH model by [12] requires solving a system of nonlinear Ordinary Differential Equations (ODEs). These equations link the time derivative of the Cauchy effective stress $T$ and void ratio $\varepsilon$, with the granulate stretching rate $D$, the void ratio $\varepsilon$ and the Cauchy effective stress $T$.

In axisymmetric conditions such as those encountered in the OE and TD tests, denoting by $T_1, T_2$ and $D_1, D_2$ the principal (axial) and the second (radial) component of $T$ and $D$,

$$T = \begin{bmatrix} T_1 & 0 & 0 \\ 0 & T_2 & 0 \\ 0 & 0 & T_2 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_2 \end{bmatrix},$$

the SH model gives:
The set of equations (9)-(12) thus includes eight parameters, i.e. \( \{ \varphi_c, h_s, n, e_{c0}, \alpha, \beta, e_d0, e_d \} \). The reader is referred to [3, 7, 8] for more details on these parameters.

The GA used in the calibration involves randomly generating different possible combinations of parameters (later referred to as candidate solutions). To be valid, the inequality \( e_{d0} < e_{c0} < e_{d0} \) must be respected at all times. To enforce this, we compute \( e_{d0} \) and \( e_{c0} \) using the ratio previously defined \( \lambda_d < 1 \) and \( \lambda_i > 1 \). Experimentally it has been observed that \( \lambda_d = 0.52 \pm 0.65 \) and \( \lambda_i \approx 1, 2 \) [3]. Therefore, the set of parameters calibrated by the GA is:

\[
P = \{ \varphi_c, h_s, n, e_{c0}, \alpha, \beta, \lambda_d, \lambda_i \} .
\]

### 3. Integration Procedure

The dynamical system (2) is integrated using the simple explicit Euler scheme. This allows for an easy check of the solution admissibility (\( \text{tr}(\mathbf{T}) < 0 \) and \( e_d \leq e \leq e_i \)) at every time step. Defining \( \mathbf{X}^k = [T^k_1, T^k_2, e^k]^T \) the state vector of the ODE system, the time integration scheme reads:

\[
\mathbf{X}^{k+1} = \mathbf{H}(\mathbf{X}^k, \mathbf{P}) \Delta t + \mathbf{X}^k ,
\]

where \( \mathbf{H} \) is the nonlinear function that returns the right term of the (2).

The SH constitutive law is a state-dependent model homogeneous of the first order in \( \mathbf{D} \) [9]. So it is possible to select an arbitrary reference \( D_1 = -1 \) and compute the integration time \( t_f \) from the maximal deformation obtained at the end of each test. For the OE test, the integration time is

\[
t_f = -\ln \left( 1 - \frac{e_{d0} - e_{fin}}{e_{d0} + 1} \right) ,
\]

while for the TD test it is

\[
t_f = e_{fin} .
\]

The parameters \( e_0, e_{fin} \), in (15) are the initial void ratio and the final void ratio, namely at the end of the compression part in the OE test. The parameter \( e_{fin} \) in (16) is the maximum axial deformation in the TD test. Once the integration time \( [0, t_f] \) is defined, the time step is computed as \( \Delta t = t_f/n_{\text{Step}}, \) where the number of time steps can be set to \( n_{\text{Step}} = 100 \). The function \( \mathbf{H}(\mathbf{X}, \mathbf{P}) \) in (14) differs in the two tests, as detailed in [8].

In the Ga-cal software, the function \( \mathbf{H} \) in (2) for the OE and TD tests are computed by the subroutines \( \text{dydtOE} \) and \( \text{dydtTD} \) respectively.

### 4. Calibration Methodology

The proposed calibration procedure combines a stochastic optimizer with a numerical solver of Ordinary Differential Equations (ODEs). For every set of parameters \( \mathbf{P} \), the ODE
The following subsections describe the optimization algorithm implemented in GA-cal, focusing on the cost-function definition in 4.1 and the GA strategy in 4.2.

4.1. Cost-function Definition

The cost-function driving the optimization of the parameters \( \mathbf{P} \) is built to account for the discrepancy between the numerical predictions and the set of measurements [13].

For a given set of parameters \( \mathbf{P} \), the subroutines \texttt{intOE} and \texttt{intTD} are used to simulate the soil response to the OE and TD tests respectively. For each of these tests, these subroutines provide the evolution of the state vector \( \mathbf{X}(t_k) \) in the dynamical system (2). The results are then manipulated prior to the comparison with experimental data. Specifically, the results of the OE tests are provided in the plane \( (\sigma_v, \epsilon) \), where \( \sigma_v = -T_1 \) is the vertical compression pressure; hence we are interested in the temporal evolution of the first and third components of the vector \( \mathbf{X}(t_k) \). The results of the TD test are given in the triaxial deviatoric diagram \( (\epsilon_a, q) \) and triaxial volumetric diagram \( (\epsilon_v, \epsilon_v) \), thus the temporal evolution of the variables of interest are:

- axial deformation
  \[
  \epsilon_a(t) = - \int_0^t D_1 \; dt ;
  \]

- volume change \( \epsilon_v \)
  \[
  \epsilon_v(t) = - \int_0^t \text{tr} (\mathbf{D}) \; dt = \frac{e(t) - e_0}{1 + e_0} ;
  \]

- deviatoric stress
  \[
  q(t) = T_2(t) - T_1(t) .
  \]

These results are projected in dimensionless plane before evaluating the model performances, to cope with the largely different numerical values taken by the variables [8]. The dimensionless variables are computed as follows

\[
\tilde{\epsilon}_a = \frac{\epsilon_a}{\epsilon_{fin}} ;
\]
\[
\tilde{\epsilon}_v = \frac{\epsilon_v}{\max(|\epsilon_v|)} ;
\]
\[
\hat{q} = \frac{q}{\max(q)} ;
\]
\[
\hat{T}_1 = -\frac{\tilde{T}_1}{\max(|\tilde{T}_1|)} ;
\]
\[
\tilde{\epsilon}_E = \ln \left( 1 - \frac{\epsilon_0 - e}{\epsilon_0 + 1} \right) / \ln \left( 1 - \frac{\epsilon_0 - \epsilon_{fin}}{\epsilon_0 + 1} \right) .
\]

The data from the OE tests are mapped to \( \Pi_1 = (\tilde{\epsilon}_E, \hat{T}_1) \), while the data from TD tests are mapped to \( \Pi_2 = (\tilde{\epsilon}_a, \hat{q}) \) and \( \Pi_3 = (\tilde{\epsilon}_a, \tilde{\epsilon}_v) \). As a result of the scaling, the experimental data for each test starts at \((0, 0)\) and end at \((1, 1)\). The scaling is performed in the subroutines \texttt{CostOE} and \texttt{CostTD}.

The discrepancy between model predictions and data is measured in terms of mean of the Discrete Fréchet Distance (DFD). This has the main merit of being invariant to the axis orientation [6]. The DFD is briefly introduced in Figure 1. Given the set of predictions of the SH model \( (X_j, Y_j) \) at times \( t_j \), we denote as \( r_{j,j+1} \) the segment connecting the points \( j \) and \( j + 1 \), so that the set of segments define a piece-wise linear approximation of the numerical response curve. Then, given the experimental point \( A = (X_A, Y_A) \), the DFD is the distance between the point \( A \) and the closest segment, i.e. the one connecting the closest points (C1 and C2 in Figure 1).

Figure 1. Definition of discrete Fréchet distance \( d \) between a points \( A(X_A, Y_A) \) and a numerical curve \( r_{j,j+1} \).

Denoting as \( d_{j,j}^{i,j} \) the vector collecting the DFD of all points on the plane \( \Pi_i \), with \( i = 1, 2, 3 \), for the tests \( j \), we define a performance metrics as

\[
\delta_i = \frac{1}{n_j} \sum_{j=1}^{n_j} \| d_{j,j}^{i,j} \|_2
\]

where \( n_i \) is the number of tests in the i-th plane, \( n_j \) is the number of points for each test and \( \| . \|_2 \) is the \( l_2 \) norm of a vector. This metrics is a function of the parameters \( \mathbf{P} \) used for the model prediction. Therefore, we define the global performance of these parameters as

\[
\text{Cost} (\mathbf{P}) = w_1 \delta_1 (\mathbf{P}) + w_2 \delta_2 (\mathbf{P}) + w_3 \delta_3 (\mathbf{P})
\]

where \( w_i \), with \( i = 1, 2, 3 \), weights the relative importance of each plane \( \Pi_1, \Pi_2, \Pi_3 \). The best set of parameters is the one that minimizes the cost function (26).

It is worth noticing that the cost function in (26) can be easily extended to include results from other tests. Moreover,
one could treat the performances $\delta_i$ in each i-th plane as an independent cost function and thus move from the current single-objective optimization to a multi-objective optimization [4]. This extension is left to future work.

In Ga-cal, the cost function (26) is computed by the subroutine Cost_Pop, while the performances in the three planes are computed by the subroutines Cost_OE and Cost_TD.

4.2. Genetic Algorithm (GA) Optimizer

The GA is a population-based stochastic optimizer which operates on a set (population) of possible candidate solution (here set of parameters $P$). These are evaluated and updated using a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

The optimization thus consists of three steps: (1) initialize, (2) evaluate and (3) update the population. The steps (2) and (3) are repeated in loop until maximum number of iterations $N_{ITER}$ is reached. The best individual $P^*$ is taken from the last update of the population.

The initialization consists in randomly creating a set of candidate solutions. The evaluation consists in running the SH model for each of these and compute the associated cost function according to the previous section. The population update uses a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

This extension is left to future work.

The initialization consists in randomly creating a set of candidate solutions. The initialization consists in running the SH model for each of these and compute the associated cost function according to the previous section. The population update uses a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

This extension is left to future work.

Following the continuous GA in [2], the main mechanism for random number generation is the sampling of a uniform distribution in the range $[0, 1]$. This was carried out using the random_number subroutine of the GNU Fortran compiler. The uniform distribution can then be used to generate other distributions such as the Gaussian and log-normal distributions [10, 11]. In this work, we use it to generate the triangular distribution because it allows for a good compromise between ‘exploration’ (i.e., the tendency to search in unexplored area) and ‘exploitation’ (i.e., the tendency to focus on the known minima) [2].

The triangular distribution, illustrated in Figure 2 is univocally defined by three numbers, i.e., the lower and upper bounds ($L$ and $U$) and the mode ($M$). Given a number $P \in [0, 1]$, sampled from a uniform distribution, the triangularly distributed random variable $x$ reads:

$$
F = \frac{M - L}{U - L}
$$

The triangular distribution, illustrated in Figure 2 is univocally defined by three numbers, i.e., the lower and upper bounds ($L$ and $U$) and the mode ($M$). Given a number $P \in [0, 1]$, sampled from a uniform distribution, the triangularly distributed random variable $x$ reads:

$$
F = \frac{M - L}{U - L}
$$

The triangular distribution, illustrated in Figure 2 is univocally defined by three numbers, i.e., the lower and upper bounds ($L$ and $U$) and the mode ($M$). Given a number $P \in [0, 1]$, sampled from a uniform distribution, the triangularly distributed random variable $x$ reads:

$$
F = \frac{M - L}{U - L}
$$

4.2.1 Random numbers generation

4.2. Genetic Algorithm (GA) Optimizer

The GA is a population-based stochastic optimizer which operates on a set (population) of possible candidate solution (here set of parameters $P$). These are evaluated and updated using a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

The initialization consists in randomly creating a set of candidate solutions. The initialization consists in running the SH model for each of these and compute the associated cost function according to the previous section. The population update uses a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

This extension is left to future work.

Following the continuous GA in [2], the main mechanism for random number generation is the sampling of a uniform distribution in the range $[0, 1]$. This was carried out using the random_number subroutine of the GNU Fortran compiler. The uniform distribution can then be used to generate other distributions such as the Gaussian and log-normal distributions [10, 11]. In this work, we use it to generate the triangular distribution because it allows for a good compromise between ‘exploration’ (i.e., the tendency to search in unexplored area) and ‘exploitation’ (i.e., the tendency to focus on the known minima) [2].

The triangular distribution, illustrated in Figure 2 is univocally defined by three numbers, i.e., the lower and upper bounds ($L$ and $U$) and the mode ($M$). Given a number $P \in [0, 1]$, sampled from a uniform distribution, the triangularly distributed random variable $x$ reads:

$$
F = \frac{M - L}{U - L}
$$

4.2.2 Population Initialization

4.2. Genetic Algorithm (GA) Optimizer

The GA is a population-based stochastic optimizer which operates on a set (population) of possible candidate solution (here set of parameters $P$). These are evaluated and updated using a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

The initialization consists in randomly creating a set of candidate solutions. The initialization consists in running the SH model for each of these and compute the associated cost function according to the previous section. The population update uses a set of statistical operators that mimic the Darwinian theory of survival of the fittest [5].

This extension is left to future work.

Following the continuous GA in [2], the main mechanism for random number generation is the sampling of a uniform distribution in the range $[0, 1]$. This was carried out using the random_number subroutine of the GNU Fortran compiler. The uniform distribution can then be used to generate other distributions such as the Gaussian and log-normal distributions [10, 11]. In this work, we use it to generate the triangular distribution because it allows for a good compromise between ‘exploration’ (i.e., the tendency to search in unexplored area) and ‘exploitation’ (i.e., the tendency to focus on the known minima) [2].

The triangular distribution, illustrated in Figure 2 is univocally defined by three numbers, i.e., the lower and upper bounds ($L$ and $U$) and the mode ($M$). Given a number $P \in [0, 1]$, sampled from a uniform distribution, the triangularly distributed random variable $x$ reads:

$$
F = \frac{M - L}{U - L}
$$

4.2.2 Population Initialization

GA-cal encodes a population of candidates (individuals) in the matrix $\text{Pop}$ of size $(N_i, P_{dim})$, where $N_i$ is the number of individuals, and $P_{dim}$ is the dimension of the optimization problem (in this case $P_{dim} = 8$ is the size of the parameters $P$). Thus each individual is stored as a row of $\text{Pop}$. The population is initialized randomly within the search space bounded by vectors containing the lowest and the largest possible values of each parameter $P_{min}$ and $P_{max}$. This operation is made using a matrix of random numbers $\Theta \in \mathbb{R}^{N_i, P_{dim}}$, uniformly distributed in the range $[0, 1]$:

$$
\text{Pop}(i, j) = P_{min}(j) + \Theta(i,j)(P_{max}(j) - P_{min}(j))
$$

where $i = 1, 2, ..., N_i$ and $j = 1, 2, ..., P_{dim}$.

4.2.3 Population Evaluation

The evaluation procedure consists of two steps. The first step computes the cost-function (26) for each set of parameters, i.e., for each individual. The second step ranks the individuals according to the cost function, used as a fitness measurement, from the best (low cost) to the worst (high cost) candidate solution. The evaluation returns the vector $\text{Class}$, containing the index of first-placed individuals.
4.2.4 Population update

The statistical operations involved in the population update are referred to as \textit{elitism}, \textit{mutation}, \textit{selection} and \textit{cross-over}. These are briefly described in the following.

- \textit{Elitism} consists in advancing some of the best individuals to the next generation. The elite individuals are taken from \texttt{Pop} using the pointer from the first element of the \texttt{Class} vector. Denoting with \(N_{eli}\) the number of elite individuals passed to the next generation we have:

\[
P_{eli} = \text{Pop}(\text{Class}(i))
\]

where \(i = 1, 2, \ldots, N_{eli}\).

- \textit{Mutation} is one of the operations related to the exploration of the parameter space: a randomly chosen portion of the population (\(N_{mut}\)) at each iteration is randomly mutated according to two mechanisms: (1) recombination (\(N_{com}\)) and (2) re-initialization (\(N_{ran}\)).

The recombination consists in combining the entries of the most successful candidates (here referred to as ‘genes’). The selection of the involved entries is carried out using the matrix \(\Psi\), which stores in each column a random vector. Denoting with \(P_{com}^{i,j}\) the number of elite individuals passed to the next generation we have:

\[
P_{com}^{i,j} = \text{Pop}(\text{Class}(\Psi(i,j)),j)
\]

where \(i = 1, 2, \ldots, N_{com}\) and \(j = 1, 2, \ldots, P_{dim}\).

The re-initialization consists in initializing \(N_{ran}\) individuals from scratch, using the same approach used in the initialization of the population.

To balance exploration and exploitation as the optimization progresses, the number of individuals undergoing mutation is progressively reduced. This procedure is akin to the annealing scheduling in simulated annealing. The reduction is defined as

\[
\mu = \mu_{in} \left( \frac{\mu_{en}}{\mu_{in}} \right)^\frac{\text{ITER} - 1}{\text{ITER} - 1}.
\]

where \(\text{ITER}\) is the iteration counter, \(\mu_{in}\) and \(\mu_{en}\) the initial and final ratio of mutation. The evolving number of mutated individuals is thus updated as \(N_{mut} = \mu N_i\), rounded to the nearest digit. Concerning the partitioning into recombined and re-initialized individuals, \(N_{com}\) is linearly increased with the iteration to further promote exploitation:

\[
N_{com} = N_{mut} \frac{\text{ITER}}{N_{ITER}}.
\]

while the number of reinitialised individuals is simply \(N_{ran} = N_{mut} - N_{com}\).

- \textit{selection} and \textit{cross-over} are used to compute the remaining \(N_{mut} = N_i - N_{eli} - N_{mut}\) individuals. The \textit{selection} defines which of the individuals mate while \textit{cross-over} defines how the mating produces new individuals from the selected ones.

The Ga-cal follows the weighting approach in [2]. Two random vectors \(X\) and \(Y\), herein denoted as parents, are sampled from a triangular distribution with \(L = M = 1\) and \(U = N_i\) rounded to the closest integer, where \(n_f\) is the fraction of individuals allowed to mate. Being the population sorted by their fitness, this gives the best individuals a higher chance of being selected. After selection, the \textit{cross-over} is performed by randomly blending genes between the parents as

\[
P_{mat}^{i,j} = \text{Pop}(\text{Class}(X(i,j))) \cdot \Theta(i,j) + \text{Pop}(\text{Class}(Y(i,j))) \cdot (1 - \Theta(i,j)),
\]

where \(i = 1, 2, \ldots, N_{mat}\) and \(j = 1, 2, \ldots, P_{dim}\) and \(\Theta\) is matrix \((N_{mat}, P_{dim})\) of random numbers uniformly distributed in the range \([0, 1]\).

After the three aforementioned operations are carried out, at each iteration the updated population is

\[
\text{Pop}_{New} = P_{eli} \cup P_{com} \cup P_{ran} \cup P_{mat}.
\]

5. The GA-cal code

The GA-cal program is written in Fortran and is structured in a main file \texttt{GA-cal_main.f90} and four modules:

- \texttt{Read_input}
- \texttt{Create_Pop}
- \texttt{Evaluate_Pop}
- \texttt{Write_out}

The code can be compiled using \texttt{gfortran}\footnote{https://gcc.gnu.org/fortran/}. The \texttt{Code::Blocks}\footnote{http://www.codeblocks.org} project attached to the source code allows to set up the compilation with all modules. Once the compilation is completed, the executable should be copied in a folder containing a subfolder named \texttt{data} where all the inputs files, described in the following, should be collected.

Running the executable produces a prompt message requesting the path and the name of an input file \texttt{'data/file}
The program’s input and output are briefly described in the 5.1 and 5.2, while the subsection from 5.3 to 5.6 provides a brief description of the GA-cal modules. The reader is referred to the comments in the code for more details.

5.1. Input files

The input data required in the subfolder data consists of seven text files.

**Input.txt** containing (1) the number of OE tests and TD tests, (2) the number of steps $n_{\text{step}}$ in the time integration of the constitutive law (14), (3) the weights for the equation (26) and (4) the names of the files that the program will read.

The structure of this file is:
```
# Parameter's GA regression problem
number_individual_N_i,........
problem_dimension_P_dim,8
number_of_iteration_N_ITER,......
number_of_Elite_N_E,........
rat_allowed_to_mate_n_f,......
coeff_mutation_fin_mu_en,....
coeff_mutation_ini_mu_in,....
```

**OE_init.txt** contains the initial conditions of each OE test, in the form of a vector. The length of this vector must be consistent with the number of data reported in the Input.txt file. The required initial conditions are (1) the principal (axial) and the second (radial) component of the Cauchy effective stress $T_1$ and $T_2$ (positive if the sample is compressed), and (2) the void ratio $e$.

The structure of this file is:
```
#_Oedometric_initial_condition_tests________
#_T1(kPa)______T2(kPa)________e_(-)_____________
....    ....    ....
....    ....    ....
....    ....    ....
```

**TD_init.txt** contains the initial conditions of each TD test. As for the previous file, these must be compatible with the number of data indicated in the Input.txt file. The initial conditions to prescribe are organised as if the previous file.

The structure of this file is thus identical to the previous one:
```
#_Triaxial_initial_condition_tests________
#_T1(kPa)______T2(kPa)________e_(-)_____________
....    ....    ....
....    ....    ....
....    ....    ....
```

**OE_data.txt** contains the experimental data collected from the OE tests. Specifically, proceeding in column order, these are (1) the void ratio $e$ (2) the vertical stress $\sigma_v = -T_1$ (3) the identification number for the test $id$, ordered in ascending order.

The structure of this file is:
```
#_Oedometric_experimental_data
#_e_(-)_________sigv_(MPa)________id_(-)____
.... .... ....
.... .... ....
.... .... ....
```

**TD_data.txt** contains the experimental data from the TD tests. In column order, these are (1) the axial deformation $\varepsilon_a$ (see (17)) (positive for the compression), (2) the volumetric deformation $\varepsilon_v$ (see (18)) (positive for the compression), (3) the deviatoric stress $q$ (see (19)) and (4) the identification number for the test $id$, ordered in ascending order.

The structure of this file is:
5.2. Output files

At the end of each execution, the program writes the following files:

- **log.Pop.txt** is a text file containing all combinations of parameters evaluated in the last run. This file is updated at each execution of GA-Cal.
- **best_fit.txt** is a text file containing the optimal combination of $P$ after each run.
- **data_structure.dat** is a binary file with the information required to encode the following binary file in output.
- **X_OE.dat** is a binary file collecting the coordinates of the experimental points of all the processed OE tests (in the plane $\sigma_v, e$).
- **X_TD.dat** is a binary file collecting the coordinates of the experimental points in the plan $(\varepsilon_a, q)$ and $(\varepsilon_e, \varepsilon_v)$.
- **HX_TD.dat** is a binary file collecting the response curves obtained with the last $Pop$ individuals in the planes $(\varepsilon_a, q)$ and $(\varepsilon_e, \varepsilon_v)$.
- **HX_OE.dat** is a binary file collecting the response curves obtained with the last $Pop$ individuals in the odometer plane $(\sigma_v, e)$.

5.3. Read_input

**Read_input** is the first module executed by the program. This reads the input files in this module, and defines all common variables. The code requires the input file name specification, from which the program obtains all the information necessary for the calibration.

5.4. Create_Pop

The Create_Pop module contains two subroutines; one for creating the initial population and one for updating it.

- **Initial_Pop** computes the initial population by continuous uniform distribution between the Lower and Upper values of each model parameter defined in the input.
- **Update_Pop** uses elitism, mutation, selection and cross-over to generate a new population at each iteration.

5.5. Evaluate_Pop

This module contains eight subroutines for evaluating the cost-function and one for reordering the population and getting the vector $Class$.

- **COST_OE** compute the cost-function contribution in the oedometer plane.
- **COST_TD** compute the cost-function contribution in the triaxial planes.
- **COST_Pop** combines the computed errors with the subroutines **COST_OE** and **COST_TD** for the definition of the Cost assigned to each individual of the population. The higher it is, the greater the deviation of the individual from the experimental data.
- **dydtEDO** compute the time derivative of the ODE system of oedometer test.
- **dydtTxD.** compute the time derivative of the ODE system of triaxial drained test.
- **intE_OE** perform the explicit Euler integration to simulate oedometer test using dydtEDO subroutine.
- **intE_TD** perform the explicit Euler integration to simulate triaxial drained test using dydtTxD subroutine.
- **elab_TD** perform the elaboration of the triaxial drained test response and from $(T_1 [MPa], T_2 [MPa], e[-])$ compute $(q [MPa], \varepsilon_a [\%], \varepsilon_v [\%])$
- **merge_argsort** returns a vector of indices that sorts the input vector in ascending order.

In addition to some specific functions used to integrate the constitutive model selected, this module contains the function **Frechet_dist** to compute the averaged Fréchet distances between the experimental points and the response curves numerically obtained.

5.6. Write_out

The Write_out module writes all the output files described in the subsection 5.2 expect for the files *log.Pop.txt*, written by the subroutines Create_Pop, and *best_fit.txt* file, written by the GA-cal main.

6. An Example of Calibration

We here illustrate an example of GA-calibration of the SH model using the experimental data obtained from the two OE and the three TD tests reported in [12]. The selected experimental data, with all the input files used, can be found in the example folder in the repository of the source code.

Launching the executable produces the command line interface in (CLI) in Figure 3a, where the input file should be
specified. Once again, all input files should be in a subfolder named data on the same path from which the executable is launched.

During execution, the program writes a series of information to the CLI which can be divided into three stages:

1. **Summary of input files.** The first lines of the CLI show all the data related to the program input files described in the previous section. The output on the CLI is here not shown for brevity.

2. **The evolution of the GA optimization** (see Figure 3b). Starting from the initial population, the first line displays the iteration counter while the others report some relevant information concerning the model performances in the three previously described planes. For each of these, the table reports the maximum and minimum values of the deviations δ_t (see eq. (25)), that is delta OE, delta TD_q and delta TD_e. The best score is also printed; this is the minimum value of the cost function (see eq. (26)). These outputs allow for evaluating the optimizer’s convergence during the iterations. In general, if one seeks a calibration that does not favour any particular plane, it suffices using weights of comparable orders of magnitude.

The user can control the weights w_i in the eq. (26), by editing the Input.txt files, if the error in one particular plane is overly large or if a specific plane needs to be prioritized. This might occur, for example, in the stability analysis of a slope, for which the behaviour in the plane (ε_a, q) is far more important than the behaviour in the plane (σ_e, ε);

3. **The summary from the optimization** (see Figure 3c). This includes three tables. The first table reports on the performances of the best ten candidate solutions and their associated cost function and ID in the last iteration. The candidate ID is simply the row number of the Pop matrix.

Note that the ID is assigned after each population update, while the ranking is carried out after each population evaluation. In the specific example shown in Figure 3c, in which five elites are chosen at each iteration, four candidates (IDs 239, 301, 239, 497) have performed better than the fourth elite. Therefore, their ID would change in the following population update. However, the algorithm closes the last iteration with an evaluation (not an update) hence the mismatch between ranking and IDs, which helps understand the degree to which the best candidates have converged.

The second table illustrate the parameters of the best five candidates (in this case IDs 1, 2, 3, 239, 301). The closeness of these parameters proves the convergence of the optimizer, as the population cannot settle over different

---

Figure 3. The command-line interface of the GA-cal in three different stages. (a) Input part with the highlighting box of the input file specification, (b) convergence check, (c) final tables output.
local minima if a sufficiently large number of iterations is performed unless these minima have the same (to machine precision) value of the cost function. The occurrence of such a highly unlike event can nevertheless be spotted by analyzing this table together with the first.

In case of large variation of the parameters, the user should thus change the setting of the optimizer in the GA_setup.txt (increasing the number of iterations, number of individuals, and/or decreasing the mutation factors). Finally, the last table shows the values of the parameter optimized (Best fit), while the last lines show the calculation time.

The results from the tutorial calibration are reported in Table 1. Being the GA a stochastic optimizer, these values might differ at each run; however, the results for this problem should not differ much and are indeed similar to those reported in the literature for this test case. In an ongoing work by the authors [?], the intrinsic variability of the GA is used to analyze the uncertainty in the model prediction (see also [14]) and the problem of parameter uniqueness.

The plots in Figure 4, obtained with the script PyOut.py attached to the code documentation, show the excellent agreement of the calibrated SH model with the experimental data. In terms of cost function and the response curve of the model, all the combinations suggested by GA-cal with different executions are practically identical when the optimization parameters allow convergence. Moreover, as reported in [8], correlations can emerge from the parameters obtained from different calibrations.

Table 1. SH model parameter for the Hochstetten sand from von Wolffersdorff (W) [12], Herle and Gudehus (H) [3], Mendez et al. (M) [8] and GA-cal (GA-cal).

| Par.     | W   | H   | M   | GA-cal |
|----------|-----|-----|-----|--------|
| φ (°)    | 33.00 | 33.00 | 32.73 | 33.52  |
| $h_s$ (GPa) | 1.00  | 1.50  | 1.32 | 2.22  |
| n       | 0.25 | 0.28 | 0.23 | 0.22 |
| $e_{\varepsilon_0}$ | 0.55 | 0.55 | 0.60 | 0.58 |
| $e_{\varepsilon_{\varepsilon_0}}$ | 0.95 | 0.95 | 1.04 | 1.03 |
| $e_{\varepsilon_{d0}}$ | 1.05 | 1.05 | 1.14 | 1.16 |
| α       | 0.25 | 0.25 | 0.23 | 0.21 |
| β       | 1.50 | 1.00 | 1.26 | 1.12 |

Acknowledgements

The authors gratefully acknowledge the support and the discussions with the engineer Pierantonio Cascioli, from GEINA srl, and Gabriele Sandro Toro, laboratory technician of the Department of Engineering and Geology of the Faculty Gabriele D’Annunzio of Chieti.

References

[1] E. Bauer. Calibration of a comprehensive hypoplastic model for granular materials. Soils and foundations, 36(1):13–26, 3 1996.
[2] Randy L. Haupt and Sue Ellen Haupt. Practical Genetic Algorithms. John Wiley & Sons, Inc., may 2003.
[3] I. Herle and G. Gudehus. Determination of parameters of a hypoplastic constitutive model from properties of grain assemblies. Mechanics of Cohesive-frictional Materials, 41Issue
[4] Tomoyuki Hiroyasu, Mitsunori Miki, Jiro Kamiura, Shinya Watanabe, and Hiro Hiroyasu. MOGADDS: Multi-Objective Genetic Algorithm with Distributed Environment Scheme, chapter 9, pages 201–227. Springer London, London, 2005.

[5] John H. Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press, 04 1992.

[6] Charles F. Jekel, Gerhard Venter, Martin P. Venter, Nielen Stander, and Raphael T. Hafika. Similarity measures for identifying material parameters from hysteresis loops using inverse analysis. International Journal of Material Forming, 12(3):355–378, Jul 2018.

[7] D. Mašín. The influence of experimental and sampling uncertainties on the probability of unsatisfactory performance in geotechnical applications. Géotechnique, 65:897–910, 2015.

[8] Francisco José Mendez, Antonio Pasculli, Miguel Alfonso Mendez, and Nicola Sciarra. Calibration of a hypoplastic model using genetic algorithms. Acta Geotechnica, 16(7):2031–2047, Jul 2021.

[9] A. Niemunis. Extended hypoplastic models for soils. Dissertation, Ruhr University Bochum, Germany, July 2003.

[10] A. Pasculli, M. Calista, and N. Sciarra. Variability of local stress states resulting from the application of monte carlo and finite difference methods to the stability study of a selected slope. Engineering Geology, 245:370 – 389, 2018.

[11] Antonio Pasculli and Massimo Mangifesta. Local seismic response change following an excavation in an area located in the territory of l’aquila city (italy). AIP Conference Proceedings, 2293(1):420072, 2020.

[12] P.-A. von Wolffersdorff. A hypoplastic relation for granular materials with a predefined limit state surface. Mechanics of Cohesive-frictional Materials, 1(3):251–271, 1996.

[13] Zhen-Yu Yin, Yin-Fu Jin, Jack Shuilong Shen, and Pierre-Yves Hicher. Optimization techniques for identifying soil parameters in geotechnical engineering: Comparative study and enhancement. International Journal for Numerical and Analytical Methods in Geomechanics, 42(1):70–94, 2018.

[14] Xinyu Zhao, Joshua Schorr, Andrés Alfonso Peña Olarte, and Roberto Cudmani. A step towards quantifying the uncertainty of the soil mechanical response through the use of genetic algorithms. In Marco Barla, Alice Di Donna, Donatella Sterpi, and Alessandra Insana, editors, Challenges and Innovations in Geomechanics, pages 263–271, Cham, 2023. Springer International Publishing.