

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

S1. TABLE OF SYMBOLS

Symbol	Description	Units
$v$	Velocity	$\text{m s}^{-1}$
$\theta$	Angle of detection	Radians
$\alpha$	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 $\text{m}^{-2}$
$x_i$	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	
$A$	Probability 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.

```

1  """
2  Systematic analysis of REM models
3  Tim Lucas
4  01/10/13
5  """
6
7
8  from sympy import *
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from datetime import datetime
12
13
14 # Use LaTeX printing
15 from sympy import init_printing ;
16 init_printing()
17 # Make LaTeX output white. Because I use a dark theme
18 init_printing(forecolor="White")
19
20
21 # Load symbols used for symbolic maths
22 t, a, r, x2, x3, x4, x1 = symbols('theta alpha r x_2 x_3 x_4 x_1', positive=True)
23 r1 = {r:1} # useful for lots of checks
24
25
26 # Define functions
27 # Calculate the final profile averaged over pi.
28 def calcModel(model):
29     x = pi**1 * sum( [integrate(m[0], m[1:]) for m in model] ).simplify().trigsimp()
30     return x
31
32 # Do the replacements fit within the area defined by the conditions?
33 def confirmReplacements(conds, reps):
34     if not all([c.subs(reps) for c in eval(conds)]):
35         print('reps' + conds[4:] + ' incorrect')
36
37 # is average profile in range 0r-2r?
38 def profileRange(prof, reps):
39     if not 0 <= eval(prof).subs(dict(reps, **r1)) <= 2:
40         print('Total ' + prof + ' not in 0, 2r')
41
42 # Are the individuals integrals >0r
43 def intsPositive(model, reps):
44     m = eval(model)
45     for i in range(len(m)):
46         if not integrate(m[i][0], m[i][1:]).subs(dict(reps, **r1)) > 0:
47             print('Integral ' + str(i+1) + ' in ' + model + ' is negative')
48
49 # Are the individual averaged integrals between 0 and 2r
50 def intsRange(model, reps):
51     m = eval(model)
52     for i in range(len(m)):
53         if not 0 <= (integrate(m[i][0], m[i][1:])/(m[i][3]-m[i][2])).subs(dict(reps, **r1)) <=
54             2:
55             print('Integral ' + str(i+1) + ' in ' + model + ' has averaged integral outside
56                 0<p<2r')
57
58 # Are the bounds the correct way around
59 def checkBounds(model, reps):
60     m = eval(model)
61     for i in range(len(m)):
62         if not (m[i][3]-m[i][2]).subs(reps) > 0:
63             print('Bounds ' + str(i+1) + ' in ' + model + ' has lower bounds bigger than
64                 upper bounds')
65
66 # create latex strings with the 1) the integral equation that defines it and 2) the final calculated
67 model.
68 # There's some if statements to split longer equations on two lines and get +s in the right place.
69 def parseLaTeX(prof):
70     m = eval('m' + prof[1:] )
71
72     f = open('/home/tim/Dropbox/liz-paper/lucasMoorcroftManuscript/supplementary-material/latexFiles
73             /'+prof+'.tex', 'w')
74     f.write('\begin{align}\n    \bar{p}_{\text{\tiny' + prof[1:] + '}} = &\frac{1}{\pi} \left
75         (\frac{1}{\pi} \int_{\text{\tiny' + prof[1:] + '}}
76
77     for i in range(len(m)):
78         # Roughly try and prevent expressions beginning with minus signs.
79         if latex(m[i][2])[0]=='-':
80             o1 = 'rev-lex'
81         else:
82             o1 = 'lex'
83

```

```

77     if latex(m[i][3])[0]=='-':
78         o2 = 'rev-lex'
79     else:
80         o2 = 'lex'
81
82     if latex(m[i][0])[0]=='-':
83         o3 = 'rev-lex'
84     else:
85         o3 = 'lex'
86
87     if latex(m[i][1])[0]=='-':
88         o4 = 'rev-lex'
89     else:
90         o4 = 'lex'
91
92     f.write('\int\limits_{'+latex(m[i][2], order=o1)+'}^'+latex(m[i][3], order=o2)+''+
93           latex(m[i][0], order=o3)+'\;\mathrm{d}' + latex(m[i][1], order=o4))
94     if len(m)>3 and i==(len(m)/2)-1:
95         f.write( '\right.\notag\\\n &\left.' )
96     if i<len(m)-1:
97         f.write('+')
98     f.write('\right)\label{' + prof + 'Def}\n')
99     f.write('\bar{p}_{\text{\tiny{' + prof[1:] + '}}} =&' + latex(eval(prof)) + '\label{' +
100           prof + 'Sln}\n\end{align}')
101     f.close()
102
103 # Apply all checks.
104 def allChecks(prof):
105     model = 'm' + prof[1:]
106     reps = eval('rep' + prof[1:])
107     conds = 'cond' + prof[1:]
108     confirmReplacements(conds, reps)
109     profileRange(prof, reps)
110     intsPositive(model, reps)
111     intsRange(model, reps)
112     checkBounds(model, reps)
113
114 #####
115 ### Define and solve all models ###
116 #####
117 # NE1 animal: a = 2*pi. sensor: t > pi, a > 3pi - t #
118
119 mNE1 = [ [2*r, x1, pi/2, t/2 ],
120          [r + r*cos(x1 - t/2), x1, t/2, pi ],
121          [r + r*cos(x1 + t/2), x1, pi, 2*pi-t/2 ],
122          [2*r, x1, 2*pi-t/2, 3*pi/2 ] ]
123
124 # Replacement values in range
125 repNE1 = {t:3*pi/2, a:2*pi}
126
127 # Define conditions for model
128 condNE1 = [pi <= t, a >= 3*pi - t]
129
130 # Calculate model, run checks, write output.
131 pNE1 = calcModel(mNE1)
132 allChecks('pNE1')
133 parseLaTeX('pNE1')
134
135 # NE2 animal: a > pi. sensor: t > pi Condition: a < 3pi - t, a > 4pi - 2t #
136
137 mNE2 = [ [2*r, x1, pi/2, t/2 ],
138          [r + r*cos(x1 - t/2), x1, t/2, 5*pi/2 - t/2 - a/2 ],
139          [r + r*cos(x1 + t/2), x1, 5*pi/2 - t/2 - a/2, 2*pi-t/2 ],
140          [2*r, x1, 2*pi-t/2, 3*pi/2 ] ]
141
142 # Replacement values in range
143 repNE2 = {t:5*pi/3, a:4*pi/3-0.1}
144
145 # Define conditions for model
146 condNE2 = [pi <= t, a >= pi, a <= 3*pi - t, a >= 4*pi - 2*t]
147
148 # Calculate model, run checks, write output.
149 pNE2 = calcModel(mNE2)
150 allChecks('pNE2')
151 parseLaTeX('pNE2')
152
153 # NE3 animal: a > pi. sensor: t > pi Condition: a < 4pi - 2t #
154
155 mNE3 = [ [2*r, x1, pi/2, t/2 ],
156          [r + r*cos(x1 - t/2), x1, t/2, t/2 + pi/2 ],
157          [r, x1, t/2 + pi/2, 5*pi/2 - t/2 - a/2 ],
158          [r + r*cos(x1 + t/2), x1, 5*pi/2 - t/2 - a/2, 2*pi-t/2 ],
159          [2*r, x1, 2*pi-t/2, 3*pi/2 ] ]

```

```

162
163 # Replacement values in range
164 repNE3 = {t:5*pi/4-0.1, a:3*pi/2}
165
166 # Define conditions for model
167 condNE3 = [pi <= t, a >= pi, a <= 4*pi - 2*t]
168
169 # Calculate model, run checks, write output.
170 pNE3 = calcModel(mNE3)
171 allChecks('pNE3')
172 parseLaTeX('pNE3')
173
174
175 # NW1 animal: a = 2*pi.   sensor: pi/2 <= t <= pi   #
176
177 mNW1 = [ [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2 ],
178          [r - r*cos(x4 - t), x4, 0, t - pi/2 ],
179          [r, x4, t - pi/2, pi/2 ],
180          [r - r*cos(x4), x4, pi/2, t ],
181          [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2 ] ]
182
183 # Replacement values in range
184 repNW1 = {t:3*pi/4}
185
186 # Define conditions for model
187 condNW1 = [pi/2 <= t, t <= pi]
188
189 # Calculate model, run checks, write output.
190 pNW1 = calcModel(mNW1)
191 allChecks('pNW1')
192 parseLaTeX('pNW1')
193
194
195
196
197 # NW2 animal: a > pi.   Sensor: pi/2 <= t <= pi. Condition: a > 2pi - t   #
198
199 mNW2 = [ [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2 ],
200          [r - r*cos(x4 - t), x4, 0, t - pi/2 ],
201          [r, x4, t - pi/2, 3*pi/2 - a/2],
202          [r - r*cos(x4), x4, 3*pi/2 - a/2, t ],
203          [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2 ] ]
204
205
206 repNW2 = {t:3*pi/4, a:15*pi/8} # Replacement values in range
207
208 # Define conditions for model
209 condNW2 = [a > pi, pi/2 <= t, t <= pi, a >= 3*pi - 2*t]
210
211 # Calculate model, run checks, write output.
212 pNW2 = calcModel(mNW2)
213 allChecks('pNW2')
214 parseLaTeX('pNW2')
215
216
217
218 # NW3 animal: a > pi.   Sensor: pi/2 <= t <= pi. Cond: 2pi - t < a < 3pi - 2t   #
219
220 mNW3 = [ [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2 ],
221          [r - r*cos(x4 - t), x4, 0, t - pi/2 ],
222          [r, x4, t - pi/2, t ],
223          [r*cos(x2 - t/2), x2, t/2, 3*pi/2 - a/2 - t/2],
224          [2*r*sin(t/2)*sin(x2), x2, 3*pi/2 - a/2 - t/2, pi/2 ] ]
225
226
227 repNW3 = {t:5*pi/8, a:6*pi/4} # Replacement values in range
228
229 # Define conditions for model
230 condNW3 = [a > pi, pi/2 <= t, t <= pi, 2*pi - t <= a, a <= 3*pi - 2*t]
231
232 # Calculate model, run checks, write output.
233 pNW3 = calcModel(mNW3)
234 allChecks('pNW3')
235 parseLaTeX('pNW3')
236
237
238
239 # NW4 animal: a > pi.   Sensor: pi/2 <= t <= pi. Condition: a <= 2pi - t   #
240
241 mNW4 = [ [2*r*sin(t/2)*sin(x2), x2, t/2, pi/2],
242          [r - r*cos(x4 - t), x4, 0, t - pi/2],
243          [r, x4, t - pi/2, t],
244          [r*cos(x2 - t/2), x2, t/2, a/2 + t/2 - pi/2] ]
245
246 repNW4 = {t:3*pi/4, a:9*pi/8} # Replacement values in range
247
248 # Define conditions for model

```

```

249 condNW4 = [a > pi, pi/2 <= t, t <= pi, a <= 2*pi - t]
250
251 # Calculate model, run checks, write output.
252 pNW4 = calcModel(mNW4)
253 allChecks('pNW4')
254 parseLaTeX('pNW4')
255
256
257 # REM animal: a=2pi. Sensor: t <= pi/2. #
258
259 mREM = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
260          [r*sin(x3), x3, t, pi/2],
261          [r, x4, 0*t, t],
262          [r*sin(x3), x3, t, pi/2],
263          [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2] ]
264
265
266 repREM = {t:3*pi/8, a:2*pi} # Replacement values in range
267
268 # Define conditions for model
269 condREM = [ t <= pi/2 ]
270
271 # Calculate model, run checks, write output.
272 pREM = calcModel(mREM)
273 allChecks('pREM')
274 parseLaTeX('pREM')
275
276
277
278 # NW5 animal: a>pi. Sensor: t <= pi/2. Condition: 2*pi - t < a #
279
280
281 mNW5 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
282          [r*sin(x3), x3, t, pi/2],
283          [r, x4, 0, t],
284          [r*sin(x3), x3, t, pi/2],
285          [r*cos(x2 - t/2), x2, pi/2 - t/2, 3*pi/2 - t/2 - a/2],
286          [2*r*sin(t/2)*sin(x2), x2, 3*pi/2 - t/2 - a/2, pi/2] ]
287
288
289 repNW5 = {t:3*pi/8, a:29*pi/16} # Replacement values in range
290
291 # Define conditions for model
292 condNW5 = [a >= pi, t <= pi/2, 2*pi - t <= a ]
293
294 # Calculate model, run checks, write output.
295 pNW5 = calcModel(mNW5)
296 allChecks('pNW5')
297 parseLaTeX('pNW5')
298
299
300 # NW6 animal: a>pi. Sensor: t <= pi/2. Condition: 2*pi - 2*t <= a <= 2*pi - t #
301
302
303 mNW6 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
304          [r*sin(x3), x3, t, pi/2],
305          [r, x4, 0, t],
306          [r*sin(x3), x3, t, pi/2],
307          [r*cos(x2 - t/2), x2, pi/2 - t/2, a/2 + t/2 - pi/2] ]
308
309 repNW6 = {t:3*pi/8, a:3*pi/2} # Replacement values in range
310
311 # Define conditions for model
312 condNW6 = [a >= pi, t <= pi/2, 2*pi - 2*t <= a, a <= 2*pi - t]
313
314 # Calculate model, run checks, write output.
315 pNW6 = calcModel(mNW6)
316 allChecks('pNW6')
317 parseLaTeX('pNW6')
318
319
320
321 # NW7 animal: a>pi. Sensor: t <= pi/2. Condition: a <= 2pi - 2t #
322
323
324 mNW7 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2],
325          [r*sin(x3), x3, t, pi/2],
326          [r, x4, 0, t],
327          [r*sin(x3), x3, pi - a/2, pi/2] ]
328
329
330 repNW7 = {t:pi/9, a:10*pi/9} # Replacement values in range
331
332 # Define conditions for model
333 condNW7 = [t <= pi/2, a >= pi, a <= 2*pi - 2*t]
334
335 # Calculate model, run checks, write output.

```

```

336 pNW7 = calcModel(mNW7)
337 allChecks('pNW7')
338 parseLaTeX('pNW7')
339
340
341
342 # SE1 animal: a <= pi. Sensor: t = 2pi. #
343
344 mSE1 = [ [ 2*r*sin(a/2), x1, pi/2, 3*pi/2 ],
345          ]
346
347
348 repSE1 = {a:pi/4} # Replacement values in range
349
350 # Define conditions for model
351 condSE1 = [a <= pi]
352
353 # Calculate model, run checks, write output.
354 pSE1 = calcModel(mSE1)
355 allChecks('pSE1')
356 parseLaTeX('pSE1')
357
358
359
360
361 # SE2 animal: a <= pi. Sensor: t > pi. Condition: a > 2pi - t, a > 4pi - 2t #
362
363 mSE2 = [ [ 2*r*sin(a/2), x1, pi/2, t/2 + pi/2 - a/2 ],
364          [ r*sin(a/2) + r*cos(x1 - t/2), x1, t/2 + pi/2 - a/2, 5*pi/2 - a/2 - t/2 ],
365          [ 2*r*sin(a/2), x1, 5*pi/2 - a/2 - t/2, 3*pi/2 ] ]
366
367
368 repSE2 = {t:19*pi/10, a:pi/2} # Replacement values in range
369
370 # Define conditions for model
371 condSE2 = [a <= pi, t >= pi, a >= 4*pi - 2*t]
372
373 # Calculate model, run checks, write output.
374 pSE2 = calcModel(mSE2)
375 allChecks('pSE2')
376 parseLaTeX('pSE2')
377
378
379 # SE3 animal: a <= pi. Sensor: t > pi. Condition: 2pi - t < a < 4pi - 2t #
380
381 mSE3 = [ [ 2*r*sin(a/2), x1, pi/2, t/2 + pi/2 - a/2 ],
382          [ r*sin(a/2) + r*cos(x1 - t/2), x1, t/2 + pi/2 - a/2, t/2 + pi/2 ],
383          [ r*sin(a/2), x1, t/2 + pi/2, 5*pi/2 - a/2 - t/2 ],
384          [ 2*r*sin(a/2), x1, 5*pi/2 - a/2 - t/2, 3*pi/2 ] ]
385
386 repSE3 = {t:3*pi/2 + 0.1, a:pi/2} # Replacement values in range
387
388 # Define conditions for model
389 condSE3 = [a <= pi, t >= pi, a >= 2*pi - t, a <= 4*pi - 2*t]
390
391 # Calculate model, run checks, write output.
392 pSE3 = calcModel(mSE3)
393 allChecks('pSE3')
394 parseLaTeX('pSE3')
395
396
397 # SE4 animal: a <= pi. Sensor: t > pi. Condition: a <= 4*pi - 2*t and a < 2*pi - t #
398
399
400 mSE4 = [ [ 2*r*sin(a/2), x1, pi/2, t/2 + pi/2 - a/2 ],
401          [ r*sin(a/2) + r*cos(x1 - t/2), x1, t/2 + pi/2 - a/2, t/2 + pi/2 ],
402          [ r*sin(a/2), x1, t/2 + pi/2, t/2 + pi/2 + a/2 ] ]
403
404
405 repSE4 = {t:3*pi/2, a:pi/3} # Replacement values in range
406
407 # Define conditions for model
408 condSE4 = [a <= pi, t >= pi/2, a <= 4*pi - 2*t, a <= 2*pi - t]
409
410 # Calculate model, run checks, write output.
411 pSE4 = calcModel(mSE4)
412 allChecks('pSE4')
413 parseLaTeX('pSE4')
414
415
416
417 # SW1 animal: a <= pi. Sensor: pi/2 <= t <= pi. Condition: a >= t and a/2 >= t - pi/2 #
418
419 mSW1 = [ [ 2*r*sin(t/2)*sin(x2), x2, pi/2 - a/2 + t/2, pi/2 ],
420          [ r*sin(a/2) - r*cos(x2 + t/2), x2, t/2, pi/2 - a/2 + t/2 ],
421          [ r*sin(a/2) - r*cos(x4 - t), x4, 0, t - pi/2 ],
422          [ r*sin(a/2), x4, t-pi/2, t - pi/2 + a/2 ] ]

```

```

423
424
425 repSW1 = {t:5*pi/8, a:6*pi/8} # Replacement values in range
426
427 # Define conditions for model
428 condSW1 = [a <= pi, pi/2 <= t, t <= pi, a >= t, a/2 >= t - pi/2]
429
430 # Calculate model, run checks, write output.
431 pSW1 = calcModel(mSW1)
432 allChecks('pSW1')
433 parseLaTeX('pSW1')
434
435
436 # SW2 animal: a <= pi. Sensor: pi/2 <= t <= pi. Condition: a <= t and a/2 >= t - pi/2 #
437
438 mSW2 = [ [2*r*sin(a/2), x2, pi/2 + a/2 - t/2, pi/2 ],
439          [r*sin(a/2) - r*cos(x2 + t/2), x2, t/2, pi/2 + a/2 - t/2],
440          [r*sin(a/2) - r*cos(x4 - t), x4, 0*t, t - pi/2 ],
441          [r*sin(a/2), x4, t - pi/2, t - pi/2 + a/2 ] ]
442
443
444 repSW2 = {t:7*pi/8, a:7*pi/8-0.1} # Replacement values in range
445
446 # Define conditions for model
447 condSW2 = [a <= pi, pi/2 <= t, t <= pi, a/2 <= t/2, a/2 >= t - pi/2]
448
449 # Calculate model, run checks, write output.
450 pSW2 = calcModel(mSW2)
451 allChecks('pSW2')
452 parseLaTeX('pSW2')
453
454
455 # SW3 animal: a <= pi. Sensor: pi/2 <= t <= pi. Condition: a <= t and a/2 <= t - pi/2 #
456
457 mSW3 = [ [2*r*sin(a/2), x2, t/2, pi/2 ],
458          [2*r*sin(a/2), x4, 0, t - pi/2 - a/2 ],
459          [r*sin(a/2) - r*cos(x4 - t), x4, t - pi/2 - a/2, t - pi/2 ],
460          [r*sin(a/2), x4, t - pi/2, t - pi/2 + a/2 ] ]
461
462
463 repSW3 = {t:7*pi/8, a:2*pi/8} # Replacement values in range
464
465 # Define conditions for model
466 condSW3 = [a <= pi, pi/2 <= t, t <= pi, a/2 <= t/2, a/2 <= t - pi/2]
467
468 # Calculate model, run checks, write output.
469 pSW3 = calcModel(mSW3)
470 allChecks('pSW3')
471 parseLaTeX('pSW3')
472
473
474 # SW4 animal: a <= pi. Sensor: t <= pi/2. Condition: a > pi - 2t & a <= t #
475
476 mSW4 = [ [2*r*sin(a/2), x2, pi/2 - t/2 + a/2, pi/2 ],
477          [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2, pi/2 - t/2 + a/2],
478          [r*sin(a/2), x3, t, pi/2 ],
479          [r*sin(a/2), x4, 0, a/2 + t - pi/2 ] ]
480
481
482 repSW4 = {t:pi/2-0.1, a:pi/4} # Replacement values in range
483
484 # Define conditions for model
485 condSW4 = [a <= pi, t <= pi/2, a >= pi - 2*t, a <= t]
486
487 # Calculate model, run checks, write output.
488 pSW4 = calcModel(mSW4)
489 allChecks('pSW4')
490 parseLaTeX('pSW4')
491
492
493 # SW5 animal: a <= pi. Sensor: t <= pi/2. Condition: a > pi - 2t & t <= a <= 2t #
494
495 mSW5 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 + t/2 - a/2, pi/2 ],
496          [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2, pi/2 + t/2 - a/2],
497          [r*sin(a/2), x3, t, pi/2 ],
498          [r*sin(a/2), x4, 0, a/2 + t - pi/2 ] ]
499
500
501 repSW5 = {t:pi/2-0.1, a:pi/2} # Replacement values in range
502
503 # define conditions for model
504 condSW5 = [a <= pi, t <= pi/2, a >= pi - 2*t, t <= a, a <= 2*t]
505
506 # Calculate model, run checks, write output.
507 pSW5 = calcModel(mSW5)
508 allChecks('pSW5')
509

```



```

510 parseLaTeX('pSW5')
511
512
513 # SW6 animal: a <= pi. Sensor: t <= pi/2. Condition: a > pi - 2t & a > 2t #
514
515 mSW6 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2 ],
516          [r*sin(x3), x3, t, a/2 ],
517          [r*sin(a/2), x3, a/2, pi/2 ],
518          [r*sin(a/2), x4, 0, a/2 + t -pi/2 ] ]
519
520
521 repSW6 = {t:pi/4, a:3*pi/4} # Replacement values in range
522
523
524 # Define conditions for model
525 condSW6 = [a <= pi, t <= pi/2, a >= pi - 2*t, a > 2*t]
526
527 # Calculate model, run checks, write output.
528 pSW6 = calcModel(mSW6)
529 allChecks('pSW6')
530 parseLaTeX('pSW6')
531
532
533 # SW7 animal: a <= pi. Sensor: t <= pi/2. Condition: a <= pi - 2t & a <= t #
534
535 mSW7 = [ [2*r*sin(a/2), x2, pi/2 - t/2 + a/2, pi/2 ],
536          [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2, pi/2 - t/2 + a/2],
537          [r*sin(a/2), x3, t, t + a/2 ] ]
538
539
540 repSW7 = {t:2*pi/8, a:pi/8} # Replacement values in range
541
542 # Define conditions for model
543 condSW7 = [a <= pi, t <= pi/2, a <= pi - 2*t, a <= t]
544
545 # Calculate model, run checks, write output.
546 pSW7 = calcModel(mSW7)
547 allChecks('pSW7')
548 parseLaTeX('pSW7')
549
550
551 # SW8 animal: a <= pi. Sensor: t <= pi/2. Condition: a <= pi - 2t & t <= a <= 2t #
552
553 mSW8 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 + t/2 - a/2, pi/2 ],
554          [r*sin(a/2) - r*cos(x2 + t/2), x2, pi/2 - t/2, pi/2 + t/2 - a/2],
555          [r*sin(a/2), x3, t, t + a/2 ] ]
556
557 repSW8 = {t:2*pi/8, a:pi/2-0.1} # Replacement values in range
558
559 # Define conditions for model
560 condSW8 = [a <= pi, t <= pi/2, a <= pi - 2*t, t <= a, a <= 2*t]
561
562 # Calculate model, run checks, write output.
563 pSW8 = calcModel(mSW8)
564 allChecks('pSW8')
565 parseLaTeX('pSW8')
566
567
568 # SW9 animal: a <= pi. Sensor: t <= pi/2. Condition: a <= pi - 2t & 2t <= a #
569
570 mSW9 = [ [2*r*sin(t/2)*sin(x2), x2, pi/2 - t/2, pi/2 ],
571          [r*sin(x3), x3, t, a/2 ],
572          [r*sin(a/2), x3, a/2, t + a/2 ] ]
573
574
575 repSW9 = {t:1*pi/8, a:pi/2} # Replacement values in range
576
577 # Define conditions for model
578 condSW9 = [a <= pi, t <= pi/2, a <= pi - 2*t, 2*t <= a]
579
580 # Calculate model, run checks, write output.
581 pSW9 = calcModel(mSW9)
582 allChecks('pSW9')
583 parseLaTeX('pSW9')
584
585
586 #####
587 ## Run tests ##
588 #####
589
590 # create gas model object
591 gas = 2*r
592
593
594 # for each model run through every adjacent model.
595 # Contains duplicatea but better for avoiding missed comparisons.
596 # Also contains replacement t->a and a->t just in case.

```

```

597
598
599 allComps = [
600 ['gas', 'pNE1', {t:2*pi}], ['gas', 'pSE1', {a:pi}],
601
602 ['pNE1', 'gas', {t:2*pi}], ['pNE1', 'pNW1', {t:pi}],
603 ['pNE1', 'pNE2', {a:3*pi-t}], ['pNE1', 'pNE2', {t:3*pi-a}],
604
605 ['pNE2', 'pNE1', {a:3*pi-t}], ['pNE2', 'pNE1', {t:3*pi-a}],
606 ['pNE2', 'pNE3', {a:4*pi-2*t}], ['pNE2', 'pNE3', {t:2*pi-a/2}],
607 ['pNE2', 'pSE2', {a:pi}],
608
609 ['pNE3', 'pNE2', {a:4*pi-2*t}], ['pNE3', 'pNE2', {t:2*pi-a/2}],
610 ['pNE3', 'pSE3', {a:pi}], ['pNE3', 'pNW2', {t:pi}],
611
612 ['pNW1', 'pNE1', {t:pi}], ['pNW1', 'pNW2', {a:2*pi}],
613
614 ['pNW2', 'pNE3', {t:pi}], ['pNW2', 'pNW3', {a:3*pi-2*t}],
615 ['pNW2', 'pNW3', {t:3*pi/2-a/2}], ['pNW2', 'pNW1', {a:2*pi}],
616
617 ['pNW3', 'pNW5', {t:pi/2}], ['pNW3', 'pNW4', {a:2*pi-t}],
618 ['pNW3', 'pNW4', {t:2*pi-a}], ['pNW3', 'pNW2', {a:3*pi-2*t}],
619 ['pNW3', 'pNW2', {t:3*pi/2-a/2}],
620
621 ['pNW4', 'pNW6', {t:pi/2}], ['pNW4', 'pNW3', {t:2*pi-a}],
622 ['pNW4', 'pNW3', {a:2*pi-t}], ['pNW4', 'pSW1', {a:pi}],
623
624 ['pREM', 'pNW1', {t:pi/2}], ['pREM', 'pNW5', {a:2*pi}],
625
626 ['pNW5', 'pREM', {a:2*pi}], ['pNW5', 'pNW6', {a:2*pi-t}],
627 ['pNW5', 'pNW6', {t:2*pi-a}], ['pNW5', 'pNW3', {t:pi/2}],
628
629 ['pNW6', 'pNW5', {a:2*pi-t}], ['pNW6', 'pNW5', {t:2*pi-a}],
630 ['pNW6', 'pNW7', {t:pi-a/2}], ['pNW6', 'pNW7', {a:2*pi-2*t}],
631 ['pNW5', 'pNW4', {t:pi/2}],
632
633 ['pNW7', 'pNW6', {t:2*pi-2*a}], ['pNW7', 'pNW6', {a:2*pi-2*t}],
634 ['pNW7', 'pSW6', {a:pi}],
635
636 ['pSE1', 'pSE2', {t:2*pi}], ['pSE1', 'gas', {a:pi}],
637
638 ['pSE2', 'pSE3', {t:2*pi-a/2}], ['pSE2', 'pSE3', {a:4*pi-2*t}],
639 ['pSE2', 'pSE1', {t:2*pi}], ['pSE2', 'pNE2', {a:pi}],
640
641 ['pSE3', 'pSE2', {a:4*pi-2*t}], ['pSE3', 'pSE2', {t:2*pi-a/2}],
642 ['pSE3', 'pSE4', {a:2*pi-t}], ['pSE3', 'pSE4', {t:2*pi-a}],
643 ['pSE3', 'pNE3', {a:pi}],
644
645 ['pSE4', 'pSE3', {t:2*pi-a}], ['pSE4', 'pSE3', {a:2*pi-t}],
646 ['pSE4', 'pSW3', {t:pi}],
647
648 ['pSW1', 'pSW5', {t:pi/2}], ['pSW1', 'pSW2', {a:t}],
649 ['pSW1', 'pSW2', {t:a}], ['pSW1', 'pNW4', {a:pi}],
650
651 ['pSW2', 'pSW1', {a:t}], ['pSW2', 'pSW1', {t:a}],
652 ['pSW2', 'pSW4', {t:pi/2}], ['pSW2', 'pSW3', {a:2*t-pi}],
653 ['pSW2', 'pSW3', {t:a/2+pi/2}],
654
655 ['pSW3', 'pSW2', {t:a/2+pi/2}], ['pSW3', 'pSW2', {a:2*t-pi}],
656 ['pSW3', 'pSE4', {t:pi}],
657
658
659 ['pSW4', 'pSW7', {a:pi-2*t}], ['pSW4', 'pSW7', {t:pi/2-a/2}],
660 ['pSW4', 'pSW5', {t:a}], ['pSW4', 'pSW5', {a:t}],
661 ['pSW4', 'pSW2', {t:pi/2}],
662
663 ['pSW5', 'pSW4', {t:a}], ['pSW5', 'pSW4', {a:t}],
664 ['pSW5', 'pSW8', {t:pi/2-a/2}], ['pSW5', 'pSW8', {a:pi-2*t}],
665 ['pSW5', 'pSW6', {a:2*t}], ['pSW5', 'pSW6', {t:a/2}],
666 ['pSW5', 'pSW1', {t:pi/2}],
667
668 ['pSW6', 'pSW9', {t:pi/2-a/2}], ['pSW6', 'pSW9', {a:pi-2*t}],
669 ['pSW6', 'pSW5', {a:2*t}], ['pSW6', 'pSW5', {t:a/2}],
670 ['pSW6', 'pNW7', {a:pi}],
671
672
673 ['pSW7', 'pSW8', {t:a}], ['pSW7', 'pSW8', {a:t}],
674 ['pSW7', 'pSW4', {t:pi/2-a/2}], ['pSW7', 'pSW4', {a:pi-2*t}],
675
676 ['pSW8', 'pSW7', {a:t}], ['pSW8', 'pSW7', {t:a}],
677 ['pSW8', 'pSW9', {a:2*t}], ['pSW8', 'pSW9', {t:a/2}],
678 ['pSW8', 'pSW5', {a:pi-2*t}], ['pSW8', 'pSW5', {t:pi/2-a/2}],
679
680 ['pSW9', 'pSW8', {a:2*t}], ['pSW9', 'pSW8', {t:a/2}],
681 ['pSW9', 'pSW6', {a:pi-2*t}], ['pSW9', 'pSW6', {t:pi/2-a/2}]
682 ]
683

```

```

684
685 # List of regions that touch a=0. Should equal 0 when a=0.
686 zeroRegions = ['pSW9', 'pSW8', 'pSW7', 'pSW4', 'pSW2', 'pSW3', 'pSE4', 'pSE3', 'pSE2', 'pSE1']
687
688 # Run through all the comparisons. Need simplify(). Even together() gives some false negatives.
689
690 checkFile = open('/home/tim/Dropbox/phd/Analysis/REM-chapter/checksFile.tex','w')
691
692 checkFile.write('All checks evaluated.\nTim Lucas - ' + str(datetime.now()) + '\n')
693 for i in range(len(allComps)):
694     if (eval(allComps[i][0]).subs(allComps[i][2]) - eval(allComps[i][1]).subs(allComps[i][2])):
695         simplify() == 0:
696             checkFile.write(str(i) + ': ' + allComps[i][0] + ' and ' + allComps[i][1] + ': OK\n')
697         else:
698             checkFile.write(str(i) + ': ' + allComps[i][0] + ' and ' + allComps[i][1] + ': Incorrect\n')
699
700 for i in range(len(zeroRegions)):
701     if eval(zeroRegions[i]).subs({a:0}).simplify() == 0:
702         checkFile.write(zeroRegions[i] + ' at a=0: OK\n')
703     else:
704         checkFile.write(zeroRegions[i] + ' at a=0: Incorrect\n')
705
706 checkFile.close()
707
708 # And print to terminal
709 #for i in range(len(allComps)):
710 #    if not (eval(allComps[i][0]).subs(allComps[i][2]) - eval(allComps[i][1]).subs(allComps[i][2])):
711 #        simplify() == 0:
712 #            print allComps[i][0] + ' and ' + allComps[i][1] + ': Incorrect\n'
713
714 #####
715 ### Define a a function that calculates p bar answer. ###
716 #####
717
718 def calcP(A, T, R):
719     assert (A <= 2*pi and A >= 0), "a is out of bounds. Should be in 0<a<2*pi"
720     assert (T <= 2*pi and T >= 0), "s is out of bounds. Should be in 0<s<2*pi"
721
722     if A > pi:
723         if A < 4*pi - 2*T:
724             p = pNW7.subs({a:A, t:T, r:R}).n()
725         elif A <= 3*pi - T:
726             p = pNE2.subs({a:A, t:T, r:R}).n()
727         else:
728             p = pNE1.subs({a:A, t:T, r:R}).n()
729     else:
730         if A < 4*pi - 2*T:
731             p = pSE3.subs({a:A, t:T, r:R}).n()
732         else:
733             p = pSE2.subs({a:A, t:T, r:R}).n()
734     return p
735
736 #####
737 ## Apply to entire grid ##
738 #####
739
740 # How many values for each parameter
741 nParas = 100
742
743 # Make a vector for a and s. Make an empty nParas x nParas array.
744 # Calculated profile sizes will go in pArray
745 tVec = np.linspace(0, 2*pi, nParas)
746 aVec = np.linspace(0, 2*pi, nParas)
747 pArray = np.zeros((nParas,nParas))
748
749 # Calculate profile size for each combination of parameters
750 for i in range(nParas):
751     for j in range(nParas):
752         pArray[i][j] = calcP(aVec[i], tVec[j], 1)
753
754 # Turn the array upside down so origin is at bottom left.
755 pImage = np.flipud(pArray)
756
757 # Plot and save.
758 pl.imshow(pImage, interpolation='none', cmap=pl.get_cmap('Blues'))
759 #pl.show()
760
761 pl.savefig('/home/tim/Dropbox/phd/Analysis/REM-chapter/imgs/profilesCalculated.png')
762
763
764
765
766 #####
767 ### Output R function. ###
768 #####

```

```

769
770 # To reduce mistakes, output R function directly from python.
771 # However, the if statements, which correspond to the bounds of each model, are not automatic.
772
773 Rfunc = open('/home/tim/Dropbox/phd/Analysis/REM-chapter/supplementaryRscript.R', 'w')
774
775 Rfunc.write("""
776 # Functions to calculate density.
777 #
778 # Tim C.D. Lucas, Elizabeth Moorcroft, Robin Freeman, Marcus J. Rowcliffe, Kate E. Jones.
779 #
780 # calcDensity is the main function to calculate density.
781 # It takes parameters z, alpha, theta, r, animalSpeed, t
782 # z - The number of camera/acoustic counts or captures.
783 # alpha - Call width in radians.
784 # theta - Sensor width in radians.
785 # r - Sensor range in metres.
786 # animalSpeed - Average animal speed in metres per second.
787 # t - Length of survey in sensor seconds i.e. number of sensors x survey duration.
788 #
789 # calcAbundance calculates abundance rather than density and requires an extra parameter
790 # area - In metres squared. The size of the region being examined.
791
792
793 # Internal function to calculate profile width as described in the text
794 calcProfileWidth <- function(alpha, theta, r){
795   if(alpha > 2*pi | alpha < 0)
796     stop('alpha is out of bounds. alpha should be in interval 0<a<2*pi')
797   if(theta > 2*pi | theta < 0)
798     stop('theta is out of bounds. theta should be in interval 0<a<2*pi')
799
800   if(alpha > pi){
801     if(alpha < 4*pi - 2*theta){
802       "" +
803       '      p <- ' + str(pNW7) +
804       '\n      } else if(alpha <= 3*pi - theta){'
805       '\n      p <- ' + str(pNE2) +
806       '\n      } else {'
807       '\n      p <- ' + str(pNE1) +
808       '\n      }'
809       '\n      } else {'
810       '\n      if(alpha < 4*pi - 2*theta){'
811       '\n      p <- ' + str(pSE3) +
812       '\n      } else {'
813       '\n      p <- ' + str(pSE2) +
814       '\n      }'
815       '\n      }'
816       '\n      return(p)'
817       '\n}' +
818       ""
819 # Calculate a population density. See above for units etc.
820 calcDensity <- function(z, alpha, theta, r, animalSpeed, t){
821   # Check the parameters are suitable.
822   if(z <= 0 | !is.numeric(z)) stop('Counts, z, must be a positive number.')
823   if(animalSpeed <= 0 | !is.numeric(animalSpeed)) stop('animalSpeed must be a positive number.')
824   if(t <= 0 | !is.numeric(t)) stop('Time, t, must be a positive number.')
825
826   # Calculate profile width, then density.
827   p <- calcProfileWidth(alpha, theta, r)
828   D <- z/{animalSpeed*t*p}
829   return(D)
830 }
831
832 # Calculate abundance rather than density.
833 calcAbundance <- function(z, alpha, theta, r, animalSpeed, t, area){
834   if(area <= 0 | !is.numeric(area)) stop('Area must be a positive number')
835   D <- calcDensity(z, alpha, theta, r, animalSpeed, t)
836   A <- D*area
837   return(A)
838 }
839 """)
840 )
841
842 Rfunc.close()

```

## 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  $\theta$  and  $\alpha$  it automatically selects the correct model to apply.

```

1  # Functions to calculate density.
2  #
3  # Tim C.D. Lucas, Elizabeth Moorcroft, Robin Freeman, Marcus J. Rowcliffe, Kate E. Jones.
4  #
5  #
6  # calcDensity is the main function to calculate density.
7  # It takes parameters z, alpha, theta, r, animalSpeed, t
8  # z - The number of camera/acoustic counts or captures.
9  # alpha - Call width in radians.
10 # theta - Sensor width in radians.
11 # r - Sensor range in metres.
12 # animalSpeed - Average animal speed in metres per second.
13 # t - Length of survey in sensor seconds i.e. number of sensors x survey duration.
14 #
15 # calcAbundance calculates abundance rather than density and requires an extra parameter
16 # area - In metres squared. The size of the region being examined.
17
18
19 # Internal function to calculate profile width as described in the text
20 calcProfileWidth <- function(alpha, theta, r){
21   if(alpha > 2*pi | alpha < 0)
22     stop('alpha is out of bounds. alpha should be in interval 0<a<2*pi')
23   if(theta > 2*pi | theta < 0)
24     stop('theta is out of bounds. theta should be in interval 0<a<2*pi')
25
26   if(alpha > pi){
27     if(alpha < 4*pi - 2*theta){
28       p <- r*(theta - cos(alpha/2) + 1)/pi
29     } else if(alpha <= 3*pi - theta){
30       p <- r*(theta - cos(alpha/2) + cos(alpha/2 + theta))/pi
31     } else {
32       p <- r*(theta + 2*sin(theta/2))/pi
33     }
34   } else {
35     if(alpha < 4*pi - 2*theta){
36       p <- r*(theta*sin(alpha/2) - cos(alpha/2) + 1)/pi
37     } else {
38       p <- r*(theta*sin(alpha/2) - cos(alpha/2) + cos(alpha/2 + theta))/pi
39     }
40   }
41   return(p)
42 }
43
44 # Calculate a population density. See above for units etc.
45 calcDensity <- function(z, alpha, theta, r, animalSpeed, t){
46   # Check the parameters are suitable.
47   if(z <= 0 | !is.numeric(z)) stop('Counts, z, must be a positive number.')
48   if(animalSpeed <= 0 | !is.numeric(animalSpeed)) stop('animalSpeed must be a positive number.')
49   if(t <= 0 | !is.numeric(t)) stop('Time, t, must be a positive number.')
50
51   # Calculate profile width, then density.
52   p <- calcProfileWidth(alpha, theta, r)
53   if(p <= 0) stop('Calculated profile width is 0. We would therefore expect 0 captures. If z is
54     not zero, then the density is undefined.')
55   D <- z/(animalSpeed*t*p)
56   return(D)
57 }
58
59 # Calculate abundance rather than density.
60 calcAbundance <- function(z, alpha, theta, r, animalSpeed, t, area){
61   if(area <= 0 | !is.numeric(area)) stop('Area must be a positive number')
62   D <- calcDensity(z, alpha, theta, r, animalSpeed, t)
63   A <- D*area
64   return(A)
65 }

```

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