Intelligence in Animals and Machines

Lab 3: Robinson et al (2011)

Maxine Sherman

m.sherman@sussex.ac.uk

The code: parameters

```
# these parameters are for the first experiment
#
# probabilities of visiting each site from each other
probs = np.array([[0.91, 0.15, 0.03], [0.06, 0.8, 0.06], [0.03, 0.05, 0.91]])

# mean time to get between each nest
time_means = np.array([[1, 36, 143], [36, 1, 116], [143, 116, 1]])
# standard deviation of time to get between each nest
time_stddevs = time_means / 5

# mean quality of each nest. Note home is -infinity so it never gets picked
quals = np.array([-np.inf, 4, 6])

# standard deviation of quality: essentially this controls
# how variable the ants assessment of each nest is. This is currently set
# as in the 1st experiment where the variability is the same for each nest
qual_stddev = np.array([1, 1, 1])
# However, if you want to change is so nests perceived w different accuracy
# you could do eg qual_stddev = [1, 1, 4]

# set the number of ants
n = 27

# these govern the ant's threshold
threshold_mean = 5
threshold_stddev = 1
```

The code: how it works

```
def RobinsonCode(n, quals, probs, threshold_mean, threshold_stddev, qual_stddev, time_means, time_stddevs, ToPlot, q
    #   n = number of replicates (>=1)
    #   quals = row vector of m site qualities
    #   (quals(1) = home site
    #       quality: -Inf for no effect of home site quality on searching)
    #   discovery_probabilities = m * m matrix of discovery probabilities from
    #     column site to row site (N.B. columns should sum to 1)
    #   threshold_mean: mean population threshold for site acceptability
    #   threshold_stddev: standard deviation in population thresholds
    #   qual_stddev: standard deviation in quality assessments: **AOP**
    #   time_means: m * m matrix of mean travel times from column site to row
    #     site (N.B. should probably be symmetric)
    #   time_stddevs: m * m matrix of travel time standard deviations, from
    #     column site to row site (N.B. should probably be symmetric)
    #   quora: 1 * m matrix of quorum times for each nest site
    #
    #   times = row vector of times to first recruitment (i.e. nest acceptance)
    #   discovers = matrix (m x i) of times of first visit to each site
    #   visits = matrix (m x i) of numbers of visits to each site
    #   accepts = row vector of ids of accepted sites (indexed from 1 (for home nest) to m)
    #   the equivalents prefixed 'preq' are the pre-quorum equivalents of these

    # MATLAB allows variables to be created (and to grow) implicitly when elements are assigned
    # - Python doesn't play that foolish game, so we have to assign some variables here which we didn't in MATLAB
    nestNum = probs.shape[0]
```

Simulation details: https://canvas.sussex.ac.uk/courses/27059/pages/robinson-et-al-model-simulation-details

Strongly recommend you read this!

The code: how it works

1. Loop ants

```
# corresponds to line 34 in m-code
for i in range(n): # note that Python indexing is from 0, whereas MATLAB is from 1
```

2. Get threshold.

```
# sample and set the ant's acceptance threshold **AOP**
thresh = threshold_stddev * np.random.randn() + threshold_mean
```

3. Continue steps until the ant has chosen a site

```
#    it's threshold and a randomly selected qua
#    of the site **AOP**
# 2. Do this until a site is accepted
while Ants[i]['selected'] == 0:
```

The code: how it works

4. At current site, evaluate the quality and compare to the threshold. Stop if it was selected.

```
    # check the quality of the current nest
    perceivedQuality = qual_stddev[accepts[i]] * np.random.randn() + quals[accepts[i]]

    # if the perceived nest quality is above the threshold, select it
    if perceivedQuality >= Ants[i]['thresh']:
        Ants[i]['selected'] = 1
        break
```

5. If unselected, stop if max steps has been reached

```
# if you have exceeded the max number of steps without stopping
# break out of the algorithm
if num_step > Max_num_steps:
    Ants[i]['selected'] = 0
    break
```

The code: how it works

6. If continuing, choose a new site.

Remember: **accepts** is where the ant is *now*, **probs** is the probability of getting to any site (row) from any site (column).

1. Get a random number, **ran**, between 0 and 1 & initialise the new site at site 0.

2. If ran is bigger than the probability of P(to new site from where ant is now), go there.

3. Continue going as far away as you can until remaining P < ran

```
# probablistically pick one of the new sites to go to
# unifrnd(0,1) generates a uniformly distributed number
# between 0 and 1
ran = np.random.uniform()
# this then does the site picking. Looks complicated but is standard
# and it works
newsite = 0 # NOTE: THIS IS 0 INSTEAD OF 1 DUE TO PYTHON INDEXING!
while ran > probs[newsite, accepts[i]]:
    ran = ran - probs[newsite, accepts[i]]
    newsite = newsite + 1
```

	From A	From B	From C
To A	0.91	0.15	0.03
To B	0.06	0.80	0.06
To C	0.03	0.05	0.91

Suppose ran = 0.98 and accepts = 0:

Compare ran (0.98) to probs[0,0] = 0.91
ran = 0.98 – 0.91 = 0.07
newsite = newsite + 1 = 1

Compare ran (0.07) to probs[newsite, 0] = 0.06
ran = 0.09 - 0.06 = 0.01
newsite = newsite + 1 = 2

Compare ran (0.01) to probs[newsite, 0] = 0.03
STOP

The code: how it works

6. There is random variation in journey times. Draw a timestep from N(mean, SD), where mean and SD are parameters. Must be at least 1! Add that to the total time.

```
# update the time taken with normally-distributed time-step size
# (>=1) **AOP**
delta = max(1, time_stddevs[newsite, accepts[i]] * np.random.randn() + time_means[newsite, accepts[i]])
current_time[i] = current_time[i] + delta
```

7. Log other info – how long it took to discover the site total

```
# update ant's current site, accepts, **AOP**
# discovers, and the number of times it has been visited, visits **AOP**
accepts[i] = newsite

# if it hasn't discovered this site before, update the time that it
# 1st discovered it, in discovers **AOP**
if discovers[newsite, i] == 0:
   discovers[newsite, i] = current_time[i]

# update the number of times it has visited this site **AOP**
visits[newsite, i] = visits[newsite, i] + 1

# Update the output variables **AOP**
num_step = num_step + 1
Ants[i]['path'].append(accepts[i])
Ants[i]['t'].append(current_time[i])
```