Tutorial-2-Macquarie_Island_case_study

February 1, 2019

We define the Macquarie Island interaction network by the species present, the interactions between them, and the signs of these interactions. The network structure is encoded as a `networkx` digraph. The function `initialise_foodweb` stores the signs of the interactions in the edge data dictionary. The network can be drawn using `draw_foodweb`.

In [90]:
```
from qualmod import initialise_foodweb, draw_foodweb
from qualmod import qualitative_community_matrix, get_conditions_lists
import numpy as np
import time

from IPython.display import Image
import os

sppList = [
    'albatrosses',
    'prions',
    'burrowSeabirds',
    'petrels',
    'herbfield',
    'macroInverts',
    'mice',
    'penguins',
    'rabbits',
    'rats',
    'redpolls',
    'skuas',
    'surfaceSeabirds',
    'tussock',
]

# key is recipient of positive effect
positive_edges_dict = {
    'prions':           ['grassland'],
    'skuas':            ['prions','burrowSeabirds','rabbits','penguins'],
    'petrels':          ['penguins','tussock','grassland'],
    'mice':             ['herbfield','macroInverts','tussock'],
    'rats':             ['macroInverts','herbfield','tussock'],
```

```
        'burrowSeabirds':   ['tussock'],
        'rabbits':          ['tussock','herbfield','grassland'],
        'macroInverts':     ['herbfield','grassland','tussock'],
        'albatrosses':      ['tussock','herbfield'],
        'redpolls':         ['macroInverts','tussock','herbfield','grassland'],
    }
    negative_edges_dict = {
        'prions':           ['prions','skuas'],
        'skuas':            ['skuas','tussock'],
        'penguins':         ['penguins','skuas','petrels'],
        'petrels':          ['petrels'],
        'mice':             ['mice','rats'],
        'rats':             ['rats'],
        'burrowSeabirds':   ['burrowSeabirds','skuas','rabbits'],
        'rabbits':          ['rabbits','skuas'],
        'surfaceSeabirds':  ['surfaceSeabirds','rats'],
        'macroInverts':     ['macroInverts','rats','mice','redpolls'],
        'tussock':          ['tussock','mice','rats','rabbits'],
        'albatrosses':      ['albatrosses'],
        'herbfield':        ['herbfield','rabbits'],
        'grassland':        ['grassland'],
        'redpolls':         ['redpolls'],
    }

    web = initialise_foodweb(positive_edges_dict, negative_edges_dict)

    draw_foodweb(web, f_name = 'macq1.dot')
    # call graphviz to create a png, display in markdown cell
    os.system("dot -Tpng macq1.dot > macq1.png")
```

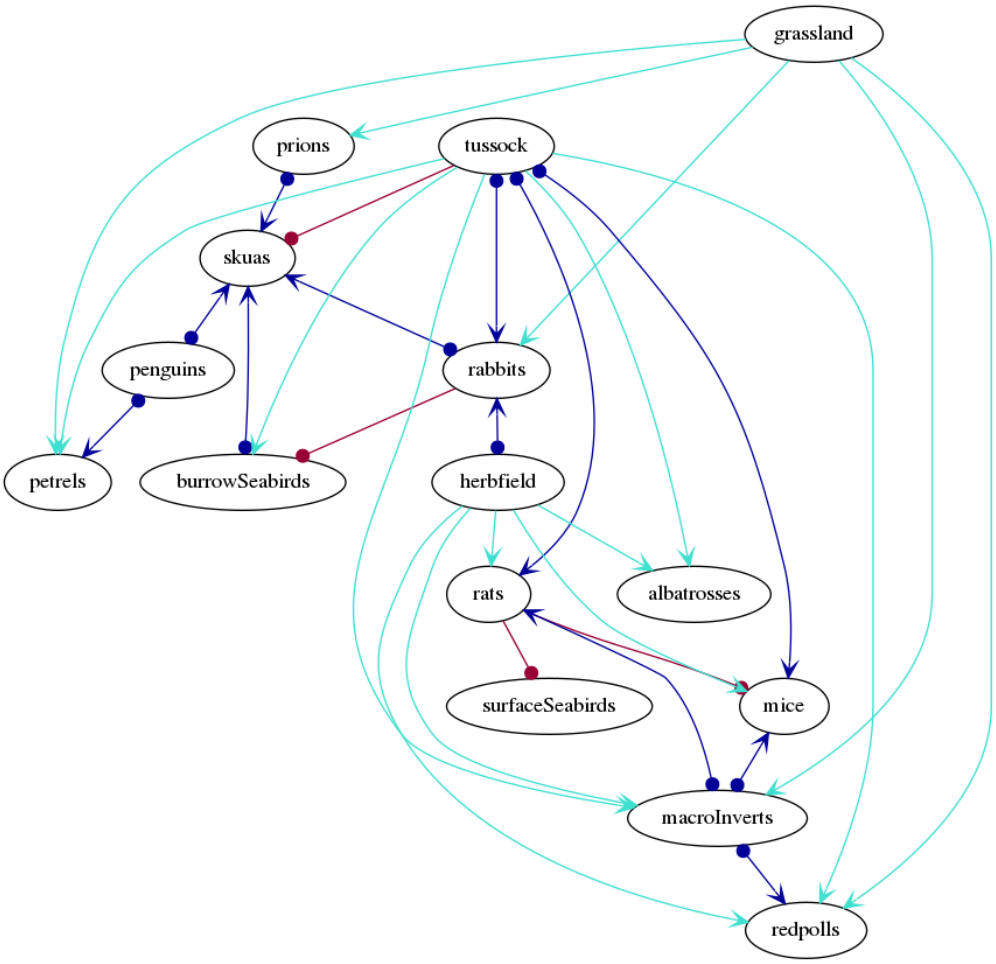
Out[90]: 0

We encode the validation criteria (with respect to a positive perturbation of the rabbits).

```
In [74]: control_list = ['rabbits']

    # Reponse to increase in rabbits
    validation = {
        'rabbits': +1,
        'tussock': -1,
    }
```

We convert the interaction network into a qualitative community matrix, `Mq` below, using the function `qualitative_community_matrix`.

```
In [78]: sz = web.order()
    (Mq, s2idx) = qualitative_community_matrix(web)
    print(Mq)
```


grassland
prions
tussock
skuas
penguins
rabbits
petrels
burrowSeabirds
herbfield
rats
albatrosses
surfaceSeabirds
mice
macroInverts
redpolls

title

```
[[-1.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0. -1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.  1.]
 [ 0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  1. -1. -1.  0.  0.  0.  0. -1. -1.  0.  0.  1.]
 [ 0.  0.  0.  1.  1. -1.  0.  0.  0.  0. -1.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. -1. -1.  0.  0.  0.  0. -1.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  1. -1.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0. -1.  0.  0.]
 [ 0.  0.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.  1.]
 [ 0.  0.  0.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0.  1.]
 [ 0.  0.  1.  1.  1.  0.  0.  0.  0.  0.  0. -1.  0.  0.  1.]
 [ 0.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0. -1.  0. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.]
 [ 0.  0.  0.  0.  0. -1.  0.  0.  0. -1. -1.  0.  0.  0. -1.]]
```

The `s2idx` returned is a bidirection dictionary that maps the name of the species to their index in the qualitative community matrix

```
In [79]: s2idx.inv[9]
```

```
Out[79]: 'rabbits'
```

The validation conditions with respect to the indices of the qualitative matrix can be obtained using the function `get_conditions_list`.

```
In [77]: # validation condition
         condn_idxs, condn = get_conditions_lists(s2idx, validation)

         print(list(zip(condn_idxs, condn)))
```

```
[(9, 1), (14, -1)]
```

To save time in this tutorial, I have hard-coded in the number of species-response combinations to be found, which is 36. If you do not know what the number is beforehand, you would use a line similar to the one commented-out above the start of the `while` loop.

The `while` loop collects species response combinations using a parameter-value sweep of the model.

```
In [91]: # Initialise collected responses with empty sets
         # Key is species perturbed and set will contain tuples of responses
         collectedResponses = {spp: set() for spp in control_list}

         t = 1
         t_last_updated = 0
         start_time = time.time()
```

```
# use something like this if you don't know the answer beforehand ~ 10 minutes
# while t < 1e6:

# I know there are 36 responses to find
while len(collectedResponses['rabbits']) < 36:


    # Find valid stable community matrix
    valid = False

    while not valid:

        # Find a random community matrix that is stable
        maxEig = 1

        while maxEig > 0:
            M = np.multiply(np.random.random_sample((sz,sz)), Mq)
            maxEig = max(np.real(np.linalg.eigvals(M)))

        # Now have a stable matrix

        # Find sensitivity matrix
        Sq = - np.linalg.inv(M)

        # Check validation criteria
        valid = all(np.sign(Sq[condn_idxs,s2idx['rabbits']]) == condn)

    # Now have a valid stable community matrix

    for ps in control_list:

        # NOTE: looking at increase in rabbits not decrease here
        response = tuple('neg' if Sq[s2idx[ms],s2idx[ps]]<0 else 'pos'
                         if Sq[s2idx[ms],s2idx[ps]]>0 else 'zer'
                         for ms in sppList)

        if response not in collectedResponses[ps]:

            # fs[ps].write(','.join(response) + '\n')
            collectedResponses[ps].add(response)
            t_last_updated = t

    t += 1

end_time = time.time()
time_elapsed = end_time - start_time
print('time elapsed = ' + str(time_elapsed) + ' seconds')
print('number of matrices sampled = ' + str(t-1))
```

```
        print('last new combination found at t = ' + str(t_last_updated))
```

```
time elapsed = 0.5220227241516113 seconds
number of matrices sampled = 889
last new combination found at t = 889
```

The possible species-response combinations are now stored in `collectedResponses['rabbits']`. If more than one control-species is involved, then its name would be the key for the `collectedResponses` dictionary.

```
In [95]: print( ' '.join([ ss[:3] for ss in sppList]))
         for response in collectedResponses['rabbits']:
             print(' '.join(response))
```

```
alb pri bur pet her mac mic pen rab rat red sku sur tus
neg pos neg neg neg pos neg pos pos pos neg neg neg neg
neg pos neg neg neg neg neg pos pos neg neg neg pos neg
neg neg neg neg neg neg neg neg pos neg neg pos pos neg
neg neg neg neg neg neg pos pos pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos neg pos neg pos neg
neg pos neg neg neg pos neg pos pos pos pos neg neg neg
neg pos neg pos neg pos neg pos pos neg pos neg pos neg
neg pos neg pos neg pos pos pos pos neg neg neg pos neg
neg pos neg pos neg pos pos pos pos pos neg neg neg neg
neg pos neg pos neg neg pos pos pos neg neg neg pos neg
neg pos neg pos neg pos neg pos pos pos neg neg neg neg
neg pos neg pos neg neg neg pos pos neg neg neg pos neg
neg neg neg neg neg neg pos neg pos neg neg pos pos neg
neg pos neg pos neg pos neg pos pos neg neg neg pos neg
neg pos neg pos neg pos pos pos pos neg pos neg pos neg
neg neg neg neg neg neg neg pos pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos pos neg neg neg neg
neg pos neg neg neg neg pos pos pos neg neg neg pos neg
neg neg neg neg neg pos pos pos pos neg pos neg neg
neg neg neg neg neg pos neg pos pos neg pos neg neg
neg neg neg neg neg pos neg neg pos pos pos pos neg neg
neg neg neg neg neg pos pos neg pos pos neg pos neg neg
neg neg neg neg neg pos pos pos pos neg neg pos pos neg
neg neg neg neg neg pos neg neg pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos neg neg neg pos neg
neg pos neg neg neg pos neg pos pos neg pos neg pos neg
neg neg neg neg neg pos pos neg pos neg neg pos pos neg
neg neg neg neg neg pos neg pos pos neg neg pos pos neg
neg neg neg neg neg pos pos pos pos neg pos pos pos neg
neg neg neg neg neg pos neg neg pos neg pos pos pos neg
```

```
neg neg neg neg neg pos neg pos pos neg pos pos pos neg
neg neg neg neg neg pos pos neg pos neg pos pos pos neg
neg pos neg neg neg pos neg pos pos neg neg neg pos neg
```

At this point in the process, it is a good idea to write the model responses. Here we write them to a csv file.

```
In [82]: # write output files

         fs = {ps: open('uniques_web1_' + ps + '.csv', 'w') for ps in control_list}

         for ps in control_list:

             header = [ps + '_' + ms for ms in sppList]
             fs[ps].write(','.join(header) + '\n')

             for response in collectedResponses[ps]:

                 fs[ps].write(','.join(response) + '\n')

             fs[ps].close()

         print(header)
```

```
['rabbits_albatrosses', 'rabbits_prions', 'rabbits_burrowSeabirds', 'rabbits_petrels', 'rabbits_
```

From here onwards, we will be performing the Boolean analysis on the species responses that were found in the model.

We read in the responses from the csv file that was written previously.

```
In [92]: import csv
         from pyeda.inter import espresso_exprs
         from findpcu import getUnobservedInts, getRespvarList2BoolvarList
         from findpcu import intList2boolexpr, boolexpr2RespvalList

         fInName = ('uniques_web1_rabbits.csv')
         str4true = 'pos'
         str4false = 'neg'


         fIn = open(fInName)
         csv_f = csv.reader(fIn)
         allResponses = next(csv_f) # get the header
         fIn.close()

         allResponses
```

```
Out[92]: ['rabbits_albatrosses',
          'rabbits_prions',
          'rabbits_burrowSeabirds',
          'rabbits_petrels',
          'rabbits_herbfield',
          'rabbits_macroInverts',
          'rabbits_mice',
          'rabbits_penguins',
          'rabbits_rabbits',
          'rabbits_rats',
          'rabbits_redpolls',
          'rabbits_skuas',
          'rabbits_surfaceSeabirds',
          'rabbits_tussock']
```

We already know that the response of tussock to rabbits will be negative, so we write a list of desired responses, in the same order as `allResponses`, that omits tussock. The `desiredResponses` list is converted into a Boolean mask, and passed to the function `unobservedInts`, which uses the list of observed responses and desired responses to return a list of unobserved responses as a list of integers.

```
In [93]: # skip tussock, because its response was a validation criterion
         desiredResponses = [
             'rabbits_albatrosses',
             'rabbits_prions',
             'rabbits_burrowSeabirds',
             'rabbits_petrels',
             'rabbits_herbfield',
             'rabbits_macroInverts',
             'rabbits_mice',
             'rabbits_penguins',
             'rabbits_rats',
             'rabbits_redpolls',
             'rabbits_skuas',
             'rabbits_surfaceSeabirds']

         boolLen = len(desiredResponses)

         # A mask to take out only those entries in our desiredResponses
         desiredResponsesMask = [True if aR in desiredResponses
                                 else False for aR in allResponses]

         unobservedInts = getUnobservedInts(fInName, desiredResponsesMask, boolLen, str4true)

In [85]: len(unobservedInts)

Out[85]: 4060
```

The function `getRespvarList2BoolvarList` is used to convert the possible species-responses in the model into Boolean variables. It uses the PyEDA package (https://pyeda.readthedocs.io/en/latest/).

The function `intList2boolexpr` turns the list of integers `unobservedInts`, corresponding to unobserved species responses, into a Boolean expression.

```
In [86]: # Create our boolean variables and some useful dictionaries
         x, x2s, r2idx = getRespvarList2BoolvarList(desiredResponses, str4true, str4false)

         # Turn each integer representing unobserved into a
         # into boolean expression
         unobservedBoolexpr = intList2boolexpr(unobservedInts, x)
```

The function `espresso_exprs`, imported from PyEDA, is used to perform the Boolean minimisation.

```
In [87]: start_time = time.time()

         boolExprMin, = espresso_exprs(unobservedBoolexpr)

         end_time = time.time()
         time_elapsed = end_time - start_time # ~ 1 second
         print(str(time_elapsed))
```

```
1.2536895275115967
```

This results in a Boolean expression that summarises the species responses that were not observed in the parameter-sweep of the model, and are assumed to be impossible.

```
In [88]: boolExprMin
```

```
Out[88]: Or(rabbits_albatrosses, rabbits_burrowSeabirds, rabbits_herbfield, And(rabbits_mice, ra
```

The function `boolexpr2RespvalList` converts each of the PCUs into a list of strings, which can be useful if we wish to store the results in a file, and is easier to read.

```
In [89]: PCUList = boolexpr2RespvalList(boolExprMin, x2s)
         PCUList
```

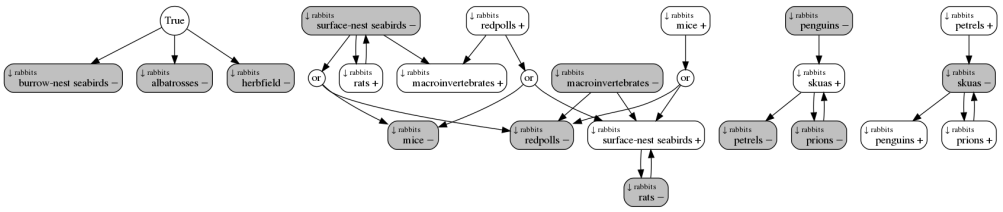
```
Out[89]: [['posrabbits_burrowSeabirds'],
          ['posrabbits_albatrosses'],
          ['posrabbits_herbfield'],
          ['negrabbits_surfaceSeabirds', 'negrabbits_rats'],
          ['negrabbits_penguins', 'negrabbits_skuas'],
          ['negrabbits_macroInverts', 'posrabbits_redpolls'],
          ['posrabbits_skuas', 'posrabbits_petrels'],
          ['negrabbits_skuas', 'negrabbits_prions'],
          ['posrabbits_prions', 'posrabbits_skuas'],
          ['posrabbits_rats', 'posrabbits_surfaceSeabirds'],
          ['negrabbits_macroInverts', 'negrabbits_surfaceSeabirds'],
          ['posrabbits_redpolls', 'negrabbits_surfaceSeabirds', 'posrabbits_mice']]
```

The `PCUList` form is also used as an input to the `draw_implication_network` function, which draws all logical implications resulting from the Boolean minimisation in their single-antecedent form.

In [94]:
```
import os
from findpcu import draw_implication_network

niceNames = {
        'rabbits': 'rabbits',
        'petrels': 'petrels',
        'mice': 'mice',
        'burrowSeabirds': 'burrow-nest seabirds',
        'macroInverts': 'macroinvertebrates',
        'herbfield': 'herbfield',
        'redpolls': 'redpolls',
        'skuas': 'skuas',
        'rats': 'rats',
        'surfaceSeabirds': 'surface-nest seabirds',
        'penguins': 'penguins',
        'prions': 'prions',
        'albatrosses': 'albatrosses',
        'tussock': 'tussock'
        }

controlSymbol = '&darr; '

draw_implication_network(PCUList,
                         'uniques_web1_rabbits_pcus',
                         niceNames, controlSymbol)

# call graphviz to create a png, display in markdown cell
os.system("dot -Tpng uniques_web1_rabbits_pcus.dot > uniques_web1_rabbits_pcus.png")
```

```
uniques_web1_rabbits_pcus.pdf has been created
```

Out[94]: 0



Implication network.