SELDOMpy v1.0
User manual

Developed by Luis Prado López (pradolopezluis@gmail.com)
Supervised by Alejandro F. Villaverde (afvillaverde@uvigo.gal)
and David Saque Henriques (davidh@iim.csic.es)
Universidade de Vigo

July 18, 2023

Contents

1 Introduction 1
1.1 Theoretical foundations 1
1.2 Version and publication history 1
2 License 1
3 Availability 2
4 Software contents 2
5 Requirements and installation 3
5.1 Requirements 3
5.2 Download and install using GitHub 3
5.3 Download and install using pip 3
6 Quick example using the MAPK network 4
6.1 Initial parameter configuration 4
6.2 Generating the initial model 4
6.2 Model reduction 6
6.3 Results 8
7 Quick example using the SP network and pyPESTO 9
7.1 Initial parameter configuration 9
7.2 Generating the initial model 9
7.2 Model reduction 11
7.3 Results 14
8 Contributors 15

1 Introduction

SELDOMpy v1.0 is a Python package capable of performing dynamic modeling of cell signaling networks from experimental data. It has been developed from the R package "SELDOM". Its purpose is to give the user community a tool that is easier to use and install than SELDOM. Although its results in very complex networks are not as good, because it does not optimize as well as SELDOM, it is a tool that can be useful for certain case studies and that still has pending improvements.

1.1 Theoretical foundations

Cell signaling networks are a mechanism that cells use to react to external stimuli. This response is represented by directed graphs formed by nodes and arrows. The value of these nodes can be measured experimentally by different methods and that is what SELDOM will use as input data.

From this data, an adjacency matrix will first be generated with the mutual information values of all the possible pairs of nodes that make up the network. With this matrix and the measurement data, the first model will be generated, whose simulated value will have to be reduced by means of optimization algorithms. Once minimized, as much as possible, you get a model in which you will have to try to eliminate as many arrow bows as possible. This process is called "model reduction" and is the last one performed by SELDOMpy. To do this, we try to eliminate each arrow bow, optimize the model without that bow as much as possible and simulate. Once simulated, its value is compared with the previous model (which still had that arc) and the Akaike criterion is applied to estimate whether it is possible to make the reduction or not. If so, try to remove another axis (without putting that one back) and repeat the process. If the criterion is not met, that axis would be put back and a different one would be tried to be eliminated.

Once you have tried to remove all the SELDOMpy axes you will be finished and you can save the final cell signaling network model generated in a binary file of extension ".pkl" that can be opened from Python. In addition, a graph that compares the values of the simulation of the final model obtained with the experimental data can also be displayed to confirm that the generated model is a valid model.

1.2 Version and publication history

The current version of SELDOMpy is v1.0 (released in July 2023).

2 License

SELDOMpy is licensed under the GNU General Public License version 3 (GPLv3), a free, copyleft license for software.

3 Availability

SELDOMpy can be downloaded from Pipy and from GitHub:

- https://pypi.org/project/SELDOMpy/
- https://github.com/lupralo31/SELDOMpy

4 Software contents

The SELDOMpy package has the following files and directories:

Root folder (/SELDOMpy/):

- **__init__.py:** Package initialization file, required for importing package functions.

Functions folder (/SELDOMpy/functions/):

- **gen_exps.py:** File for the generation of an "exps" list with the data of the experimental measurements of the nodes, extracted from an Excel file.
- **buildmim.py:** File for the generation of the adjacency matrix from the calculation of mutual information between all possible pairs of nodes in the network.
- **getLBODEmodel.py:** File to generate the first cell signaling network model, which will later have to be reduced.
- **getRandomLBODEmodel.py:** File to generate the model from a random adjacency matrix with the "Roulette Wheel" method.
- **optimization.py:** File to perform optimization that minimizes the value of model simulations.
- **simulate_logic_based_ode.py:** File to perform the simulation of the model and obtain an average of all the values of the simulated nodes.
- **simulate_logic_based_ode_obs.py:** File to perform the simulation of the model and obtain the values of the simulated nodes.
- **reduce_fun.py:** File with functions to reduce the number of arrow arcs that the model has.
- **plot_results.py:** File to graphically represent the experimental measurements of the nodes together with the values obtained from the simulation of the final model.
- **extras.py:** File with the definition of auxiliary functions and the "Ivpsol" class used to configure the simulation parameters.

Scr folder (/SELDOMpy/src/):

Folder with subdirectories containing .c and .h files required for model simulation.

Examples folder (/SELDOMpy/examples/):

Folder with two subdirectories, one for each model (“MAPK” and “SP”). Each of the subdirectories contains the following 3 files, with "model" being the model name:

- **model.xlsx:** Excel file with the data of the experimental measurements of the models. It is important that the data Excel always has this format, so the user is recommended to download this file and modify it adapting it to their needs.
- **model_training_and_reduce.py:** It contains the steps to follow with the SELDOMpy functions to be able to execute the SELDOM algorithm.
- **model_training_and_reduce_pyPESTO.py:** It contains the same steps that model_training_and_reduce.py with the only difference that now the optimization is done in the file itself using the optimizer "pyPESTO" instead of the one included in the package itself.

Documentation folder (/SELDOMpy/doc/):

Folder containing this manual.

5 Requirements and installation

5.1 Requirements

In order to install SELDOMpy on your computer you must meet the following requirements:

- Have Python 3 installed on your terminal.
- Have installed the following packages: numpy, scikit-learn, pandas, mealpy, matplotlib, setuptools, openpyxl, pypesto, scipy.

5.2 Download and install using GitHub

1. Download SELDOMpy from: https://github.com/lupralo31/SELDOMpy
2. Open it in a development environment.
3. Install the packages from the above dependencies by running the following in the terminal:

```
pip install numpy, scikit-learn, pandas, mealpy, matplotlib, setuptools,
openpyxl, pypesto, scipy
```

5.3 Download and install using pip

To install SELDOMpy using Pipy there are 2 possibilities depending on your computer:

- **If you are using a 64-bit Windows with Python 3.8**: It is sufficient to run the following:

  ```
  pip install SELDOMpy
  ```

- **If you are using another version of Python or operating system:** You will need to have a C compiler installed on your computer. Visual Studio offers one, but the package adapts to the one you want. The commands you must execute are the following:

  ```
  pip install numpy
  ```

  ```
  python -m pip install pip==22.0.4
  ```

  ```
  pip install SELDOMpy
  ```

When you run the `pip install SELDOMpy` command, the program's dependencies are automatically installed.

6 Quick example using the MAPK network

Once the SELDOMpy package is installed (as indicated above), its functions can be executed as long as it is imported from a program that uses the virtual environment in which it has been installed. This example will be explained from the file "MAPK_training_and_reduce.py" (in the path (/SELDOMpy/examples/MAPK/). For the program to work properly and SELDOMpy to detect the location of the Excel data, both this file and the MAPK_training_and_reduce.py must be copied to the root folder of the project.

6.1 Initial parameter configuration

The first step is to **import all the functions** of the package with the following command. The "os" package is also imported to save the results to a binary file:

```
from SELDOMpy import *
import os
```

Once the packages are imported, the randómica seed must be configured. This is used to vary the randomness when generating the adjacency matrix. It also establishes the number of "bins" that will be used to discretize the continuous measurements of the nodes and the maximum number of entries per node:

```
initial_seed = 11
rdm.seed(initial_seed)

NBINS = 10
REALK = 2
```

6.2 Generating the initial model

Once the initial parameters have been configured, the next step is to generate an initial cell signaling network model. To do this, the first thing is to import the data of the experiments from the Excel file. It is **IMPORTANT** that both this Excel file and the MAPK_training_and_reduce.py in which the commands are being executed are in the root directory of the project. The experiment data is imported into a list with the following commands:

```
exps, conc_data = gen_exps("MAPK.xlsx")

for iexp in range(len(exps)):
    exps[iexp].y0 = exps[iexp].exp_data[0]
```

The adjacency matrix is then generated from the mutual information data calculated according to the number of bins set at the start:

```
adjMat = buildmim(conc_data, NBINS)
```

Then the model is generated and the parameters of "ivpsol" are established to configure the simulation as desired by the user:

```
model = getLBODEmodel(adjMat=adjMat, random=True,
is_stimuli=exps[0].is_stimuli, maxInput=REALK, min_tau=0, max_tau=1,
max_n=5, min_n=1, min_k=0.1, max_k=1)

ivpsol = Ivpsol(atol=1e-5, rtol=1e-5, max_step_size=1e100,
max_num_steps=int(5000), max_error_test_fails=int(50))

model.nthreads = int(4)

model.namesSpecies = exps[0].namesSpecies
```

Once the model is generated, an optimization is carried out to minimize as much as possible the average value of its simulations. After optimizing, the results are saved in variables:

```
res = optimization(model=model, exps=exps, ivpsol=ivpsol, epoch=1500,
pop_size=50, miu_f=0.3, miu_cr=0.6)

f = res.f
time = res.time
fbest = res.fbest
xbest = res.xbest
neval = res.neval
numeval = res.numeval
```

This initial model can be saved to a binary file within a directory in the root folder of the project as follows:

```
list_save = [f, time, fbest, xbest, neval, numeval, model, conc_data, exps,
initial_seed]
file_name =
f"results/MAPK_opt_REALK_{REALK}_NBINS_{NBINS}_SEED_{initial_seed}.pkl"

direct = os.path.dirname(file_name)

if not os.path.exists(direct):
    os.makedirs(direct)

with open(file_name, "wb") as file:
    pickle.dump(list_save, file)
```

Finally, the results of the simulation are shown in a graph against the experimental data measured in the nodes:

```
for i in range(len(model.index_opt)):
    model.x[model.index_opt[i]-1] = res.xbest[i]
res_sim = simulate_logic_based_ode_obs(model, exps, ivpsol)
plot_results(exps, res_sim, 1, 10, 1, 4, 1)
```

If desired, you can save the graphics as PDF by pressing the button indicated at the bottom left:

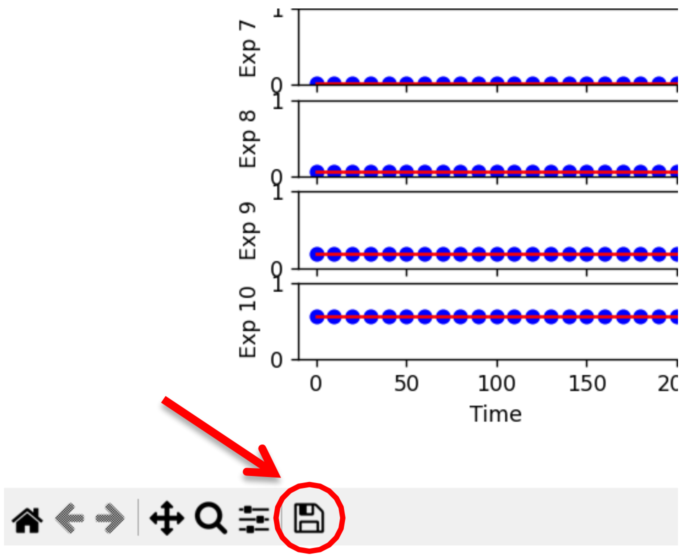


Figure 1: Graph save button.

6.2 Model reduction

Once the first cell signaling network model has been generated, the next step is to reduce the number of arrow bows it contains. To do this, we start by applying the Akaike criterion to the original model and calculating the number of arrow bows it has:

```
conc_data_train = copy.deepcopy(conc_data)

n_data_train = 0
for i in range(len(conc_data_train)):
    for j in range(len(conc_data_train[i])):
        if conc_data_train[i][j] is not None:
            n_data_train += 1

ivpsol = Ivpsol(atol=1e-5, rtol=1e-5, max_step_size=0.1,
max_num_steps=int(5000), max_error_test_fails=int(50))

model.nthreads = int(1)

res = akaike(xbest, model, exps, ivpsol, 10, n_data_train)

best_model = get_reduced_model(model, exps[0].is_stimuli)

edges = rdm.sample(range(1, best_model.count_inputs+1),
best_model.count_inputs)
```

Once the number of arrow bows has been calculated, we try to eliminate each of them by calculating the AIC of the model without that bow and comparing it with the previous AIC using the Akaike criterion. If the criterion is met, in the next iteration it is reapplied by removing another arrow bow without putting that one back. If it is not met, it is tried to eliminate another arc, but putting that one back.

```
for i in range(len(edges)):
    for p in range(len(xbest)):
        best_model.x[best_model.index_opt[p] - 1] = xbest[p]

    reduced = get_reduced_model(best_model, exps[0].is_stimuli, edges[i])

    opt = optimization(model=model, exps=exps, ivpsol=ivpsol, epoch=1500,
pop_size=50, miu_f=0.3, miu_cr=0.6)

    for p in range(len(opt.xbest)):
        reduced.x[reduced.index_opt[p] - 1] = opt.xbest[p]

    f_reduced = optim_fun(opt.xbest, reduced, exps, ivpsol)
    AIC_reduced = akaike(opt.xbest, reduced, exps, ivpsol,
reduced.active_pars, n_data_train)

    f_best = optim_fun(xbest, best_model, exps, ivpsol)
    AIC_best = akaike(xbest, best_model, exps, ivpsol,
best_model.active_pars, n_data_train)

    if AIC_reduced <= AIC_best:
        print('\nreduced')
        print('----------------')
        print(f_reduced)
        print(f_best)
        best_model = reduced
        xbest = opt.xbest
        print(edges[i])
        print(' ---------------- ')
    else:
        print('\nCould not reduce')
        print('----------------')
        print(f_reduced)
        print(f_best)
        print(edges[i])
        print(' ---------------- ')
```

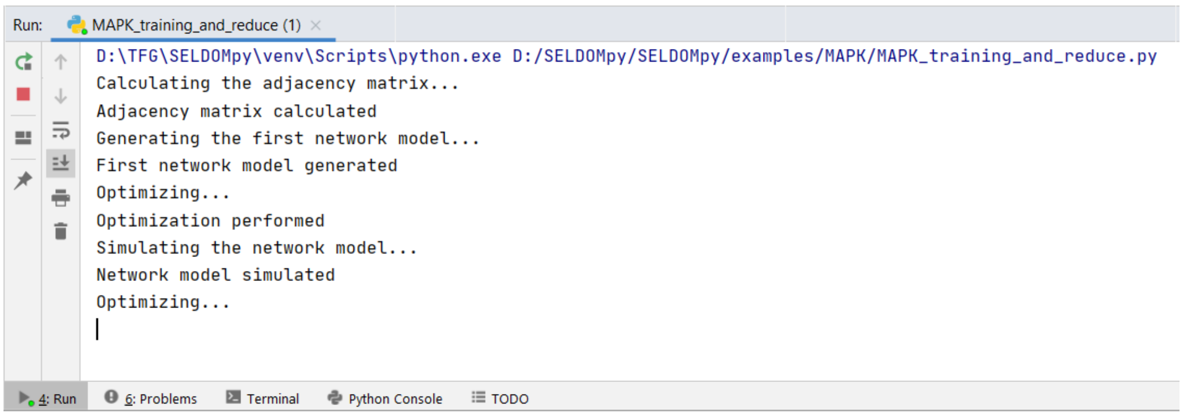
When the “for loop” is finished, the final model will be obtained, which as before can be saved in a file and represented graphically:

```
list_save = [xbest, best_model, exps, conc_data, initial_seed, ivpsol]

file_name =
f"results_reduced_AIC/MAPK_opt_REALK_{REALK}_NBINS_{NBINS}_SEED_{initial_seed
}.pkl"

direct = os.path.dirname(file_name)

if not os.path.exists(direct):
    os.makedirs(direct)

with open(file_name, "wb") as file:
    pickle.dump(list_save, file)
```

```
res_sim = simulate_logic_based_ode_obs(best_model, exps, ivpsol)
plot_results(exps, res_sim, 1, 10, 1, 4, 1)
```

6.3 Results

During the process of running SELDOMpy, screen messages will be displayed informing where the program is in the process:



```
D:\TFG\SELDOMpy\venv\Scripts\python.exe D:/SELDOMpy/SELDOMpy/examples/MAPK/MAPK_training_and_reduce.py
Calculating the adjacency matrix...
Adjacency matrix calculated
Generating the first network model...
First network model generated
Optimizing...
Optimization performed
Simulating the network model...
Network model simulated
Optimizing...
```

Figure 2: Process information messages.

At the end of the program the graphs of the final model will be obtained, which can be saved in PDF format as explained above.

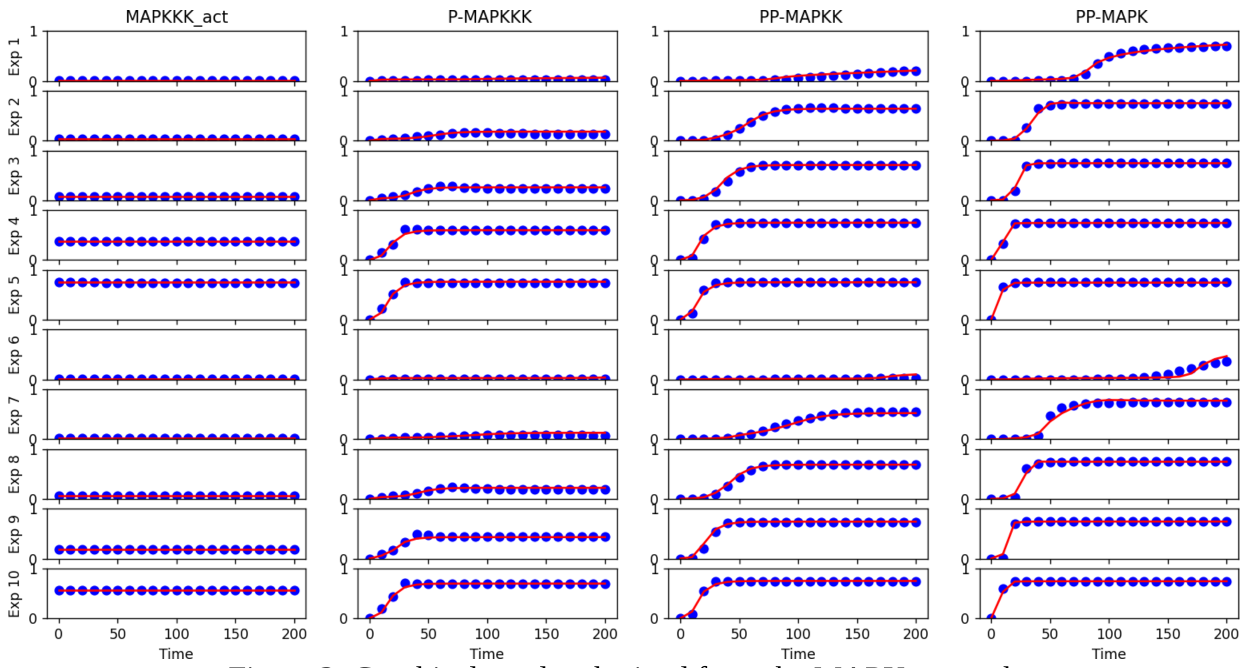


Figure 3: Graphical results obtained from the MAPK network.

7 Quick example using the SP network and pyPESTO

In order to be able to adapt to the different types of cell signaling networks, the two program examples also include two files that perform optimization with the pyPESTO package.

This time the example will be explained using the SP cell signaling network model. As with MAPK it will be necessary to import the SP_training_and_reduce_pyPESTO.py files and Excel SP.xlsx to the root folder of the project.

7.1 Initial parameter configuration

As with the MAPK network, the necessary packets are initially imported and certain parameters are configured (mentioned in section 6.1). As this time pyPESTO will be used instead of SELDOMpy's own optimizer, some more imports will have to be made.

```
from SELDOMpy import *
import os
import pypesto
import pypesto.optimize as optimize
import threading
from scipy.optimize import OptimizeWarning
import warnings

# Optimization warnings are filtered
warnings.filterwarnings("ignore", category=OptimizeWarning)

initial_seed = 6
rdm.seed(initial_seed)

NBINS = 10
REALK = 3
```

7.2 Generating the initial model

Following the steps in example 6, we proceed to generate the first cell signaling network model. For this, the procedure to be followed is mostly the same. The only difference is in the optimization, whose algorithm will now have to run entirely in the main.

As before, the instructions are executed to generate the list of experiments from the Excel file, the adjacency matrix from the mutual information values, the initial model of the signaling network "SP" and the configuration of the ivpsol parameter for the simulations.

```
adjMat = buildmim(conc_data, NBINS)

model = getLBODEmodel(adjMat, random=True, is_stimuli=exps[0].is_stimuli,
maxInput=REALK, min_tau=0.1, max_tau=12, max_n=5, min_n=1, min_k=0.1,
max_k=1)

ivpsol = Ivpsol(atol=1e-5, rtol=1e-5, max_step_size=0.1,
max_num_steps=int(5000), max_error_test_fails=int(50))

model.nthreads = int(8)
model.namesSpecies = exps[0].namesSpecies
```

Once these steps are done, an optimization must be performed, for which the target function is first defined:

```
def opt_fun(X):
    global numeval
    numeval += 1

    for w in range(len(model_op.index_opt)):
        model_op.x[model_op.index_opt[w] - 1] = X[w]

    sol_opt_fun = simulate_logic_based_ode(model_op, exps, ivpsol)
    if not f: # In the first simulation
        f.append(sol_opt_fun)
        time.append(datetime.datetime.now())
        neval.append(numeval)
    if sol_opt_fun < f[-1]:  # If the simulated value is less than the last
                             # best saved in list f
        f.append(sol_opt_fun)  # This new value is saved in list f
        time.append(datetime.datetime.now())  # The time it occurred is saved
                                              # in the "time" list
        neval.append(numeval)  # The iteration number in which that value was
                               # evaluated is saved

    return sol_opt_fun
```

Once defined, proceed to configure the **pyPESTO** optimizer. Because of the way it works, it must always be executed from the main with the condition if__name__== '__main__'. In each line its function is commented. Once this code is executed, an object is obtained with the same parameters that SELDOMpy.optimization() would give. The big difference is that now a global search with Scatter Search is combined with a local search with an optimizer (in this case Scipy Powell), but the user can choose from all those available in the pyPESTO library.

```
if __name__ == '__main__':
    # Boundaries are calculated
    LB = []
    UB = []
    for i in range(len(model_op.LB)):
        LB.append(model_op.LB[i])
        UB.append(model_op.UB[i])
    LB = np.array(LB)
    UB = np.array(UB)

    # The objective function is defined
    objective = pypesto.Objective(fun=opt_fun)

    # The optimization problem is defined
    problem = pypesto.Problem(objective=objective, lb=LB, ub=UB)

    # The local optimizer is defined
    optimizer = optimize.ScipyOptimizer(method="Powell")

    # The global optimizer is defined
    optimizer_ESS = pypesto.optimize.CESSOptimizer([{"dim_refset": 20,
"local_n1": 3, "local_n2": 3, "n_diverse": 50, "max_eval": 20000, "n_procs":
8, "local_optimizer": optimizer}], max_iter=2)
```

```
    # The starting points of the variables to be optimized are calculated
    start =
pypesto.startpoint.to_startpoint_method(pypesto.startpoint.uniform)

    # Optimization results are obtained
    result = optimizer_ESS.minimize(problem, start)

    # All active threads are obtained
    active_threads = threading.enumerate()

    # All threads are expected to finish their execution
    for hilo in active_threads:
        if hilo != threading.currentThread():
            hilo.join()

    fbest = result.optimize_result[0]['fval']
    xbest = result.optimize_result[0]['x'].tolist()

    res = Res_opt(f, time, fbest, xbest, neval, numeval)

    f = res.f
    time = res.time
    fbest = res.fbest
    xbest = res.xbest
    neval = res.neval
    numeval = res.numeval
```

Once this code is executed, the results can be saved in a binary file (as in MAPK) and displayed graphically along with the experimentally measured data.

```
list_save = [f, time, fbest, xbest, neval, numeval, model, conc_data, exps,
initial_seed]
file_name =
f"results/SP_opt_REALK_{REALK}_NBINS_{NBINS}_SEED_{initial_seed}_pyPESTO.pkl"

direct = os.path.dirname(file_name)

if not os.path.exists(direct):
    os.makedirs(direct)

with open(file_name, "wb") as file:
    pickle.dump(list_save, file)

for i in range(len(model.index_opt)):
    model.x[model.index_opt[i]-1] = res.xbest[i]

res_sim = simulate_logic_based_ode_obs(model, exps, ivpsol)
plot_results(exps, res_sim, 1, 10, 1, 4, 1.5)
plot_results(exps, res_sim, 1, 10, 5, 8, 1.5)
plot_results(exps, res_sim, 1, 10, 8, 13, 1.5)
```

7.2 Model reduction

The process of reducing the model will also be the same. Optimizations will be performed again with pyPESTO in the same way as explained in section 7.1. Once the reduction is finished, the final model is saved in a binary file to be imported into future projects and the results are displayed graphically.

```
if __name__ == '__main__':
    conc_data_train = copy.deepcopy(conc_data)

    n_data_train = 0
    for i in range(len(conc_data_train)):
        for j in range(len(conc_data_train[i])):
            if conc_data_train[i][j] is not None:
                n_data_train += 1

    ivpsol = Ivpsol(atol=1e-5, rtol=1e-5, max_step_size=0.1,
max_num_steps=int(5000), max_error_test_fails=int(50))

    model.nthreads = int(8)

    res = akaike(xbest, model, exps, ivpsol, 10, n_data_train)

    best_model = get_reduced_model(model, exps[0].is_stimuli)

    edges = rdm.sample(range(1, best_model.count_inputs+1),
best_model.count_inputs)

    for i in range(len(edges)):
        for p in range(len(xbest)):
            best_model.x[best_model.index_opt[p] - 1] = xbest[p]

        reduced = get_reduced_model(best_model, exps[0].is_stimuli, edges[i])

        f = []
        time = []
        numeval = 0
        neval = []
        model_op = copy.deepcopy(model)

        LB = []
        UB = []
        for u in range(len(model_op.LB)):
            LB.append(model_op.LB[u])
            UB.append(model_op.UB[u])
        LB = np.array(LB)
        UB = np.array(UB)

        objective = pypesto.Objective(
            fun=opt_fun,
        )

        problem = pypesto.Problem(objective=objective, lb=LB, ub=UB)

        optimizer = optimize.ScipyOptimizer(method="Powell")

        optimizer_ESS = pypesto.optimize.CESSOptimizer([{"dim_refset": 20,
"local_n1": 3, "local_n2": 3, "n_diverse": 50, "max_eval": 20000, "n_procs":
8, "local_optimizer": optimizer}], max_iter=2)

        start =
pypesto.startpoint.to_startpoint_method(pypesto.startpoint.uniform)

        result = optimizer_ESS.minimize(problem, start)

        active_threads = threading.enumerate()
        for hilo in active_threads:
            if hilo != threading.currentThread():
                hilo.join()

        fbest = result.optimize_result[0]['fval']
        xbest = result.optimize_result[0]['x'].tolist()

        opt = Res_opt(f, time, fbest, xbest, neval, numeval)

        for p in range(len(opt.xbest)):
            reduced.x[reduced.index_opt[p] - 1] = opt.xbest[p]

        f_reduced = optim_fun(opt.xbest, reduced, exps, ivpsol)
        AIC_reduced = akaike(opt.xbest, reduced, exps, ivpsol,
reduced.active_pars, n_data_train)

        f_best = optim_fun(xbest, best_model, exps, ivpsol)
        AIC_best = akaike(xbest, best_model, exps, ivpsol,
best_model.active_pars, n_data_train)

        if AIC_reduced <= AIC_best:
            print('\nreduced')
            print('----------------')
            print(f_reduced)
            print(f_best)
            best_model = reduced
            xbest = opt.xbest
            print(edges[i])
            print(' ----------------')
        else:
            print('\nCould not reduce')
            print('----------------')
            print(f_reduced)
            print(f_best)
            print(edges[i])
            print(' ----------------')

    list_save = [xbest, best_model, exps, conc_data, initial_seed, ivpsol]

    file_name =
f"results_reduced_AIC/SP_opt_REALK_{REALK}_NBINS_{NBINS}_SEED_{initial_seed}_
pyPESTO.pkl"

    direct = os.path.dirname(file_name)

    if not os.path.exists(direct):
        os.makedirs(direct)

    with open(file_name, "wb") as file:
        pickle.dump(list_save, file)
```

As this time the model is very large the graphs will be shown in 4 files, showing four nodes in the first two and six in the third:

```
    res_sim = simulate_logic_based_ode_obs(best_model, exps, ivpsol)
    plot_results(exps, res_sim, 1, 10, 1, 4, 1.5)
    plot_results(exps, res_sim, 1, 10, 5, 8, 1.5)
    plot_results(exps, res_sim, 1, 10, 8, 13, 1.5)
```

7.3 Results

SELDOMpy is a package created from the R SELDOM package. It was created to make it easier for the user community to install and use. However, it has trouble optimizing very complex networks.

SP is a network whose nodes are other signaling networks. The simplest ones, such as MAPK, are obtained perfectly. However, in very large networks it has more difficulties, as will be observed in the graphs, so it is pending to improve the optimization algorithm in the future. For everything else it works without problems achieving very good results some even in a shorter execution time than SELDOM.

The graphs obtained are the following, which can be saved in the user's terminal in the same way as in the previous example.

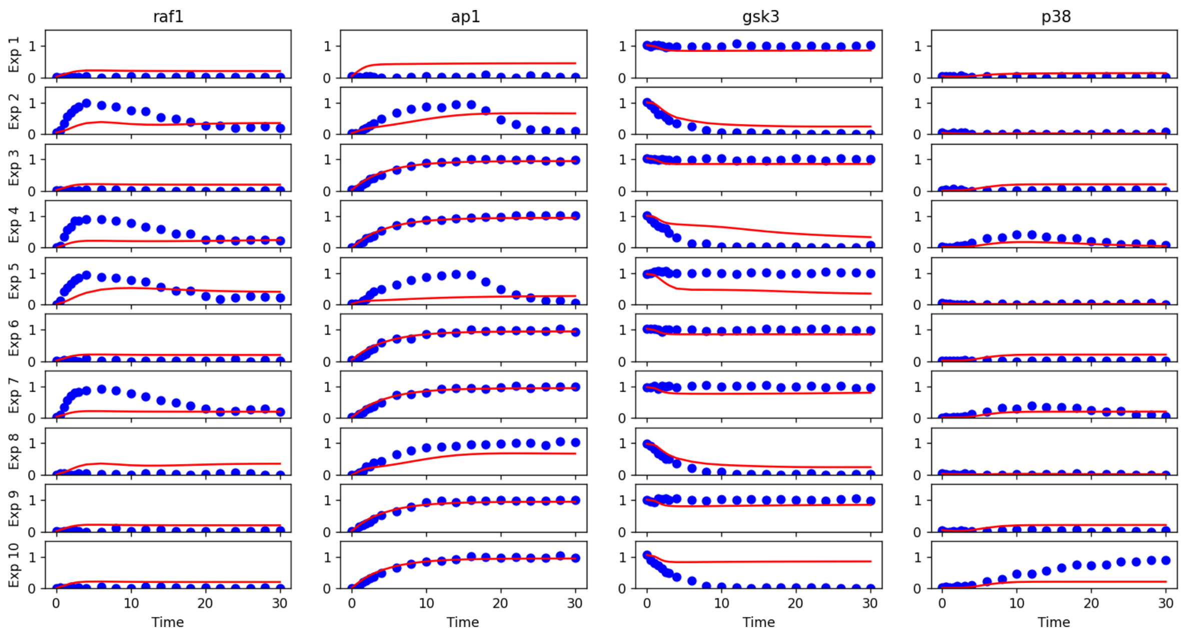


Figure 4: Graphical results obtained from the SP network using pyPESTO (Nodes 1-4).

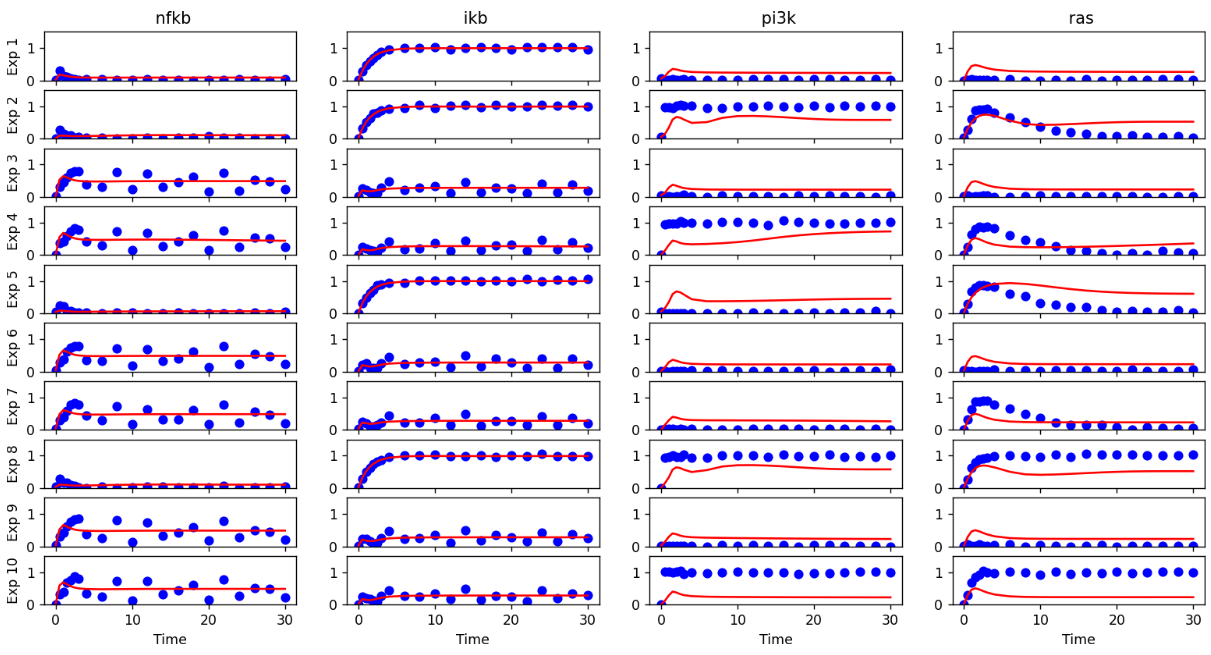


Figure 5: Graphical results obtained from the SP network using pyPESTO (Nodes 5-8).

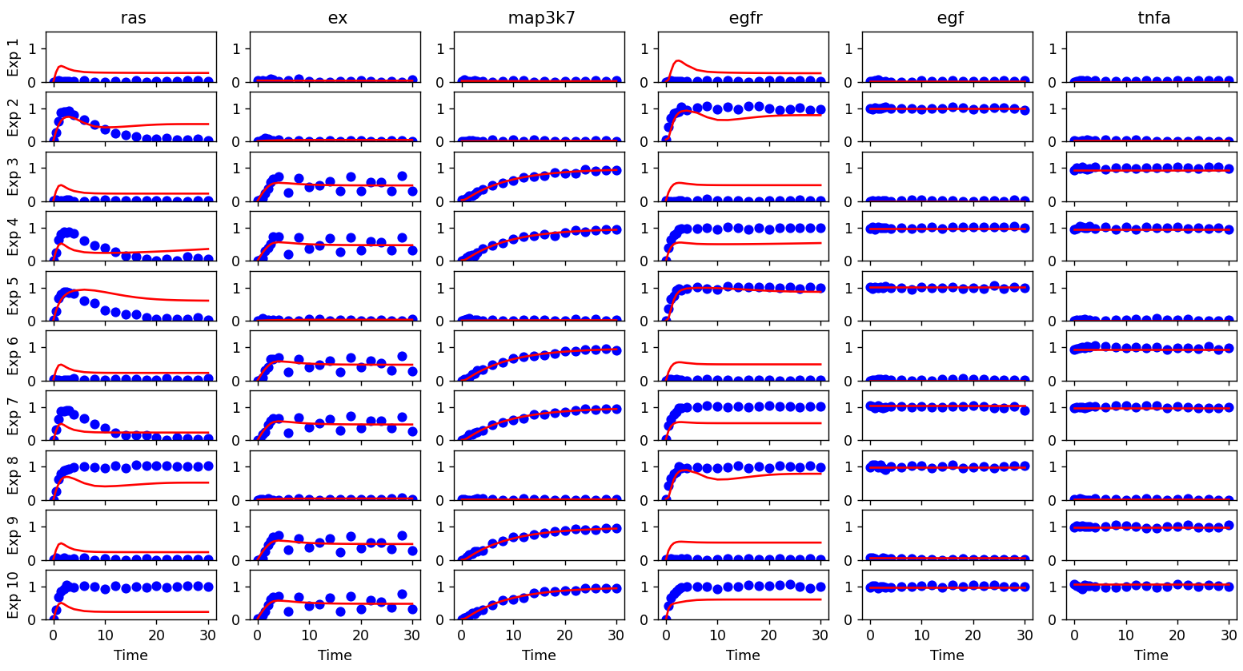


Figure 5: Graphical results obtained from the SP network using pyPESTO (Nodes 9-14).

8 Contributors

The SELDOMpy package has been created by Luis Prado López (pradolopezluis@gmail.com) as his Final Degree Project at UVigo. The project has been supervised at all times by Alejandro Fernández Villaverde (afvillaverde@uvigo.gal), researcher and professor at UVigo, and David Saque Henriques (davidh@iim.csic.es), researcher at the Institute of Marine Research of the CSIC. Both collaborated in the creation of SELDOM, a program from which SELDOMpy is derived.