User guide for DAHSI

H. Ribera, S. Shirman, A. V. Nguyen, N. M. Mangan

November 10, 2021

Contents

1 Overview of the guide

This document walks through the steps for running the DAHSI algorithm presented in [1]. For the program to work we need IPOPT, a optimizer in the C language, PyIpopt, a python package which interfaces python with IPOPT, and Cython. Following installation we provide a quick guide to the code base and show how to run the DASHI algorithm.

2 Installing Required Programs and Packages

This document will assume that the user is using a Linux distribution and has basic compliers installed including gcc, gfortran and python.

2.1 Python Packages

These python scripts link to the sympy library. To install these, use

```
$ apt-get install sympy
```

if you are running Ubuntu
or

```
$ dnf install sympy
```

If you are running Fedora
or download directly from sympy.org.

If you already have sympy installed on your computer, make sure that you are running version 1.4. Otherwise you will recieve an error while trying to run XXX.

These scripts also rely on the numpy library. To install these, use

```
$ apt-get install numpy
```

if you are running Ubuntu
or

```
$ dnf install numpy
```

If you are running Fedora
or download directly from numpy.org.

2.2 IPOPT

The instructions for installing IPOPT have been modified from the minAone installation user guide by Jingxin Ye and Nirag Kadakia. Thanks to them for their clear explanation.

Download

Get it here: `https://projects.coin-or.org/Ipopt`

- Download and unzip latest version of IPOPT
- As of right now this is 3.11.7 - Efficacy of installation instructions may degrade over time as packages are updated.
- Go into ThirdParty folder in the IPOPT directory then execute the following commands.

```
$ cd Blas
$ ./get.Blas
$ cd ../Lapack
$ ./get.Lapack
$ cd ../ASL
$ ./get.ASL
$ cd ../Metis
$ ./get.Metis
$ cd ../Mumps
$ ./get.Mumps
```

- Get the HSL subroutines from `http://hsl.rl.ac.uk/ipopt`
- Note that there are two releases for HSL - you will want the more complete one that contains ma57, ma77, and ma97.
- While the freely available ma27 will work for many problems, the newer packages are faster, work on larger problems, and can use multi-core architecture.
- This will require filling out a form stating essentially that you are in academia and waiting a couple hours for a link to download.
- Unpack the resulting library into the ThirdParty folder such that the path is (IPOPT Path)/ThirdParty/HSL/coinhsl

Install

- Go to the IPOPT directory

```
$ mkdir build
$ cd build
$ ../configure
```

- Note that if you have lapack or blas installed previously you can use –with-lapack and –with-blas to link to those packages
- If something goes wrong refer here `http://www.coin-or.org/Ipopt/documentation/node19.html#ExpertInstall`
- Assuming everything worked:

```
$ make
$ make test
$ make install
```

2.3 PyIpopt

Following a sucessful IPOPT installation you must finish by installing pyIpopt.

Execute the following commands

- Download the pyIpopt directory

```
$ git clone http://github.com/xuy/pyipopt.git
```

- Open the setup.py file in the pyipopt directory. Edit the line beginning with

```
IPOPT_DIR =
```

to list the IPOPT build directory after the equal sign. This is the directory in which you ran the "make" commands of the previous section. This line should look something like this

```
IPOPT_DIR = 'Path-to-IPOPT/build/'
```

Note: The path above needs to be relative to the pyipopt directory for python to properly read the location. For example, if your pyipopt and IPOPT directories are located in the same place this would be

```
IPOPT_DIR = '../IPOPT-Directory/build/'
```

- Within the setup.py file find the line listing the libraries. You will see that the 'coinhsl' library is commented out. Uncomment this line.
- In the pyipopt directory build and install with the following two commands

  ```
  $ python setup.py build
  $ sudo python setup.py install
  ```

  If you do not have sudo access on your machine. Instead run

  ```
  $ python setup.py build
  $ python setup.py install --user
  ```

- Go back to your home directory. Open your .bashrc file

  ```
  $ gedit .bashrc
  ```

  At the end of the file add

  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/(IPOPT path)/build/lib/
  ```

- Reload your .bashrc file

  ```
  $ source ~/.bashrc
  ```

- Go into the examples folder in the pyIpopt directory.

  ```
  $ python hs071.py
  ```

  If this example runs with no issue, then you have successfully installed pyIpopt and are ready to download and run XXX.

2.4 Cython

First just install Cython by running the command (if you have pip installed)

```
$ pip install Cython --user
```

3 Running DAHSI

In this section we will first give an explanation of what each file in the DAHSI code package does and then give a quick guide on how to run one of the examples included in it.

3.1 File descriptions

There are a few files that build up the whole program. Here we go one by one to understand how it all works.

- `header.py`. This file imports those functions and variables that will be used in other files.
- `Read_Files.py`. This script reads all the information needed from the input files. These input files are three: `File1.txt`, `File2.txt` and `File3.txt`. Note that when counting lines, those starting with `#` are omitted. Let's see what information each of them contains.

  `File1.txt`.

  The first line contains the number of state variables (`num_vars`), parameters (`num_params`) and measured state variables(`num_meas`), the time step and the total number of time points (`num_tpoints`), in that order and separated by commas.

  The next lines assign names to our state variables, each name in one different line.

  The next lines assign names to the parameters we seek to estimate, each name in one different line.

  The next lines define the equations of our problem. One line for each equation.

  We now set the upper and lower bounds for the state variables, separated by a comma. One line for each state variable.

  Finally, we also get the upper and lower bounds for the parameters, separated by a comma. One line for each parameter.

  In the file example below, we can see that our problem has 3 state variables, 30 parameter to estimate and 2 measured state variables. Our data consists of 501 time steps, with a step size of 0.01. Our state variables names are x, y and z, and our parameter names are $Bk0j$, for $k = 1, 2, 3$, and $j = 1, \ldots, 10$. The bounds for the state variables and the parameters are the next lines that follow.

  The order of the state variables is first the observed variables and then the unobserved. In this case, we have two measured variables, x and z, and one unobserved variable, y. The equations are written in the same order.

File1.txt

```
# Number of state variables, number of parameters, number of measured...
...state variables, time step, number of time points.
3,30,2,0.01,501
# State variable names.
x
z
y
# Parameter names.
B101
B102
B103
B104
B105
B106
B107
B108
B109
B110
B201
B202
B203
B204
B205
B206
B207
B208
B209
B210
B301
B302
B303
B304
B305
B306
B307
B308
B309
B310
# Equations, in the same order as variables.
B101+B102*x+B103*y+B104*z+B105*x*x+B106*x*y+B107*x*z+B108*y*y+B109*y*z+B110*z*z
```

```
B301+B302*x+B303*y+B304*z+B305*x*x+B306*x*y+B307*x*z+B308*y*y+B309*y*z+B310*z*z
B201+B202*x+B203*y+B204*z+B205*x*x+B206*x*y+B207*x*z+B208*y*y+B209*y*z+B210*z*z
# Upper and lower bounds for the the state variables search (min,max).
-5,5
0,10
-10,10
# Upper and lower bounds for the parameter search (min,max).
-10,10
-20,20
-20,20
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-60,60
-10,10
-10,10
-10,10
-10,10
-10,10
-20,20
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
-10,10
```

`File2.txt`.

This file contains the information about the measured state variables.

In the first lines we give names to the data of each measured state variable. One line for each measured state variable.

The next lines include the file name (and its extension) in which we can find the data of each measured variable. One line per measured variable, and in the same order as in the first part of the file.

In the example file below, since we have two measured state variables: we call them `datax` and `dataz`; we then indicate that the data of our measured state variables can be found in the `datax_Circuit.dat` and `datay_Circuit.dat` files.

File2.txt

```
# Data names.
datax
dataz
# Data file names.extension
datax_LorenzSynth_0p01.dat
dataz_LorenzSynth_0p01.dat
```

`File3.txt`.

In this file we find information about the parameters needed to do the variational annealing process as well as the search in model space.

The first two lines define values for α and $\beta_{\max}$.

The next two lines indicate the initial λ and the maximum value it can attain. This λ value controls the sparsity of the models recovered. We should always start with one that includes all possible functions and set a maximum λ for which all the terms drop to zero.

In the example file below we have set $\alpha = 1.2$ and a maximum β value to 134. Initial λ is set to 2.5 and the maximum to 5.5.

File3.txt

```
# alpha, beta_max.
1.1
256
# For lambda sweep, define lambda_min and lambda_max.
0.3
0.5
```

- `Obj_Funks.py`. This file defines the measurement and model part of the action. For the measurement part, we have

$$\frac{1}{N}\sum_{\substack{i=1\\ i\,\text{odd}}}^{N-1}\left(\mathbf{X}(t_i)-\mathbf{Y}(t_i)\right)^2+\left(\mathbf{X}(t_{i+1})-\mathbf{Y}(t_{i+1})\right)^2. \tag{1}$$

The time discretisation used is given by the Hermite-Simpson's rule. Thus, the model part of the action is

$$\begin{aligned}\frac{1}{N}\sum_{\substack{i=1\\ i\,\text{odd}}}^{N-2} R_f\Bigg\{&\|\mathbf{X}(t_{i+2})-\mathbf{X}(t_i)-\frac{2\Delta t}{6}\left[\hat{\mathbf{F}}(\mathbf{X}(t_i))+4\hat{\mathbf{F}}(\mathbf{X}(t_{i+1}))+\hat{\mathbf{F}}(\mathbf{X}(t_{i+2}))\right]\|^2\\ &+\|\mathbf{X}(t_{i+1})-\frac{1}{2}\left(\mathbf{X}(t_{i+1})+\mathbf{X}(t_{i+2})\right)-\frac{2\Delta t}{8}\left[\hat{\mathbf{F}}(\mathbf{X}(t_i))-\hat{\mathbf{F}}(\mathbf{X}(t_{i+2}))\right]\|^2\Bigg\}\end{aligned} \tag{2}$$

These functions can be readily changed as desired by the user.

- `Build_ObjJacHess.py`. This file calculates the objective function, its Jacobian and Hessian for a general time point and saves them as strings.

- `WriteStrings.py`. This file takes the strings generated with `Build_ObjJacHess.py` and writes them in the "blank template" `OneLoopInC_Blank.pyx`, which will create the functions for the objective function, its Jacobian and Hessian in *cython* format. The output is the file `OneLoopInC.pyx`.

- `OneLoopInC_Blank.pyx`. "Blank template" , which will contain the functions for the objective function, its Jacobian and Hessian in *cython* format.

- `setup.py`. It contains the information to compile `OneLoopInC.pyx` into C functions. These functions will be used in `main_loop.py` when we create the functions that PyIpopt is going to call.

- `PyIpopt_Funks.py`. In this file we include the definitions of the constraint function of the problem and its Jacobian. If the problem is unconstrained, this file does not need to be changed.

- `compile.py`. Runs `Build_ObjJacHess.py`, `WriteStrings.py` and `setup.py`.

- `main_loop.py`. We create `eval_f`, `eval_grad_f` and `eval_h` in PyIpopt format. This file then implements Algorithm 1. $R_{f,0}$ and how λ is increased through the sweep are specified here.

  The output of this script is a file named `Di_Mj_ICk.dat`, where `i` is the number of state variables, `j` is the number of measured state variables and `k` is the `taskID` we have given when executing it.

  Each line in this file is the solution of the problem given a λ and a β value. For each λ, we will have $\beta_{\max} + 1$ lines. So, if we do a sweep for N_λ different λ values, the file will have $N_\lambda(\beta_{\max} + 1)$ lines.

  Each line contains:

  - In the first column, the λ value.
  - In the second column, the β value.
  - In the third column, the minimum of the cost function that corresponds to this initialisation at the last step of VA.
  - In the last `num_params` columns, we find the parameter estimates.

Algorithm 1 DAHSI Algorithm.

procedure DAHSI
  Input: measurements $\mathbf{Y}$, generic model library $\mathbf{\Theta}$, λ_{max}, β_{max}, α
  Calculate discrete function $\hat{\mathbf{F}}$ from $\mathbf{\Theta}$
  for $l = 1 : L$ **do**
    $x_l = y_l$ ▷ Fit measurements to data
  Randomly initialise unobserved variables $\{x_{l+1}, \ldots, x_D\}$
  $\mathbf{X}^{\text{ini}} = \{x_1, x_2, \ldots, x_l, x_{l+1}, \ldots, x_D\}$
  Initialise $\mathbf{p}^{\text{ini}} = 0$ ▷ Force sparsity
  Assemble pair $\{\mathbf{X}^{\text{ini}}, \mathbf{p}^{\text{ini}}\}$
  $R_{f,0} = \epsilon$
  while $\lambda < \lambda_{\max}$ **do**
    for $\beta = 0 : \beta_{\max}$ **do** ▷ Variational Annealing
      $R_f = R_{f,0}\alpha^{\beta}$
      $\{\mathbf{X}^{(\beta)}, \mathbf{p}^{(\beta)}\} = \min_{\mathbf{X},\mathbf{p}} A_E(\mathbf{X}, \mathbf{Y}) + R_f A_M(\mathbf{X}, \mathbf{p}, \hat{\mathbf{F}})$ ▷ Minimize via IPOPT
      if $p_{k,j}^{(\beta)} < \lambda$ **then** ▷ Hard-threshold $\mathbf{p}$
        $p_{k,j}^{(\beta)} = 0$
    model$^{(\lambda)} \leftarrow \mathbf{p}^{(\beta)}$ ▷ Store models
    $\lambda = 2\lambda$ ▷ Increase λ

3.2 Example to run DAHSI: Lorenz synthetic data

Here we will briefly walk through the steps to produce one output file from solving the problem of model selection using synthetic data from the Lorenz system when we only have information about x and z. `File1.txt`, `File1.txt`, and `File1.txt` used are the ones shown in Section 3.1.

- Download the DAHSI package from `https://github.com/hribera/DAHSI`.
- Go to the folder `Example_LorenzSynth` directory.
- Run the command to build the program

  ```
  $ python compile.py
  ```

  which effectively executes these three commands:

  ```
  $ python Build_ObjJacHess.py
  $ python WriteStrings.py
  $ python setup.py build_ext --inplace
  ```

  See Section 3.1 for details on what each of these files do.

  Note: If you have multiple versions of python installed, you must specify the appropriate version (Python 2 is recommended).
  Within the `compile.py` file, replace `python` with `python2` in each of the three lines.
  Run the command to build the program

  ```
  $ python2 compile.py
  ```

  which effectively executes these three commands:

  ```
  $ python2 Build_ObjJacHess.py
  $ python2 WriteStrings.py
  $ python2 setup.py build_ext --inplace
  ```

- Run the algorithm

  ```
  $ python main_loop.py taskID
  ```

  where `taskID` is a non-negative integer. This will run the algorithm with one initialisation, sweeping through $\lambda = 0.3$ through $\lambda = 0.5$, in steps of $\Delta\lambda = 0.05$.

Note: If you have multiple versions of python installed run the algorithm

```
$ python2 main_loop.py taskID
```

4 Scripts to plot figures

We provide all the data (obtained by running DAHSI for particular noise instances and parameters described in our manuscript) necessary to reproduce the figures in the manuscript. Figures can be reproduced directly from the `read_and_plot` folder without the need to re-run DAHSI by running the corresponding script that has for name the figure number and corresponding panel.

Alternatively, if one wants to re-run DAHSI for a new system or different algorithm parameters, one can create the `.mat` files necessary to re-plot the figures. The steps are as follows:

1. Compile the program and run N_I initialisations via `main_loop.py taskID`. We provide two examples, found in `Example_LorenzSynth` and `Example_Circuit` that are ready to be compiled with the corresponding input time-series. This will generate the output files needed.

   Note: In the `Example_LorenzSynth` directory we provide already the results of 100 initialisations in the subfolder `outputfiles` to facilitate running these MATLAB scripts.

2. Go to the `read_and_plot` folder.

   Note: If not running either of those examples, provide data for validation (will need to change `.m` files accordingly to correspond to the new data files).

3. Run `main_SaveResults.m`. This will generate two `.mat` files: `parameters_*` and `structures_*`. One needs to change where the DAHSI output files are located in `Read_ResultFiles` depending on the example one is running. By default, it takes the output files from `Example_LorenzSynth`.

4. To reproduce the AIC plots for the two examples given, one needs to run `AIC_LorenzSynth.m` and `AIC_LorenzCircuit.m`.

   Note: If one wants to generate the AIC plots for a different example than the ones provided, one will need to do parameter estimation on the models down-selected (given by `structures_*`). If that is the case, one will need to adapt `AIC_LorenzSynth.m`.

   Now Figure 1(a), (b), (f), (h), (g), Figure 2(a), (b), (c), and Figure 4(a), (c), (d) are reproducible with one's own generated `.mat` files.

In the `Example_LorenzSynth` directory we also provide an example bash script `sub.sh` that was used to submit parallel initialisations in Northwestern's High Performance Computing Cluster Quest.

References

[1] H. Ribera, S. Shirman, A. V. Nguyen, and N. M. Mangan, *Model selection of chaotic systems from data with hidden variables using sparse data assimilation*, arXiv preprint arXiv:2105.10068, (2021).