ALAMOPY user manual and installation guide v. 2019.6.27

June 27, 2019

For information about this software, contact Marissa Engle at `mengle@andrew.cmu.edu`.

Contents

1 Introduction . . . 1
1.1 Licensing and software requirements . . . 2
1.2 Installation . . . 2
2 Algorithms implemented in ALAMO . . . 2
3 Starting out with ALAMOPY . . . 2
3.1 Data Arguments . . . 3
3.2 Available Basis Functions . . . 3
3.3 ALAMO Regression Options . . . 4
3.4 Validation Capabilities . . . 4
3.5 File Options . . . 5
4 Example doalamo: camel6.py . . . 5
5 ALAMOPY results dictionary . . . 6
5.1 Output models . . . 6
5.2 Fitness metrics . . . 6
5.3 Regression description . . . 6
5.4 Performance specs . . . 7
6 Advanced user options . . . 7
6.1 Custom Basis Functions . . . 7
6.2 Custom Constraints . . . 8
6.3 Basis Function Groups and Constraints . . . 8
7 Additional Capabilities of ALAMOPY . . . 9
7.1 almplot . . . 9
7.2 almconfidence . . . 9
8 Troubleshooting ALAMOPY . . . 10

8.1 Python packages 10
8.2 Can't read tracefile 10
8.3 Information from the python files 10

1 Introduction

The purpose of ALAMOPY (Automatic Learning of Algebraic MOdels PYthon wrapper) is to provide a wrapper for the software ALAMO which generates algebraic surrogate models of black-box systems for which a simulator or experimental setup is available. Consider a system for which the outputs z are an unknown function f of the system inputs x. The software identifies a function f, i.e., a relationship between the inputs and outputs of the system, that best matches data (pairs of x and corresponding z values) that are collected via simulation or experimentation. ALAMO can:

- build an algebraic model of a simulation or experimental black-box system
- use previously collected data for model building
- call a user-specified (simulation) function to collect measurements
- enforce response variable bounds, physical limits, and boundary conditions
- use a preexisting data set for model validation
- output models in simple algebraic form

The problems addressed by the software have long been studied in the fields of statistics, design of experiments, and machine learning. Whereas existing techniques from this literature can be used to fit data to models, the main challenges addressed by the software are in determining where to run the simulations or experiments, what models to fit, and how to determine if the model is accurate and as simple as possible. A distinguishing feature of the software is that it provides models that are accurate yet as simple as possible. Moreover, ALAMO is capable of utilizing theory-driven insights alongside data. The ALAMO models can be used to facilitate subsequent system analysis, optimization, and decision making.

ALAMOPY allows access to python packages as well as the capabilities provided by ALAMO. The wrapper provides methods to interact with ALAMO, streamline the customized constraints and basis functions, output an importable python model, and analyze results with python specific packages.

1.1 Licensing and software requirements

The ALAMOPY code is provided under the IDAES framework and is available for download with the idaes tool kit at `https://www.idaes.org`. ALAMO is required to run ALAMOPY regressions. The ALAMO code is available for download at `http://minlp.com/alamo`. The same URL provides information about licensing the software and any software dependencies.

1.2 Installation

Install ALAMOPY to your distribution of python. ALAMOPY is compatible with python 2 and 3. Navigate to the `alamo_python` directory containing the `setup.py` script and run the python installation from there. This will allow you to import the `alamopy` python package to python scripts.

```
    pip install .
```

or

```
    python setup.py install
```

Required python packages to run ALAMOPY include `numpy`, `sympy`, `scipy`, `matplotlib`, and `pandas`, and are listed in `requirements.txt`.

Install ALAMO and the ALAMO license file in any directory of your choice and add it to your path. Do the same for any ALAMO dependencies (currently, the only dependency is GAMS and this is optional).

2 Algorithms implemented in ALAMO

Refer to the ALAMO manual for details.

3 Starting out with ALAMOPY

ALAMOPY's main function is `alamo`. Data can be read in or simulated using available python packages. The main arguments of the alamo python function are inputs and outputs, which are 2D arrays of data. For example

```
regression_results = alamo(x_inputs, z_outputs, **kargs)
```

where ****kargs** is a set of named keyword arguments than can be passed to the alamo python function to customize the basis function set, names of output files, and other options available in ALAMO.

3.1 Data Arguments

- **xmin**, **xmax**: minimum/maximum values of inputs, if not given they are calculated
- **zmin**, **zmax**: minimum/maximum values of outputs, if not given they are calculated
- **xlabels**: user-specified labels given to the inputs
- **zlabels**: user-specified labels given to the outputs

```
alamo(x_inputs, z_outputs, xlabels=['x1','x2'], zlabels=['z1','z2'])
alamo(x_inputs, z_outputs, xmin=(-5,0),xmax=(10,15))
```

3.2 Available Basis Functions

- **linfcns, expfcns, logfcns, sinfcns, cosfcns**: 0-1 option to include linear, exponential, logarithmic, sine, and cosine transformations. For example

  ```
  linfcns = 1, expfcns = 1, logfcns = 1, sinfcns = 1, cosfcns = 1
  ```

  This results in basis functions = x_1, $\exp(x_1)$, $\log(x_1)$, $\sin(x_1)$, $\cos(x_1)$

- **monomialpower, multi2power, multi3power**: list of monomial, binomial, and trinomial powers. For example

  ```
  monomialpower = (2,3,4), multi2power = (1,2,3), multi3power(1,2,3)
  ```

  This results in the following basis functions:

  Monomial functions = x^2, x^3, x^4
  Binomial functions = (x_1x_2), $(x_1x_2)^2$, $(x_1x_2)^3$
  Trinomial functions = $(x_1x_2x_3)$, $(x_1x_2x_3)^2$, $(x_1x_2x_3)^3$

- **ratiopower**: list of ratio powers. For example

  ```
  ratiopower = (1,2,3)
  ```

  This results in basis functions = (x_1/x_2), $(x_1/x_2)^2$, $(x_1/x_2)^3$

```
alamo(x_inputs, z_outputs, linfcns=1, logfcns=1, expfcns=1)
alamo(x_inputs, z_outputs, linfcns=1, multi2power=(2,3))
```

Note: Custom basis functions are discussed in the Advanced User Section.

3.3 ALAMO Regression Options

- **showalm**: print ALAMO output to the screen
- **expandoutput**: add a key to the output dictionary for multiple outputs
- **solvemip, builder, linearerror**: A $0-1$ indicator to solve with an optimizer (GAMS-SOLVER), use a greedy heuristic, or use a linear objective instead of squared error.
- **modeler**: Fitness metric to beused for model building (1-8)
  - 1. **BIC**: Bayesian infromation criterion

  - 2. **Cp**: Mallow's Cp
  - 3. **AICc**: the corrected Akaike's information criterion
  - 4. **HQC**: the Hannan-Quinn information criterion
  - 5. **MSE**: mean square error
  - 6. **SSEp**: sum of square error plus a penalty proportional to the model size (Note: convpen is the weight of the penalty)
  - 7. **RIC**: the risk information criterion
  - 8. **MADp**: the maximum absolute eviation plus a penalty proportional to model size (Note: convpen is the weight of the penalty)

- **regularizer**: Regularization method used to reduce the number of potential basis functions before optimization of the selected fitness metric. Possible values are 0 and 1, corresponding to no regularization and regularization with the lasso, respectively.
- **maxterms**: Maximum number of terms to be fit in the model
- **convpen**: When MODELER is set to 6 or 8 the size of the model is weighted by CONVPEN.
- **almopt**: name of the alamo option file
- **simulator**: a python function to be used as a simulator for ALAMO, a variable that is a python function (not a string)
- **maxiter**: max iteration of runs

3.4 Validation Capabilities

- **xval, zval**: validation input/output variables
- **loo**: leave-one-out evaluation
- **lmo**: leave-many-out evaluation
- **cvfun**: cross-validation function (True/False)

3.5 File Options

- **almname**: specify a name for the .alm file
- **savescratch**: saves .alm and .lst
- **savetrace**: saves tracefile
- **saveopt**: save .opt options file
- **savegams**: save the .gms gams file

4 Example doalamo: camel6.py

The following file is referred to as ‘camel6.py’ and pertains to learning the six hump camel function. There are two inputs and one output in the model. An initial sampling data set is specified and is comprised of 10 sampled data points.

```
import alamopy

#Import additional python modules for creating the synthetic data
import math
import examples
import numpy as np

def main():
    # Specify number of points to be used in the training set
    # Validation data can be provided optionally
    ndata=10
    x = np.random.uniform([-2,-1],[2,1],(ndata,2))
    z = [0]*ndata

    # specify simulator as examples.sixcamel
    sim = examples.sixcamel
    for i in range(ndata):
        z[i] = sim(x[i][0],x[i][1])

    # Use alamopy’s python function wrapper to avoid using ALAMO’s I/O format
    almsim = alamopy.wrapwriter(sim)
    # Call alamo through the alamopy wrapper
    res = alamopy.alamo(x,z,almname=’cam6’, logfcns = 1, monomialpower=(2,3),
    multi2power=(1,2), simulator=almsim, expandoutput=True, maxiter=20)

    # Display results
    print(’Model: {}’.format(res[’model’]))

if __name__ == ’__main__’:
    np.random.seed(100)
    main()
```

Several additional examples of ALAMOPY input files accompany the distributed code.

5 ALAMOPY results dictionary

The results from alamopy.alamo are returned as a python dictionary. The data can be accessed by using the dictionary keys listed below. For example

```
regression_results = doalamo(x_input, z_output, **kargs)
model = regression_results['model']
```

5.1 Output models

- **f(model)**: A callable function
- **pymodel**: name of the python model written
- **model**: string of the regressed model

Note: A python script named after the output variables is written to the current directory. The model can be imported and used for further evaluation, for example to evaluate residuals:

```
import z1
residuals = [y-z1.f(inputs[0],inputs[1]) for y,inputs in zip(z,x)]
```

5.2 Fitness metrics

- **size**: number of terms chosen in the regression
- **R2**: R2 value of the regression
- Objective value metrics: **ssr, rmse, madp**

5.3 Regression description

- **version**: Version of ALAMO
- **xlabels, zlabels**: The labels used for the inputs/outputs
- **xdata, zdata**: array of xdata/zdata
- **ninputs, nbas**: number of inputs/basis functions

5.4 Performance specs

There are three types of regression problems that are used: ordinary linear regression (olr), classic linear regression (clr), and a mixed integer program (mip). Performance metrics include the number of each problems and the time spent on each type of problem. Additionally, the time spent on other operations and the total time are included.

- **numolr, olrtime, numclr, clrtime, nummip, miptime**: number of type of regression problems solved and time
- **othertime**: Time spent on other operations
- **totaltime**: Total time spent on the regression

6 Advanced user options

Similar to ALAMO, there are advanced capabilities for customization and constrained regression facilitated by methods in ALAMOPY including custom basis functions, custom constraints on the response surface, and basis function groups. These methods interact with the regression using the alamo option file.

6.1 Custom Basis Functions

Custom basis functions can be added to the built-in functions to expand the functional forms available. In ALAMO, this can be done with the following syntax

```
NCUSTOMBAS #
BEGIN_CUSTOMBAS
```

e.g. $x_1^2 x_2^2$

$\vdots$

```
END_CUSTOMBAS
```

To use this advanced capability in ALAMOPY, the following function is called. Note it is necessary to use the xlabels assigned to the input parameters.

```
addCustomFunctions(fcn_list)
addCustomFunctions(["x1^2 * x2^2", "...", "..." ...])
```

6.2 Custom Constraints

Custom constraints can be placed on response surface or regressed function of the output variable. In ALAMO, this is controlled using custom constraints, CUSTOMCON. The constraints, a function $g(x\_inputs, z\_outputs)$ are applied to a specific output variable, which is the index of the output variable, and are less than or equal to 0 ($g \leq 0$).

```
CRNCUSTOM #
BEGIN_CUSTOMCON
```

e.g. $1\ z_1 - x_1 + x_2 + 1$
$\vdots$

```
END_CUSTOMCON
```

To use this advanced capability in ALAMOPY, the following function is called. Note it is necessary to use the xlabels assigned to the input parameters.

```
addCustomConstraints(custom_constraint_list, **kargs)
addCustomConstraints(["1 z_1 - x_1 + x_2 +1", "...", "..." ...])
```

6.3 Basis Function Groups and Constraints

In addition to imposing constraints on the response surface it produces, ALAMO has the ability to enforce constraints on groups of selected basis functions. This can be accomplished using NGROUPS and identifying groups of basis functions. For ALAMO, this is achieved by first defining the groups with

```
NGROUPS 3
BEGIN_GROUPS
# Group-id Member-type Member-indices <Powers>
1 LIN 1 2
2 MONO 1 2
3 GRP 1 2
END_GROUPS
```

To add groups to ALAMOPY, you can use the following methods. Each Basis group has an index number that will be used as reference in the group constraints. The groups are defined by three or four parameters. Options for Member-type are LIN, LOG, EXP, SIN, COS, MONO, MULTI2, MULTI3, RATIO, GRP, RBF, and CUST.

```
addBasisGroup(type_of_function, input_indices, powers)
addBasisGroups(groups)

addBasisGroup("MONO", "1", "2")
addBasisGroups([["LIN","1 2"],["MONO","1","2"],["GRP","1 2"]])
```

With the groups defined, constraints can be placed on the groups using the constraint-types NMT (no-more-than), ATL (at-least), REQ (requires), and XCL (exclude). For NMT and ATL the integer-parameter is the number of members in the group that should be selected based on the constraint. For REQ and XCL the integer-parameter is the group-id number of excluded or required basis functions.

```
BEGIN_GROUPCON
Group-id Output-id Constraint-type Integer-parameter
3 1 NMT 1
END_GROUPCON
```

To add the basis constraints to alamopy, you can use the following methods.

```
addBasisConstraint(group_id, output_id, constraint_type, intParam)
addBasisConstraints(groups_constraint_list)

addBasisConstraint(3,1,NMT,1)
addBasisConstraints([[3,1,"NMT",1]])
```

7 Additional Capabilities of ALAMOPY

There are additional capabilities available in ALAMOPY using the available python packages such as ALMPLOT and ALMCONFIDENCE.

7.1 almplot

The plotting capabilities of ALAMOPY are available in the almplot function. Almplot will plot the function based on one of the inputs.

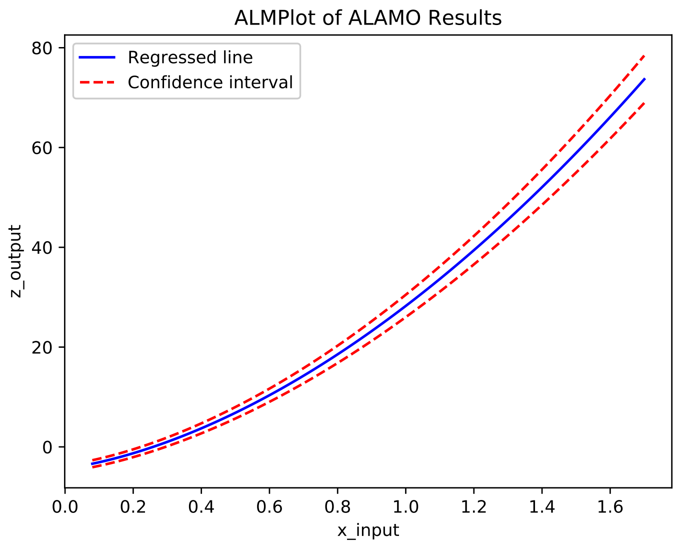
```
result = alamopy.alamo(x_in, z_out, kargs)
alamopy.almplot(result)
```

7.2 almconfidence

Confidence intervals can similarly be calculated for the weighting of selected basis functions.

This adds **conf_inv** (confidence intervals) and **covariance** (covariance matrix) to the results dictionary. This also gets incorporated into the plotting function if it is available.

```
result = alamopy.alamo(x_in, z_out, kargs)
result = alamopy.almconfidence(result)
alamopy.almplot(result)
```



8 Troubleshooting ALAMOPY

ALAMOPY is an open software package for Python, so the python code is visible by the user. If the error you are seeing is not covered in this section, either check the code to see what might be causing the error or email the code or the error message for help.

8.1 Python packages

All of the used python packages should be installed with the installation of ALAMOPY. If a package is missed or different versions exist, there can be some errors.

- Sympy package: Error with lamdify
- Matplotlib package: Error with plotting

8.2 Can't read tracefile

If the tracefile cannot be read, it is likely that alamo did not run. Run alamopy.alamo with the keyword argument showalm and savealm (showalm=True, savealm=True). Run the .alm file using alamo directly. The error can be with calling 'alamo' on the file or it could be in the .alm file itself.

8.3 Information from the python files

Since this is open software, users can see the python code for alamopy. For a deep dive into the code, the key files to look at are **doalamo.py** and **shared.py** that control the majority of the capabilites available.