“Feedback loop and self-loop detection in an ODE”

Contents

Introduction

A biological system can be mathematically represented as a system of ordinary differential equations (ODEs). The Jacobian matrix of a system of ODEs is the matrix of the partial derivatives. The Jacobian matrix can be represented as a graph where each species is a node and an entry unequal to zero in the matrix is an edge in the graph. With directed path detection in the graph it is possible to determine if a species is affecting itself either directly (self-loop) or via other species (loop). The loop detection package can calculate self-loops and loops and shows if these are postive or negative. The output is a data frame which can easily be filtered for different parameters (e.g. loop length, loops containing a certain edge). The packages NetworkX, Numpy and Pandas have to be installed to use loop detection.

Calculate Jacobian matrix

The following code shows how to calculate the Jacobian matrix of a user-defined ODE.

A file defining the function of the ODE system (*NEGm4*) and the parameters (*param_NEGm4*) will be opened. Here we use an example of a chain model with negative feedback. To calculate the Jacobian, we need the package Numdifftools and the function *numdifftools.Jacobian.* It is important that the function in the file returns a *numpy* array. :

```
import numdifftools as nd
import numpy as np
exec(open("NEGm4_func.py").read())
jac = nd.Jacobian(NEGm4)
jacobian = jac(np.array([1,0.1,1,1]),t=0,parameters=param_NEGm4)
jacobian
array([[-1.6 ,  0.  ,  0.  ,  0.75],
   [ 1.5 , -2.  ,  0.  , -0.75],
   [ 0.  ,  1.  , -3.  ,  0.  ],
   [ 0.  ,  0.  ,  2.  , -3.  ]])
```

The Jacobian can then be used in the function *get_all_loops.* If the user already calculated the Jacobian, it is also possible to use this in the function. The input has to be a *numpy* array.

Find all loops

To find all loops for the given Jacobian matrix the function *get_all_loops* creates a directed graph from the matrix with the function *networkx.DiGraph.* To find all cycles in the graph, the function uses *networkx.simple_cycles.* :

```
exec(open("loop_detection.py").read())
loops = get_all_loops(jacobian=jacobian)
loops
              loop length sign
0  [0, 1, 2, 3, 0]      4    1
1           [0, 0]      1   -1
2     [1, 2, 3, 1]      3   -1
3           [1, 1]      1   -1
4           [2, 2]      1   -1
5           [3, 3]      1   -1
```

The output is a *pandas* data frame with three columns: loop, length and sign. So each row in the data frame shows the loop, the respective length and sign (1=positive loop, -1=negative loop). In the example ODE we found six loops. The first one is a path with the nodes and edges 0-> 1-> 2-> 3-> 0, which is positive and has length four. The third one is a path with the nodes and edges 1-> 2-> 3-> 1, which is negative and has length three. The other four loops are negative self loops.

Helpful functions to get an overview about the results

Within the *pandas* data frame it is easy to sort (e.g. for length). The following example shows how to sort for length in ascending order with the inbuilt function *sort_values*. :

```
loops.sort_values("length")
              loop length sign
1           [0, 0]      1   -1
3           [1, 1]      1   -1
4           [2, 2]      1   -1
5           [3, 3]      1   -1
2     [1, 2, 3, 1]      3   -1
0  [0, 1, 2, 3, 0]      4    1
```

With the inbuilt function *value_count* it is also possible to get an overview how many negative and positive loops or how many loops with a certain length were found. :

```
loops.sign.value_counts()
-1    5
 1    1
Name: sign, dtype: int64

loops.length.value_counts()
1    4
3    1
4    1
Name: length, dtype: int64
```

Extract and read in the results

To store the results in a .tsv file for example, we can use the *pandas* function *.to_csv* and specify the file as tab-delimited. To read in such a file afterwards, we can use the *pandas.read_csv* function. :

```
loops.to_csv("loops_NEGm4.tsv",sep='\t')
loops = pandas.read_csv("loops_NEGm4.tsv",delimiter='\t')
```