SUPPLEMENTARY INFORMATION: A GENERALISED RANDOM ENCOUNTER MODEL FOR ESTIMATING ANIMAL DENSITY WITH REMOTE SENSOR DATA

S1. Table of symbols

Symbol	Description	Units
v	Velocity	$\mathrm{m\,s^{-1}}$
θ	Angle of detection	Radians
α	Animal call/beam width	Radians
r	Detection distance	Metres
$\bar{p}$	Average profile width	Metres
p	A specific profile width	Metres
t	Time	Seconds
z	Number of detections	
D	Animal density	animals $\mathrm{m^{-2}}$
x_i	Focal Angle $i \in \{1, 2, 3, 4\}$	Radians
T	Step length	Seconds
N	Number of steps per simulation	
d	Time step index	
S	Probability of remaining stationary	
A	Probabilty of changing direction	

TABLE S1. List of symbols used to describe the gREM

S2. SUPPLEMENTARY METHODS

S2.1. **Introduction.** This supplementary methods derives all the models used in the paper. For continuity, the gas model derivation is included here as well as in the main text. The calculation of all integrals is included in the Python script S3.

S3. Supplementary Script: Symbolic algebra Python Script

This script uses the SymPy package SymPy Development Team (2014), a computer algebra system to calculate the equations for p in the various models and to perform unit checks on the results.

```
"""
Systematic analysis of REM models
Tim Lucas
01/10/13
"""


from sympy import *
import numpy as np
import matplotlib.pyplot as pl
from datetime import datetime


# Use LaTeX printing
from sympy import init_printing ;
init_printing()
# Make LaTeX output white. Because I use a dark theme
init_printing(forecolor="White")


# Load symbols used for symbolic maths
t, a, r, x2, x3, x4, x1 = symbols('theta alpha r x_2 x_3 x_4 x_1', positive=True)
r1 = {r:1} # useful for lots of checks


# Define functions
# Calculate the final profile averaged over pi.
def calcModel(model):
        x = pi**-1 * sum( [integrate(m[0], m[1:]) for m in model] ).simplify().trigsimp()
        return x

# Do the replacements fit within the area defined by the conditions?
def confirmReplacements(conds, reps):
        if not all([c.subs(reps) for c in eval(conds)]):
                print('reps' + conds[4:] + ' incorrect')

# is average profile in range 0r-2r?
def profileRange(prof, reps):
        if not 0 <= eval(prof).subs(dict(reps, **r1)) <= 2:
                print('Total ' + prof + ' not in 0, 2r')

# Are the individuals integrals >0r
def intsPositive(model, reps):
        m = eval(model)
        for i in range(len(m)):
                if not integrate(m[i][0], m[i][1:]).subs(dict(reps, **r1)) > 0:
                    print('Integral ' + str(i+1) + ' in ' + model + ' is negative')

# Are the individual averaged integrals between 0 and  2r
def intsRange(model, reps):
        m = eval(model)
        for i in range(len(m)):
                if not 0 <= (integrate(m[i][0], m[i][1:])/(m[i][3]-m[i][2])).subs(dict(reps, **r1)) <= 2:
                        print('Integral ' + str(i+1) + ' in ' + model + ' has averaged integral outside 0<p<2r')

# Are the bounds the correct way around
def checkBounds(model, reps):
        m = eval(model)
        for i in range(len(m)):
                if not (m[i][3]-m[i][2]).subs(reps) > 0:
                        print('Bounds ' + str(i+1) + ' in ' + model + ' has lower bounds bigger than upper bounds')

# create latex strings with the 1) the integral equation that defines it and 2)  the final calculated model.
# There's some if statements to split longer equations on two lines and get +s in the right place.
def parseLaTeX(prof):
        m = eval( 'm' + prof[1:] )

        f = open('/home/tim/Dropbox/liz-paper/lucasMoorcroftManuscript/supplementary-material/latexFiles/'+prof+'.tex', 'w')
        f.write('\\begin{align}\n    \\bar{p}_{\\text{\\tiny{' + prof[1:] + '}}} =&\\frac{1}{\pi} \left(\;\;')
        for i in range(len(m)):
    # Roughly try and prevent expressions beginning with minus signs.
    if latex(m[i][2])[0]=='-':
      o1 = 'rev-lex'
    else:
      o1 = 'lex'

```

```
    if latex(m[i][3])[0]=='-':
      o2 = 'rev-lex'
    else:
      o2 = 'lex'

    if latex(m[i][0])[0]=='-':
      o3 = 'rev-lex'
    else:
      o3 = 'lex'

    if latex(m[i][1])[0]=='-':
      o4 = 'rev-lex'
    else:
      o4 = 'lex'

                f.write('\int\limits_{'+latex(m[i][2], order=o1)+'}^{'+latex(m[i][3], order=o2)+'}'+
                    latex(m[i][0], order=o3)+'\;\mathrm{d}' +latex(m[i][1], order=o4))
                if len(m)>3 and i==(len(m)/2)-1:
                        f.write( '\\right.\\notag\\\\\n &\left.' )
                if i<len(m)-1:
                        f.write('+')
        f.write('\\right)\label{' + prof + 'Def}\\\\\n    ')
        f.write('\\bar{p}_{\\text{\\tiny{' + prof[1:] + '}}}  =& ' + latex(eval(prof)) + '\label{' +
            prof + 'Sln}\n\\end{align}')
        f.close()


# Apply all checks.
def allChecks(prof):
        model = 'm' + prof[1:]
        reps = eval('rep' + prof[1:])
        conds = 'cond' + prof[1:]
        confirmReplacements(conds, reps)
        profileRange(prof, reps)
        intsPositive(model, reps)
        intsRange(model, reps)
        checkBounds(model, reps)

#####################################################
### Define and solve all models                   ###
#####################################################

# NE1 animal: a = 2*pi.  sensor: t > pi, a > 3pi - t  #

mNE1 = [ [2*r,                 x1, pi/2, t/2        ],
         [r + r*cos(x1 - t/2), x1, t/2,  pi         ],
         [r + r*cos(x1 + t/2), x1, pi,   2*pi-t/2   ],
         [2*r,                 x1, 2*pi-t/2, 3*pi/2 ] ]

# Replacement values in range
repNE1 = {t:3*pi/2, a:2*pi}

# Define conditions for model
condNE1 = [pi <= t, a >= 3*pi - t]

# Calculate model, run checks, write output.
pNE1 = calcModel(mNE1)
allChecks('pNE1')
parseLaTeX('pNE1')


# NE2 animal: a > pi.  sensor: t > pi Condition: a < 3pi - t, a > 4pi - 2t  #

mNE2 = [ [2*r,                 x1, pi/2, t/2        ],
         [r + r*cos(x1 - t/2), x1, t/2,  5*pi/2 - t/2 - a/2 ],
         [r + r*cos(x1 + t/2), x1, 5*pi/2 - t/2 - a/2,   2*pi-t/2 ],
         [2*r,                 x1, 2*pi-t/2, 3*pi/2 ] ]

# Replacement values in range
repNE2 = {t:5*pi/3, a:4*pi/3-0.1}

# Define conditions for model
condNE2 = [pi <= t, a >= pi, a <= 3*pi - t, a >= 4*pi - 2*t]

# Calculate model, run checks, write output.
pNE2 = calcModel(mNE2)
allChecks('pNE2')
parseLaTeX('pNE2')


# NE3 animal: a > pi.  sensor: t > pi Condition: a < 4pi - 2t  #

mNE3 = [ [2*r,                 x1, pi/2, t/2        ],
         [r + r*cos(x1 - t/2), x1, t/2,  t/2 + pi/2         ],
         [r                  , x1, t/2 + pi/2,   5*pi/2 - t/2 - a/2 ],
         [r + r*cos(x1 + t/2), x1, 5*pi/2 - t/2 - a/2,   2*pi-t/2 ],
         [2*r,                 x1, 2*pi-t/2, 3*pi/2 ] ]
```

```

# Replacement values in range
repNE3 = {t:5*pi/4-0.1, a:3*pi/2}

# Define conditions for model
condNE3 = [pi <= t, a >= pi, a <= 4*pi - 2*t]

# Calculate model, run checks, write output.
pNE3 = calcModel(mNE3)
allChecks('pNE3')
parseLaTeX('pNE3')


# NW1 animal: a = 2*pi.   sensor:  pi/2 <= t <= pi      #

mNW1 = [ [2*r*sin(t/2)*sin(x2), x2, t/2,      pi/2     ],
        [r - r*cos(x4 - t),     x4, 0,        t - pi/2 ],
        [r,                     x4, t - pi/2, pi/2     ],
        [r - r*cos(x4),         x4, pi/2,     t        ],
        [2*r*sin(t/2)*sin(x2),  x2, t/2,      pi/2     ] ]

# Replacement values in range
repNW1 = {t:3*pi/4}

# Define conditions for model
condNW1 = [pi/2 <= t, t <= pi]

# Calculate model, run checks, write output.
pNW1 = calcModel(mNW1)
allChecks('pNW1')
parseLaTeX('pNW1')




# NW2 animal: a > pi.  Sensor: pi/2 <= t <= pi. Condition: a > 2pi - t          #

mNW2 = [ [2*r*sin(t/2)*sin(x2), x2, t/2,          pi/2        ],
         [r - r*cos(x4 - t),    x4, 0,            t - pi/2    ],
         [r,                    x4, t - pi/2,     3*pi/2 - a/2],
         [r - r*cos(x4),        x4, 3*pi/2 - a/2, t           ],
         [2*r*sin(t/2)*sin(x2), x2, t/2,          pi/2        ] ]


repNW2 = {t:3*pi/4, a:15*pi/8} # Replacement values in range

# Define conditions for model
condNW2 = [a > pi, pi/2 <= t, t <= pi, a >= 3*pi - 2*t]

# Calculate model, run checks, write output.
pNW2 = calcModel(mNW2)
allChecks('pNW2')
parseLaTeX('pNW2')



# NW3 animal: a > pi.  Sensor: pi/2 <= t <= pi. Cond: 2pi - t < a < 3pi - 2t    #

mNW3 = [ [2*r*sin(t/2)*sin(x2), x2, t/2,                pi/2              ],
         [r - r*cos(x4 - t),    x4, 0,                  t - pi/2          ],
         [r,                    x4, t - pi/2,           t                 ],
         [r*cos(x2 - t/2),      x2, t/2,                3*pi/2 - a/2 - t/2],
         [2*r*sin(t/2)*sin(x2), x2, 3*pi/2 - a/2 - t/2, pi/2              ] ]


repNW3 = {t:5*pi/8, a:6*pi/4} # Replacement values in range

# Define conditions for model
condNW3 = [a > pi, pi/2 <= t, t <= pi, 2*pi - t <= a, a <= 3*pi - 2*t]

# Calculate model, run checks, write output.
pNW3 = calcModel(mNW3)
allChecks('pNW3')
parseLaTeX('pNW3')



# NW4 animal:  a > pi.  Sensor: pi/2 <= t <= pi. Condition: a <= 2pi - t      #

mNW4 = [ [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2],
         [r - r*cos(x4 - t),    x4, 0, t - pi/2],
         [r,                    x4, t - pi/2, t],
         [r*cos(x2 - t/2),      x2, t/2, a/2 + t/2 - pi/2] ]

repNW4 = {t:3*pi/4, a:9*pi/8} # Replacement values in range

# Define conditions for model
```

```
condNW4 = [a > pi,  pi/2 <= t, t <= pi, a <= 2*pi - t]

# Calculate model, run checks, write output.
pNW4 = calcModel(mNW4)
allChecks('pNW4')
parseLaTeX('pNW4')


# REM animal: a=2pi.  Sensor: t <= pi/2.                                    #

mREM = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
         [r*sin(x3),            x3, t,          pi/2],
         [r,                    x4, 0*t,          t],
         [r*sin(x3),            x3, t,          pi/2],
         [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2] ]


repREM = {t:3*pi/8, a:2*pi} # Replacement values in range

# Define conditions for model
condREM = [ t <= pi/2  ]

# Calculate model, run checks, write output.
pREM = calcModel(mREM)
allChecks('pREM')
parseLaTeX('pREM')



# NW5 animal: a>pi.  Sensor: t <= pi/2. Condition: 2*pi - t < a             #


mNW5 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
         [r*sin(x3),            x3, t,          pi/2],
         [r,                    x4, 0,          t],
         [r*sin(x3),            x3, t,          pi/2],
         [r*cos(x2 - t/2),      x2, pi/2 - t/2, 3*pi/2 - t/2 - a/2],
         [2*r*sin(t/2)*sin(x2), x2, 3*pi/2 - t/2 - a/2, pi/2] ]


repNW5 = {t:3*pi/8, a:29*pi/16} # Replacement values in range

# Define conditions for model
condNW5 = [a >= pi, t <= pi/2, 2*pi - t <= a  ]

# Calculate model, run checks, write output.
pNW5 = calcModel(mNW5)
allChecks('pNW5')
parseLaTeX('pNW5')


# NW6 animal: a>pi.  Sensor: t <= pi/2. Condition:  2*pi - 2*t <= a <= 2*pi - t  #


mNW6 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
         [r*sin(x3),            x3, t,          pi/2],
         [r,                    x4, 0,          t],
         [r*sin(x3),            x3, t,          pi/2],
         [r*cos(x2 - t/2),      x2, pi/2 - t/2, a/2 + t/2 - pi/2] ]

repNW6 = {t:3*pi/8, a:3*pi/2} # Replacement values in range

# Define conditions for model
condNW6 = [a >= pi, t <= pi/2, 2*pi - 2*t <= a, a <= 2*pi - t]

# Calculate model, run checks, write output.
pNW6 = calcModel(mNW6)
allChecks('pNW6')
parseLaTeX('pNW6')



# NW7 animal: a>pi.  Sensor: t <= pi/2. Condition: a <= 2pi - 2t  #


mNW7 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
         [r*sin(x3),            x3, t,          pi/2],
         [r,                    x4, 0,          t   ],
         [r*sin(x3),            x3, pi - a/2,   pi/2] ]


repNW7 = {t:pi/9, a:10*pi/9} # Replacement values in range

# Define conditions for model
condNW7 = [t <= pi/2, a >= pi, a <= 2*pi - 2*t]

# Calculate model, run checks, write output.
```

```
pNW7 = calcModel(mNW7)
allChecks('pNW7')
parseLaTeX('pNW7')



# SE1 animal: a <= pi.  Sensor: t =2pi.            #

mSE1 = [ [ 2*r*sin(a/2),x1, pi/2, 3*pi/2       ],
         ]


repSE1 = {a:pi/4} # Replacement values in range

# Define conditions for model
condSE1 = [a <= pi]

# Calculate model, run checks, write output.
pSE1 = calcModel(mSE1)
allChecks('pSE1')
parseLaTeX('pSE1')




# SE2 animal: a <= pi.  Sensor: t > pi. Condition: a > 2pi - t, a > 4pi - 2t    #

mSE2 = [ [ 2*r*sin(a/2),                        x1, pi/2,               t/2 + pi/2 - a/2       ],
         [ r*sin(a/2) + r*cos(x1 - t/2),        x1, t/2 + pi/2 - a/2,   5*pi/2 - a/2 - t/2 ],
         [ 2*r*sin(a/2),                        x1, 5*pi/2 - a/2 - t/2, 3*pi/2]  ]


repSE2 = {t:19*pi/10, a:pi/2} # Replacement values in range

# Define conditions for model
condSE2 = [a <= pi, t >= pi,  a >= 4*pi - 2*t]

# Calculate model, run checks, write output.
pSE2 = calcModel(mSE2)
allChecks('pSE2')
parseLaTeX('pSE2')


# SE3 animal: a <= pi.  Sensor: t > pi. Condition: 2pi - t < a < 4pi - 2t #

mSE3 = [ [ 2*r*sin(a/2),                        x1, pi/2,               t/2 + pi/2 - a/2  ],
         [ r*sin(a/2) + r*cos(x1 - t/2),        x1, t/2 + pi/2 - a/2,   t/2 + pi/2        ],
         [ r*sin(a/2),                          x1, t/2 + pi/2,         5*pi/2 - a/2 - t/2],
         [ 2*r*sin(a/2),                        x1, 5*pi/2 - a/2 - t/2, 3*pi/2            ] ]

repSE3 = {t:3*pi/2 + 0.1, a:pi/2} # Replacement values in range

# Define conditions for model
condSE3 = [a <= pi, t >= pi,  a >= 2*pi - t, a <= 4*pi - 2*t]

# Calculate model, run checks, write output.
pSE3 = calcModel(mSE3)
allChecks('pSE3')
parseLaTeX('pSE3')


# SE4 animal: a <= pi.  Sensor: t > pi. Condition: a <= 4*pi - 2*t and a < 2*pi - t #


mSE4 = [ [ 2*r*sin(a/2),                       x1, pi/2,             t/2 + pi/2 - a/2  ],
         [ r*sin(a/2) + r*cos(x1 - t/2),       x1, t/2 + pi/2 - a/2, t/2 + pi/2        ],
         [ r*sin(a/2),                         x1, t/2 + pi/2,       t/2 + pi/2 + a/2  ] ]


repSE4 = {t:3*pi/2, a:pi/3} # Replacement values in range


# Define conditions for model
condSE4 = [a <= pi, t >= pi/2, a <= 4*pi - 2*t , a <= 2*pi - t]

# Calculate model, run checks, write output.
pSE4 = calcModel(mSE4)
allChecks('pSE4')
parseLaTeX('pSE4')


# SW1 animal: a <= pi.  Sensor: pi/2 <= t <= pi. Condition: a >= t and a/2 >= t - pi/2 #

mSW1 =  [ [2*r*sin(t/2)*sin(x2),              x2, pi/2 - a/2 + t/2, pi/2             ],
          [r*sin(a/2) - r*cos(x2 + t/2),      x2, t/2,              pi/2 - a/2 + t/2],
          [r*sin(a/2) - r*cos(x4 - t),        x4, 0,                t - pi/2         ],
          [r*sin(a/2),                        x4, t-pi/2,           t - pi/2 + a/2  ] ]
```

```


repSW1 = {t:5*pi/8, a:6*pi/8} # Replacement values in range

# Define conditions for model
condSW1 = [a <= pi, pi/2 <= t, t <= pi, a >= t, a/2 >= t - pi/2]

# Calculate model, run checks, write output.
pSW1 = calcModel(mSW1)
allChecks('pSW1')
parseLaTeX('pSW1')


# SW2 animal: a <= pi.  Sensor: pi/2 <= t <= pi. Condition: a <= t and a/2 >= t- pi/2 #

mSW2 =  [ [2*r*sin(a/2),                 x2, pi/2 + a/2 - t/2, pi/2             ],
          [r*sin(a/2) - r*cos(x2 + t/2), x2, t/2,              pi/2 + a/2 - t/2],
          [r*sin(a/2) - r*cos(x4 - t),   x4, 0*t,              t - pi/2       ],
          [r*sin(a/2),                   x4, t - pi/2,         t - pi/2 + a/2 ] ]


repSW2 = {t:7*pi/8, a:7*pi/8-0.1} # Replacement values in range

# Define conditions for model
condSW2 = [a <= pi, pi/2 <= t, t <= pi, a/2 <= t/2, a/2 >= t - pi/2]

# Calculate model, run checks, write output.
pSW2 = calcModel(mSW2)
allChecks('pSW2')
parseLaTeX('pSW2')



# SW3 animal: a <= pi.  Sensor: pi/2 <= t <= pi. Condition: a <= t and a/2 <= t- pi/2 #

mSW3 =  [ [2*r*sin(a/2),                     x2, t/2,             pi/2           ],
          [2*r*sin(a/2),                     x4, 0,               t - pi/2 - a/2 ],
          [r*sin(a/2) - r*cos(x4 - t),       x4, t - pi/2 - a/2, t - pi/2       ],
          [r*sin(a/2),                       x4, t - pi/2,        t - pi/2 + a/2 ] ]


repSW3 = {t:7*pi/8, a:2*pi/8} # Replacement values in range

# Define conditions for model
condSW3 = [a <= pi, pi/2 <= t, t <= pi, a/2 <= t/2, a/2 <= t - pi/2]

# Calculate model, run checks, write output.
pSW3 = calcModel(mSW3)
allChecks('pSW3')
parseLaTeX('pSW3')


# SW4 animal: a <= pi.  Sensor: t <= pi/2. Condition: a > pi - 2t &  a <= t      #

mSW4 = [ [2*r*sin(a/2),                 x2, pi/2 - t/2 + a/2, pi/2            ],
         [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2,       pi/2 - t/2 + a/2],
         [r*sin(a/2),                   x3, t,                pi/2            ],
         [r*sin(a/2),                   x4, 0,                a/2 + t - pi/2  ] ]

repSW4 = {t:pi/2-0.1, a:pi/4} # Replacement values in range

# Define conditions for model
condSW4 = [a <= pi,  t <= pi/2,  a >= pi - 2*t,  a <= t]

# Calculate model, run checks, write output.
pSW4 = calcModel(mSW4)
allChecks('pSW4')
parseLaTeX('pSW4')


# SW5 animal: a <= pi.  Sensor: t <= pi/2. Condition: a > pi - 2t &  t <= a <= 2t    #

mSW5 = [ [2*r*sin(t/2)*sin(x2),         x2, pi/2 + t/2 - a/2, pi/2            ],
         [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2,       pi/2 + t/2 - a/2],
         [r*sin(a/2),                   x3, t,                pi/2        ],
         [r*sin(a/2),                   x4, 0,                a/2 + t -pi/2   ] ]


repSW5 = {t:pi/2-0.1, a:pi/2} # Replacement values in range

# define conditions for model
condSW5 = [a <= pi,  t <= pi/2,  a >= pi - 2*t,  t <= a, a <= 2*t]


# Calculate model, run checks, write output.
pSW5 = calcModel(mSW5)
allChecks('pSW5')
```

```
parseLaTeX('pSW5')


# SW6 animal: a <= pi.  Sensor: t <= pi/2. Condition: a > pi - 2t &  a > 2t      #

mSW6 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2             ],
         [r*sin(x3),            x3, t,          a/2              ],
         [r*sin(a/2),           x3, a/2,        pi/2             ],
         [r*sin(a/2),           x4, 0,          a/2 + t -pi/2    ] ]


repSW6 = {t:pi/4, a:3*pi/4} # Replacement values in range


# Define conditions for model
condSW6 = [a <= pi,  t <= pi/2,  a >= pi - 2*t,  a > 2*t]

# Calculate model, run checks, write output.
pSW6 = calcModel(mSW6)
allChecks('pSW6')
parseLaTeX('pSW6')


# SW7 animal: a <= pi.  Sensor: t <= pi/2. Condition: a <= pi - 2t & a <= t     #

mSW7 = [ [2*r*sin(a/2),                x2, pi/2 - t/2 + a/2, pi/2             ],
         [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2,       pi/2 - t/2 + a/2],
         [r*sin(a/2),                x3, t,                t + a/2          ] ]


repSW7 = {t:2*pi/8, a:pi/8} # Replacement values in range

# Define conditions for model
condSW7 = [a <= pi, t <= pi/2, a <= pi - 2*t, a <= t]

# Calculate model, run checks, write output.
pSW7 = calcModel(mSW7)
allChecks('pSW7')
parseLaTeX('pSW7')


# SW8 animal: a <= pi.  Sensor: t <= pi/2. Condition: a <= pi - 2t & t <= a <= 2t   #

mSW8 = [ [2*r*sin(t/2)*sin(x2),        x2, pi/2 + t/2 - a/2, pi/2             ],
         [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2,       pi/2 + t/2 - a/2],
         [r*sin(a/2),                x3, t,                t + a/2          ] ]

repSW8 = {t:2*pi/8, a:pi/2-0.1} # Replacement values in range

# Define conditions for model
condSW8 = [a <= pi, t <= pi/2, a <= pi - 2*t, t <= a, a <= 2*t]

# Calculate model, run checks, write output.
pSW8 = calcModel(mSW8)
allChecks('pSW8')
parseLaTeX('pSW8')


# SW9 animal: a <= pi.  Sensor: t <= pi/2. Condition: a <= pi - 2t &  2t <= a      #

mSW9 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2    ],
         [r*sin(x3),            x3, t,          a/2     ],
         [r*sin(a/2),           x3, a/2,        t + a/2 ] ]


repSW9 = {t:1*pi/8, a:pi/2} # Replacement values in range

# Define conditions for model
condSW9 = [a <= pi,  t <= pi/2,  a <= pi - 2*t,  2*t <= a]

# Calculate model, run checks, write output.
pSW9 = calcModel(mSW9)
allChecks('pSW9')
parseLaTeX('pSW9')


###################
## Run tests     ###
###################

# create gas model object
gas = 2*r


# for each model run through every adjacent model.
# Contains duplicatea but better for avoiding missed comparisons.
# Also contains replacement t->a and a->t just in case.
```

```


allComps = [
['gas', 'pNE1', {t:2*pi}], ['gas', 'pSE1', {a:pi}],

['pNE1', 'gas', {t:2*pi}], ['pNE1', 'pNW1', {t:pi}],
['pNE1', 'pNE2',{a:3*pi-t}], ['pNE1', 'pNE2',{t:3*pi-a}],

['pNE2', 'pNE1',{a:3*pi-t}], ['pNE2', 'pNE1',{t:3*pi-a}],
['pNE2', 'pNE3',{a:4*pi-2*t}], ['pNE2', 'pNE3',{t:2*pi-a/2}],
['pNE2', 'pSE2',{a:pi}],

['pNE3', 'pNE2',{a:4*pi-2*t}], ['pNE3', 'pNE2',{t:2*pi-a/2}],
['pNE3', 'pSE3',{a:pi}], ['pNE3', 'pNW2',{t:pi}],

['pNW1','pNE1', {t:pi}], ['pNW1','pNW2',{a:2*pi}],

['pNW2','pNE3',{t:pi}], ['pNW2','pNW3',{a:3*pi-2*t}],
['pNW2','pNW3',{t:3*pi/2-a/2}], ['pNW2','pNW1',{a:2*pi}],

['pNW3','pNW5',{t:pi/2}], ['pNW3','pNW4',{a:2*pi-t}],
['pNW3','pNW4',{t:2*pi-a}], ['pNW3','pNW2',{a:3*pi-2*t}],
['pNW3','pNW2',{t:3*pi/2-a/2}],

['pNW4','pNW6',{t:pi/2}], ['pNW4','pNW3',{t:2*pi-a}],
['pNW4','pNW3',{a:2*pi-t}], ['pNW4','pSW1',{a:pi}],

['pREM','pNW1', {t:pi/2}], ['pREM','pNW5',{a:2*pi}],

['pNW5','pREM',{a:2*pi}], ['pNW5','pNW6',{a:2*pi-t}],
['pNW5','pNW6',{t:2*pi-a}], ['pNW5','pNW3',{t:pi/2}],

['pNW6','pNW5',{a:2*pi-t}], ['pNW6','pNW5',{t:2*pi-a}],
['pNW6','pNW7',{t:pi-a/2}], ['pNW6','pNW7',{a:2*pi-2*t}],
['pNW5','pNW4',{t:pi/2}],

['pNW7','pNW6',{t:2*pi-2*a}], ['pNW7','pNW6',{a:2*pi-2*t}],
['pNW7','pSW6',{a:pi}],

['pSE1','pSE2',{t:2*pi}], ['pSE1','gas',{a:pi}],

['pSE2','pSE3',{t:2*pi-a/2}], ['pSE2','pSE3',{a:4*pi-2*t}],
['pSE2','pSE1',{t:2*pi}], ['pSE2','pNE2',{a:pi}],

['pSE3','pSE2',{a:4*pi-2*t}], ['pSE3','pSE2',{t:2*pi-a/2}],
['pSE3','pSE4',{a:2*pi-t}], ['pSE3','pSE4',{t:2*pi-a}],
['pSE3','pNE3',{a:pi}],

['pSE4','pSE3',{t:2*pi-a}], ['pSE4','pSE3',{a:2*pi-t}],
['pSE4','pSW3',{t:pi}],

['pSW1','pSW5',{t:pi/2}], ['pSW1','pSW2',{a:t}],
['pSW1','pSW2',{t:a}], ['pSW1','pNW4',{a:pi}],

['pSW2','pSW1',{a:t}], ['pSW2','pSW1',{t:a}],
['pSW2','pSW4',{t:pi/2}], ['pSW2','pSW3',{a:2*t-pi}],
['pSW2','pSW3',{t:a/2+pi/2}],

['pSW3','pSW2',{t:a/2+pi/2}], ['pSW3','pSW2',{a:2*t-pi}],
['pSW3','pSE4',{t:pi}],


['pSW4','pSW7',{a:pi-2*t}], ['pSW4','pSW7',{t:pi/2-a/2}],
['pSW4','pSW5',{t:a}], ['pSW4','pSW5',{a:t}],
['pSW4','pSW2',{t:pi/2}],

['pSW5','pSW4',{t:a}], ['pSW5','pSW4',{a:t}],
['pSW5','pSW8',{t:pi/2-a/2}], ['pSW5','pSW8',{a:pi-2*t}],
['pSW5','pSW6',{a:2*t}], ['pSW5','pSW6',{t:a/2}],
['pSW5','pSW1',{t:pi/2}],

['pSW6','pSW9',{t:pi/2-a/2}], ['pSW6','pSW9',{a:pi-2*t}],
['pSW6','pSW5',{a:2*t}], ['pSW6','pSW5',{t:a/2}],
['pSW6','pNW7',{a:pi}],


['pSW7','pSW8',{t:a}], ['pSW7','pSW8',{a:t}],
['pSW7','pSW4',{t:pi/2-a/2}], ['pSW7','pSW4',{a:pi-2*t}],

['pSW8','pSW7',{a:t}], ['pSW8','pSW7',{t:a}],
['pSW8','pSW9',{a:2*t}], ['pSW8','pSW9',{t:a/2}],
['pSW8','pSW5',{a:pi-2*t}], ['pSW8','pSW5',{t:pi/2-a/2}],

['pSW9','pSW8',{a:2*t}], ['pSW9','pSW8',{t:a/2}],
['pSW9','pSW6',{a:pi-2*t}], ['pSW9','pSW6',{t:pi/2-a/2}]
]

```

```

# List of regions that touch a=0. Should equal 0 when a=0.
zeroRegions = ['pSW9', 'pSW8', 'pSW7', 'pSW4', 'pSW2', 'pSW3', 'pSE4', 'pSE3', 'pSE2', 'pSE1']

# Run through all the comparisons. Need simplify(). Even together() gives some false negatives.

checkFile = open('/home/tim/Dropbox/phd/Analysis/REM-chapter/checksFile.tex','w')

checkFile.write('All checks evaluated.\nTim Lucas - ' + str(datetime.now()) + '\n')
for i in range(len(allComps)):
        if (eval(allComps[i][0]).subs(allComps[i][2]) - eval(allComps[i][1]).subs(allComps[i][2])).simplify() == 0:
                checkFile.write(str(i) + ': ' + allComps[i][0]+ ' and ' +allComps[i][1]+': OK\n')
        else:
                checkFile.write(str(i) + ': ' + allComps[i][0]+ ' and ' +allComps[i][1]+': Incorrect\n')

for i in range(len(zeroRegions)):
        if eval(zeroRegions[i]).subs({a:0}).simplify() == 0:
                checkFile.write(zeroRegions[i] + ' at a=0: OK\n')
        else:
                checkFile.write(zeroRegions[i] + ' at a=0: Incorrect\n')

checkFile.close()


# And print to terminal
#for i in range(len(allComps)):
#        if not (eval(allComps[i][0]).subs(allComps[i][2]) - eval(allComps[i][1]).subs(allComps[i][2])).simplify() == 0:
#                print allComps[i][0] + ' and ' + allComps[i][1]+': Incorrect\n'


######################################################################
### Define a a function that calculates p bar answer.             ####
######################################################################

def calcP(A, T, R):
  assert (A <= 2*pi and A >= 0), "a is out of bounds. Should be in 0<a<2*pi"
  assert (T <= 2*pi and T >= 0), "s is out of bounds. Should be in 0<s<2*pi"

  if A > pi:
    if A < 4*pi - 2* T:
      p = pNW7.subs({a:A,  t:T, r:R}).n()
    elif A <= 3*pi -  T:
                        p = pNE2.subs({a:A,  t:T, r:R}).n()
    else:
                        p = pNE1.subs({a:A,  t:T, r:R}).n()
  else:
    if A < 4*pi - 2* T:
                        p = pSE3.subs({a:A,  t:T, r:R}).n()
    else:
                        p = pSE2.subs({a:A,  t:T, r:R}).n()
        return p


#############################
## Apply to entire grid   ###
#############################

# How many values for each parameter
nParas = 100

# Make a vector for a and s. Make an empty nParas x nParas array.
# Calculated profile sizes will go in pArray
tVec = np.linspace(0, 2*pi, nParas)
aVec = np.linspace(0, 2*pi, nParas)
pArray = np.zeros((nParas,nParas))

# Calculate profile size for each combination of parameters
for i in range(nParas):
        for j in range(nParas):
                pArray[i][j] = calcP(aVec[i], tVec[j], 1)

# Turn the array upside down so origin is at bottom left.
pImage = np.flipud(pArray)

# Plot and save.
pl.imshow(pImage, interpolation='none', cmap=pl.get_cmap('Blues') )
#pl.show()

pl.savefig('/home/tim/Dropbox/phd/Analysis/REM-chapter/imgs/profilesCalculated.png')



############################
#### Output R function.  ###
############################
```

```

# To reduce mistakes, output R function directly from python.
# However, the if statements, which correspond to the bounds of each model, are not automatic.

Rfunc = open('/home/tim/Dropbox/phd/Analysis/REM-chapter/supplementaryRscript.R', 'w')

Rfunc.write("""
# Functions to calculate density.
#
# Tim C.D. Lucas, Elizabeth Moorcroft, Robin Freeman, Marcus J. Rowcliffe, Kate E. Jones.
#
# calcDensity is the main function to calculate density.
# It takes parameters z, alpha, theta, r, animalSpeed, t
# z - The number of camera/acoustic counts or captures.
# alpha - Call width in radians.
# theta - Sensor width in radians.
# r - Sensor range in metres.
# animalSpeed - Average animal speed in metres per second.
# t - Length of survey in sensor seconds i.e. number of sensors x survey duration.
#
# calcAbundance calculates abundance rather than density and requires an extra parameter
# area - In metres squared. The size of the region being examined.


# Internal function to calculate profile width as described in the text
calcProfileWidth <- function(alpha, theta, r){
        if(alpha > 2*pi | alpha < 0)
    stop('alpha is out of bounds. alpha should be in interval 0<a<2*pi')
        if(theta > 2*pi | theta < 0)
    stop('theta is out of bounds. theta should be in interval 0<a<2*pi')

  if(alpha > pi){
          if(alpha < 4*pi - 2*theta){
""" +
'           p <- ' + str(pNW7) +
'\n                } else if(alpha <= 3*pi - theta){'
'\n                        p <- ' + str(pNE2) +
'\n                } else {'
'\n                        p <- ' + str(pNE1) +
'\n                }'
'\n        } else {'
'\n         if(alpha < 4*pi - 2*theta){'
'\n                        p <- ' + str(pSE3) +
'\n     } else {'
'\n                        p <- ' + str(pSE2) +
'\n                }'
'\n        }'
'\n        return(p)'
'\n}' +
"""
# Calculate a population density. See above for units etc.
calcDensity <- function(z, alpha, theta, r, animalSpeed, t){
        # Check the parameters are suitable.
        if(z <= 0 | !is.numeric(z)) stop('Counts, z, must be a positive number.')
        if(animalSpeed <= 0 | !is.numeric(animalSpeed)) stop('animalSpeed must be a positive number.')
        if(t <= 0 | !is.numeric(t)) stop('Time, t, must be a positive number.')

        # Calculate profile width, then density.
        p <- calcProfileWidth(alpha, theta, r)
        D <- z/{animalSpeed*t*p}
        return(D)
}

# Calculate abundance rather than density.
calcAbundance <- function(z, alpha, theta, r, animalSpeed, t, area){
        if(area <= 0 | !is.numeric(area)) stop('Area must be a positive number')
        D <- calcDensity(z, alpha, theta, r, animalSpeed, t)
        A <- D*area
        return(A)
}
"""
)

Rfunc.close()
```

REM–Analysis.py

S4. Supplementary Script: R implementation of models

This is a simple implementation of the models derived in the paper in R (R Development Core Team, 2010). Once given the parameters θ and α it automatically selects the correct model to apply.

```r

# Functions to calculate density.
#
# Tim C.D. Lucas, Elizabeth Moorcroft, Robin Freeman, Marcus J. Rowcliffe, Kate E. Jones.
#
# calcDensity is the main function to calculate density.
# It takes parameters z, alpha, theta, r, animalSpeed, t
# z - The number of camera/acoustic counts or captures.
# alpha - Call width in radians.
# theta - Sensor width in radians.
# r - Sensor range in metres.
# animalSpeed - Average animal speed in metres per second.
# t - Length of survey in sensor seconds i.e. number of sensors x survey duration.
#
# calcAbundance calculates abundance rather than density and requires an extra parameter
# area - In metres squared. The size of the region being examined.


# Internal function to calculate profile width as described in the text
calcProfileWidth <- function(alpha, theta, r){
        if(alpha > 2*pi | alpha < 0)
    stop('alpha is out of bounds. alpha should be in interval 0<a<2*pi')
        if(theta > 2*pi | theta < 0)
    stop('theta is out of bounds. theta should be in interval 0<a<2*pi')

  if(alpha > pi){
          if(alpha < 4*pi - 2*theta){
            p <- r*(theta - cos(alpha/2) + 1)/pi
                } else if(alpha <= 3*pi - theta){
                        p <- r*(theta - cos(alpha/2) + cos(alpha/2 + theta))/pi
                } else {
                        p <- r*(theta + 2*sin(theta/2))/pi
                }
        } else {
          if(alpha < 4*pi - 2*theta){
                        p <- r*(theta*sin(alpha/2) - cos(alpha/2) + 1)/pi
    } else {
                        p <- r*(theta*sin(alpha/2) - cos(alpha/2) + cos(alpha/2 + theta))/pi
                }
        }
        return(p)
}
# Calculate a population density. See above for units etc.
calcDensity <- function(z, alpha, theta, r, animalSpeed, t){
        # Check the parameters are suitable.
        if(z <= 0 | !is.numeric(z)) stop('Counts, z, must be a positive number.')
        if(animalSpeed <= 0 | !is.numeric(animalSpeed)) stop('animalSpeed must be a positive number.')
        if(t <= 0 | !is.numeric(t)) stop('Time, t, must be a positive number.')

        # Calculate profile width, then density.
        p <- calcProfileWidth(alpha, theta, r)
        if(p <= 0) stop('Calculated profile width is 0. We would therefore expect 0 captures. If z is not zero, then the density is undefined.')
        D <- z/{animalSpeed*t*p}
        return(D)
}

# Calculate abundance rather than density.
calcAbundance <- function(z, alpha, theta, r, animalSpeed, t, area){
        if(area <= 0 | !is.numeric(area)) stop('Area must be a positive number')
        D <- calcDensity(z, alpha, theta, r, animalSpeed, t)
        A <- D*area
        return(A)
}
```

supplementaryRscript.R

References

R Development Core Team (2010) *R: A Language And Environment For Statistical Computing*. R Foundation For Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. 13

SymPy Development Team (2014) *SymPy: Python library for symbolic mathematics*. 3