ELECTRONIC APPENDIX

Phylogeny can make the mid-domain effect an inappropriate null model

by

T. Jonathan Davies, Richard Grenyer
and John L. Gittleman

Biol. Lett. (doi:10.1098/rsbl.2005.0297)

ELECTRONIC APPENDIX A

METHODS

(a) *Model description*

Each replicate was initiated with a single occupied cell in the centre of the matrix, representing the ancestral 'seed' taxon, and a constant per-cell probability of range expansion and contraction. The following series of stepwise functions were then iterated, simulating speciation to a 'fauna' of 500 species:

1) Range expansion: For each species any occupied cell could expand into an adjacent cell with the probability RE. If the newly occupied cell had previously been unoccupied by that species the total number of occupied cells was increased by one. For each taxon cells were limited to two states (occupied or unoccupied) though a single cell could be occupied by many taxa.
2) Range contraction: The probability of any occupied cell becoming unoccupied (RC) was evaluated for all occupied cells for each species per iteration, and the number of occupied cells adjusted accordingly.
3) Speciation: An initial speciation probability (Z) of 0.001 was specified for the seed taxon. At each iteration all taxa with geographical extents greater than a single cell had a chance of speciating. If speciation occurred the geographical range of the taxon was split into two segments representing two new (daughter) species with a speciation probability inherited from their ancestral taxon, as described in the main text.

By seeding each replicate in the centre of the grid we may be more likely to recover a mid-domain effect. Our interest was in evaluating when there was a significant departure from the mid-domain effect hence our analyses are conservative. Computational time constraints prohibited extending the simulation runs to greater than 500 taxa. While we believe our results are robust to the number of evolved species, we cannot be certain that some emergent properties, that are not observed in the current runs, become evident for very large numbers of species.

(b) ***Model parameterisation***

RE and RC were varied in turn between 0.1 and 1 in 0.1 increments whilst holding σ at zero (speciation probability was invariant among lineages) for 100 replicates. We selected three paired-values of RE and RC that spanned the parameter space for which all runs went to completion and all cells were occupied by at least one taxon. Each simulation run of 1000 replicates was repeated for the alternative values of RE and RC.

(c) ***Assessing the MDE***

Implicit within the assumptions of MDE models is that species richness decreases symmetrically from a peak, and that the peak should be at the midpoint of all dimensions about which the effect is examined. For a two dimensional space, this can be described analytically with the expected species richness value at any point equal to $4pqstS$, where p and q are the relative distances from the longitudinal extremes of domain, s and t are the relative distances from the latitudinal extremes of domain and S is the total number of species (Bokma *et al.* 2001, extended from the one-dimensional formulae of Willig & Lyons 1998). However, comparing spatial patterns in species richness is confounded by the non-independence of values, a point of high species richness will likely lie adjacent to points of similar species richness, making statistical comparisons difficult (Jetz & Rahbek 2001). It is possible to correct for spatial non-independence; however, current methods are computationally intractable for large two-dimensional surfaces (e.g. see Rangel & Diniz-Filho 2004). We therefore adopted two alternative measures to indicate departure from the MDE.

First, we calculated symmetry of species richness about the midpoint across a latitudinal transect:

1) For each latitude (row in the final 100 x 100 richness map), the maximum species richness was found.
2) Latitudes were then ranked, with ties being awarded the mean rank.
3) The difference in rank was then calculated for each pair of latitudes equidistant about the midpoint latitude (i.e. between rows 50 and 51, 49 and 52 etc.) and these differences summed for all latitudes.

By ranking, we attempt to reduce the influence of one or a few very high richness scores on the overall symmetry for a given map. For this statistic, a perfectly symmetrical distribution about the midpoint would received a summed rank-difference of zero, the most symmetrical distribution possible without tied ranks would received a score of n/2 for n rows, and more asymmetrical distributions would receive higher scores. We performed a Kolmogorov-Smirnov (K-S) test (Sohkal & Rohlf 1995) on pair-wise comparison of the distribution of symmetry values, to evaluate whether the inherited models depart from the null, in which range location is independent from phylogenetic relatedness (the randomised models). In addition, the significance of the difference in median asymmetry between cases was examined by a simple randomisation procedure. Values were drawn at random from the 1000 randomised and 1000 inherited richness maps and allocated into two populations of the same sizes as the original. The observed difference in median was compared to the distribution of differences generated from 10000 replicates of random sampling (following Manly 1991).

Since the symmetry statistic would give equal scores to a flat latitudinal gradient in richness, or one symmetrically arranged with the maxima at the extremes, we produced a frequency surface for the location of the cell(s) of maximum species richness for each case (i.e. 1000 replicates of a given model). For presentation, the frequency surface was passed through a 2-d normal kernel smoother algorithm (the image.smooth function in the contributed ‘fields’ package in R) using a constant but arbitrary bandwidth. The underlying values are presented as text files in the Supplementary information.

In the text, we present only the asymmetry and frequency surfaces from models with per-cell-per-generation probabilities of expansion and contraction of 0.7 and 0.2 respectively. Here we present the same results for two other parameter pairs (1/0.4 and 0.2/0.1) together with the pairwise testing for differences in median asymmetry and the results of the KS tests.

(d) ***Computational Methods***

The simulation code was written in C++. Testing and prototyping was performed using the Bloodshed GNU C++ Compiler (http://www.bloodshed.net) for Windows, and the final version run on a cluster of LINUX workstations, using Intel compilers, at the

University of Virginia Research Computing Support Center. Post-simulation analysis and visualization was carried out in the statistical language R (http://www.r-project.org).

RESULTS

Supporting tests for the differences in symmetry are presented as table 1 (Kolmogorov-Smirnoff tests) and table 2 (randomisation tests of significance of the observed differences in median asymmetry score) for all three pairs of contraction/expansion probabilities. The distributions of asymmetry scores for the two pairs of probabilities not presented in the main text are given as figures A1 (RE=0.2, RC=0.1) & A2 (RE=1.0, RC=0.4). Accompanying maxima-frequency surfaces are given as figures A3 (RE=0.2, RC=0.1) & A4 (RE=1.0, RC=0.4). In all cases, the subheadings a-d refer to the same models as in the main text: a) σ=0 (equal-rates phylogeny), b) σ=0 with randomized range locations, c) σ=0.3 (more imbalanced phylogenies), d) σ=0.3 with randomized range locations.

SUPPLEMENTARY REFERENCES

Belair-Franche, J. & Contreras, D 2002 A Pearson test for symmetry with an application to the Spanish business cycle. *Spanish Econ. Rev.* **4**, 221-228.

Manly, B. F. J. 1991 *Randomization, bootstrap and monte carlo methods in biology*. London: Chapman & Hall.

Rangel, T. F. L. V. B. & Diniz-Filho, J. A. F. 2002 Worldwide patterns in species richness of Falconiformes: analytical null models, geometric constraints and the mid-domain effect. *Braz. J. Biol.* **64**, 299-308.

Sohkal, R. R. & Rohlf, F. J. 1995 *Biometry*. New York: Freeman & Company.

Willig M. R. & Lyons, S. K. 1998 An analytical model of latitudinal gradients of species richness with an empirical test for marsupials and bats in the New World. *Oikos* **80**, 93-98.

RE 0.2, RC 0.1

	(a)	(b)	(c)	(d)
(a)	x	0.593 ***	0.671 ***	0.18 ***
(b)	x	x	0.877 ***	0.429 ***
(c)	x	x	x	0.703 ***
(d)	x	x	x	x

RE 0.7, RC 0.2

	(a)	(b)	(c)	(d)
(a)	x	0.29 ***	0.556 ***	0.676 ***
(b)	x	x	0.634 ***	0.549 ***
(c)	x	x	x	0.705 ***
(d)	x	x	x	x

RE 1.0, RC 0.4

	(a)	(b)	(c)	(d)
(a)	x	0.393 ***	0.597 ***	0.492 ***
(b)	x	x	0.685 ***	0.297 ***
(c)	x	x	x	0.665 ***
(d)	x	x	x	x

*** p<=0.001

Table 1 Kolmogorov-Smirnoff test D-statistic, where the null hypothesis is that the symmetry scores for both cases are drawn from the same distribution. Cases a – d are as described in the Appendix (Results).

RE 0.2, RC 0.1

	(a)	(b)	(c)	(d)
(a)	X	245 ***	960 ***	63.5 ***
(b)	X	x	1205 ***	181.5 ***
(c)	X	x	x	1023.5 ***
(d)	X	x	x	x

RE 0.7, RC 0.2

	(a)	(b)	(c)	(d)
(a)	X	127 ***	563 ***	464 ***
(b)	X	x	690 ***	337 ***
(c)	X	x	x	1027 ***
(d)	X	x	x	x

RE 1.0, RC 0.4

	(a)	(b)	(c)	(d)
(a)	X	137 ***	419 ***	227 ***
(b)	X	x	556 ***	90 ***
(c)	X	x	x	646 ***
(d)	X	x	x	x

*** P<=0.0001

Table 2: Significance of the difference in median asymmetry as assessed by a jackknife randomization test with 10,000 replicates. Cases a – d are as described in the Appendix (Results).

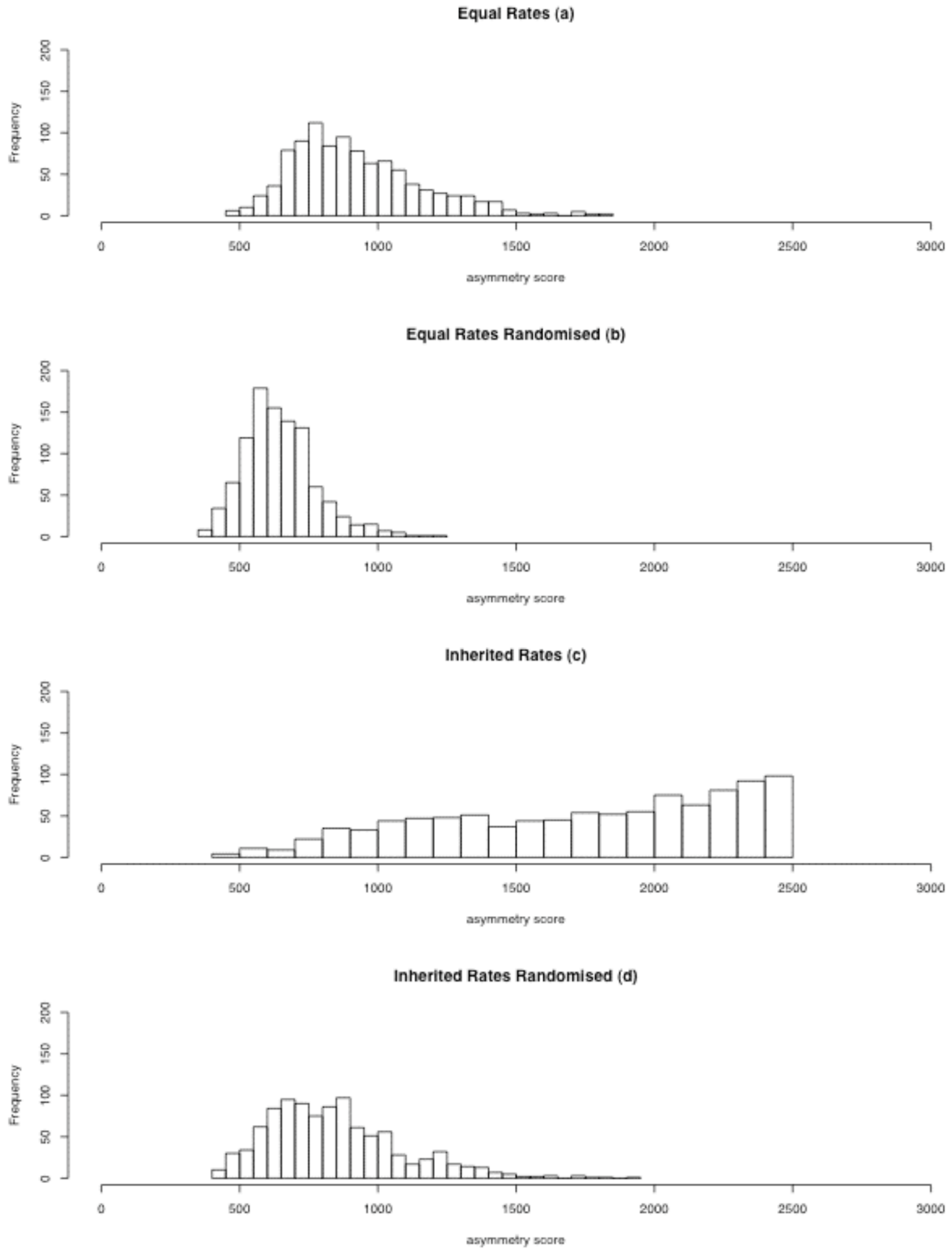
Equal Rates (a)
Equal Rates Randomised (b)
Inherited Rates (c)
Inherited Rates Randomised (d)
Frequency
0
50
100
150
200
asymmetry score
0
500
1000
1500
2000
2500
3000

FIGURE A1

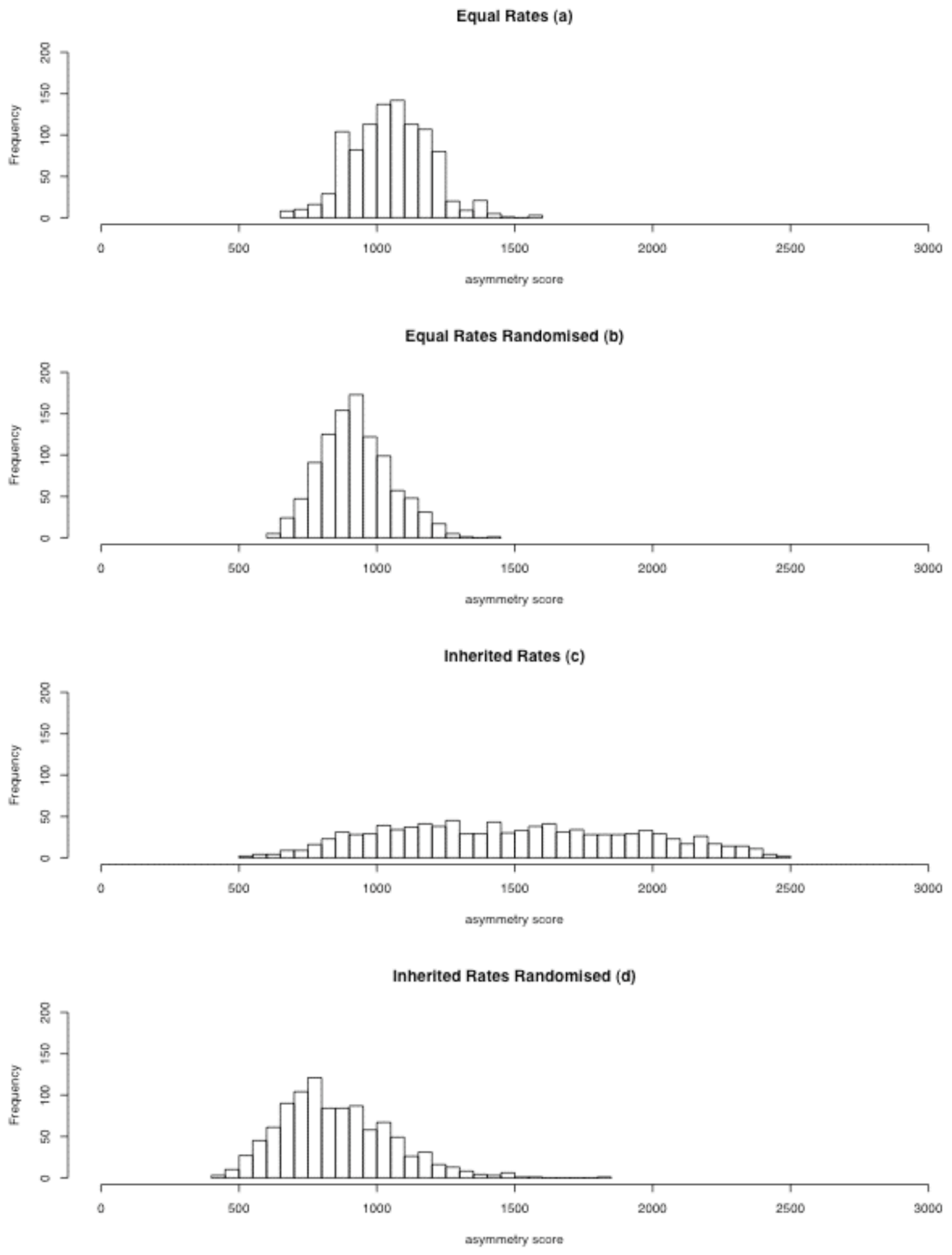
Equal Rates (a)
Equal Rates Randomised (b)
Inherited Rates (c)
Inherited Rates Randomised (d)
Frequency
asymmetry score
0
50
100
150
200
500
1000
1500
2000
2500
3000

FIGURE A2

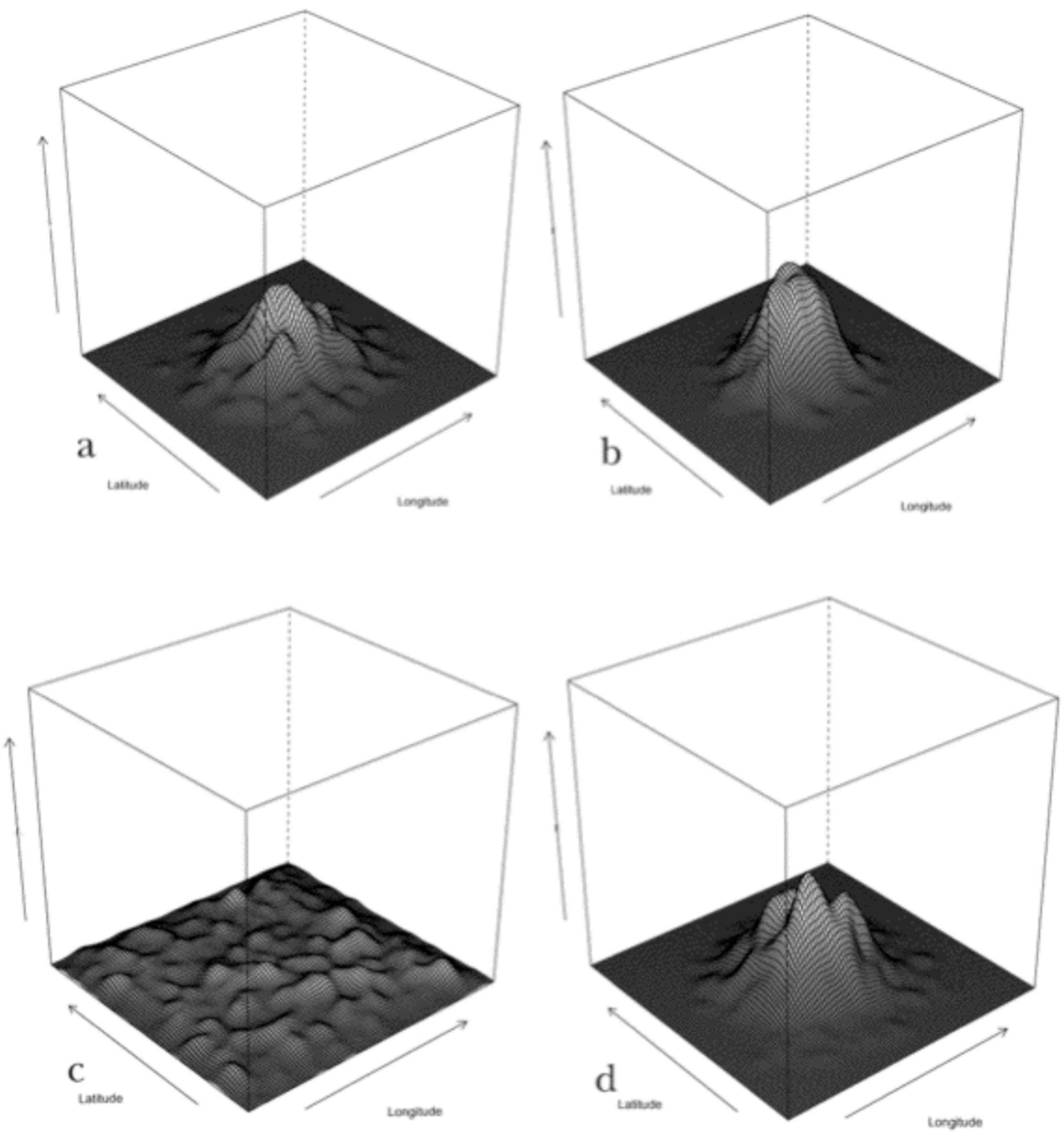
a
Latitude
Longitude
b
Latitude
Longitude
c
Latitude
Longitude
d
Latitude
Longitude

FIGURE A3

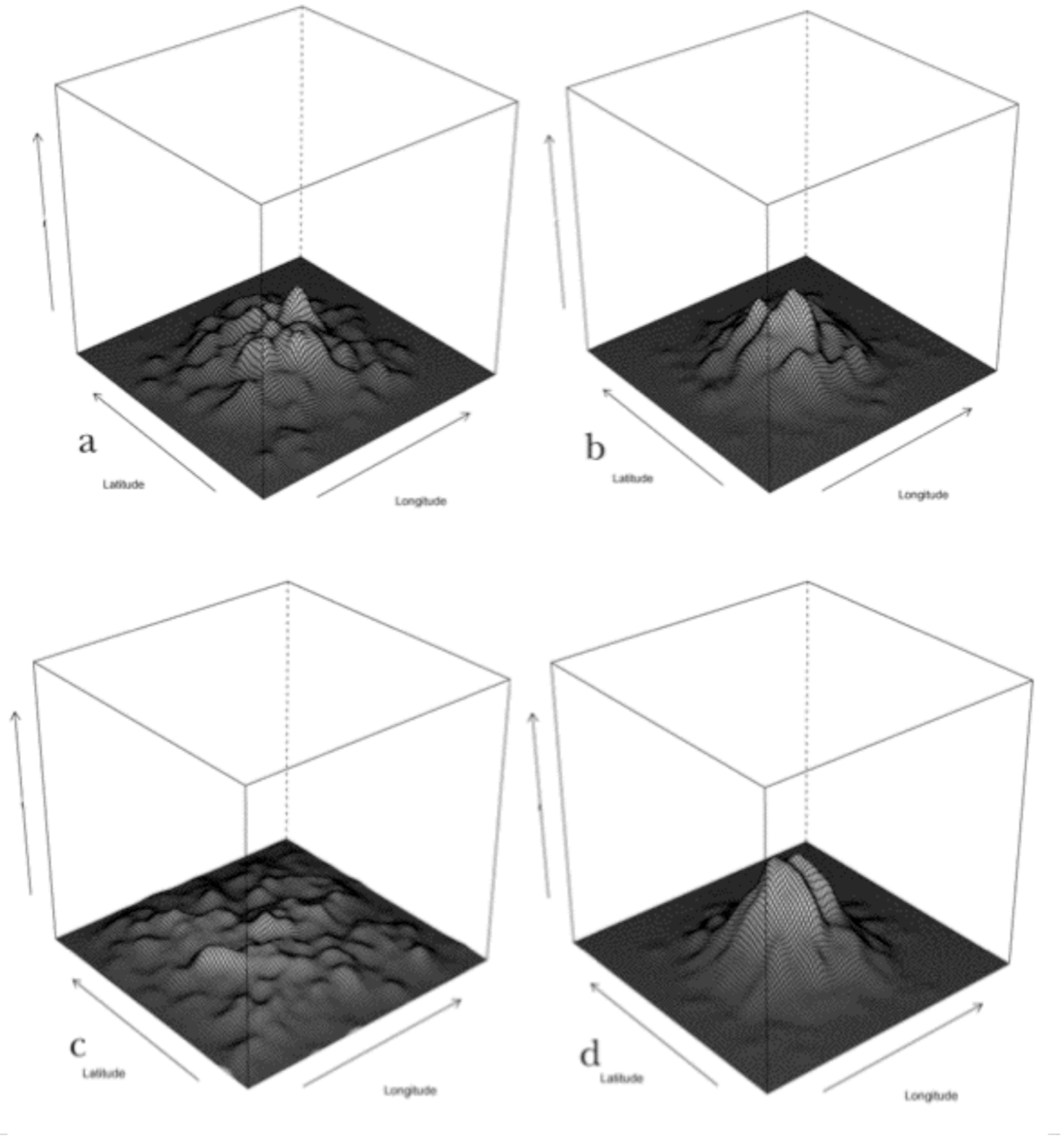
a
Latitude
Longitude
b
Latitude
Longitude
c
Latitude
Longitude
d
Latitude
Longitude

FIGURE A4

ELECTRONIC APPENDIX B

```
//Phylogeny can make the mid-domain effect an inappropriate null model
//Authors: T. Jonathan Davies*, Richard Grenyer and John L. Gittleman
//Department of Biology, Gilmer Hall, University of Virginia,
Charlottesville, VA 22904 USA
//*Author for correspondence (jdavies@virginia.edu)
//code written  code was written in C++.
//Testing and prototyping was performed using the Bloodshed GNU C++
Compiler (http://www.bloodshed.net)
//This version compiled and run on Windows 2000 Professional

// v0.1.a

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <fstream.h>                    // writing a text file
#include <string.h>
using namespace std;

//declare arrays
int continent[10000][100][100],          //continent[a][b][c] - a =
number matices to create (MUST BE LARGER THAN speciesSNo), b & c =
dimensions of geographical matrix
newspecies[100][100],                    //temporary matrix filled in
each loop to number new species, size must be equivalent to
geographical matrix size above
SpeciesRichness[100][100],               //final matrix for adding up
species richness, size must be equivalent to geographical matrix size
above
cellnumber[10000],                       //keeps track of the number of
occupied cells in speciating lineages
africa[100][100];                        //continent file (land = 0,
sea = 9)
float speciesinfo[10000][7];             //speciesinfo[a][b] - a = each
species matrix, b = unique species identifiers: (p(ext), p(migrate),
ancestor, generation, ID No, No occupied cells, p(split)).
double r;                                // random number


int main()//initialise main loop
{

//declare variables
int w, x, y,  a, b, xx, yy, OccupiedCells, counter, SpNo,
  reps, stoploop, speciesNo, matrix, newcell, cellcount,
   splitting, randcells, direction, OldSpNo, replicates,
   stdevrun,AddCells, NeverMult,
   extant, taxcount, EternalLoop
;
float migrate, death, pdeath, pmigrate, psplit, split, stdev, runs,
stdevinc;
char comm;
```

```
replicates = 1000;//NUMBER OF REPLICATES TO RUN


srand( (unsigned)time( NULL ) ); // initiate Rand() function;


//SET STARTING PARAMETERS *****************************
speciesNo = 500; //nb many species may go extint
matrix = 100;
migrate = 0.7;
death = 0.2;
stdev = 0.3;
//stdevinc = 0.1;//increase in standard deviation each loop
split = 0.001; //probability of having parapatric speciation
//****************************************************

//for (stdevrun = 1; stdevrun <= 1; stdevrun++)
//{

EternalLoop = 1;// this is a counter to enable the programme to quit if
populations keep going extinct and hence never obtain specified number
of extant species
runs = 1;

while (runs <= replicates)//number of replicates to run
{


//Initialise parameters in the loops
OccupiedCells = 1;
counter = 0;
SpNo = 1;
OldSpNo = SpNo;
taxcount = 1;

//sets the elements in continent to zero
for(w=0; w<speciesNo; w++)
{
for(x=0; x<matrix; x++)
{
for (y = 0; y<matrix; y++)
{
continent[w][x][y]=0;
}//next y
}//next x
}//next w


//a = rand() % 9;//for randomising starting cell (x co-ord)
//b = rand() % 9;//for randomising starting cell (y co-ord)
a = 50;// x co-ord of starting cell
b = 50;// y co-ord of starting cell

continent[0][a][b]=1;//centre of origin

speciesinfo[0][0] = death;
```

```
speciesinfo[0][1] = migrate;
speciesinfo[0][2] = 0;//ancestor
speciesinfo[0][3] = 0;//generation
speciesinfo[0][4] = 0;//unique sp ID
speciesinfo[0][5] = 1;//No occupied cells in matrix (one initial cell)
speciesinfo[0][6] = split;//initial probability of speciating
(splitting matrix in to two)


while (taxcount<speciesNo)//loop until get required number of species
{
//insert main programme


for( reps = 0; reps < OldSpNo; reps++)//Loop through all the species in
each generation
{

if (taxcount == speciesNo)//reached required number if species - only
run loop if this is not true
{
break;
}


//EXTINCTION
//*********************************************************************
*
if(counter > 0)//only kicks in after the second generation so the
initial cell does not immediately go extinct!
{

if (speciesinfo[reps][5] > 0)//then this specis is not extinct
{

for (x = 0; x< matrix; x++)
{
for (y = 0; y< matrix; y++)
{
if (continent[reps][x][y] == 1)
{
pdeath = rand();
pdeath = pdeath/RAND_MAX;//32767;
if (speciesinfo[reps][0] > pdeath)
{
continent[reps][x][y] = 0;//that cell becomes unoccupied
OccupiedCells = OccupiedCells - 1;//one less total numbe of occupied
cells
speciesinfo[reps][5] = speciesinfo[reps][5] - 1;//one less occupied
cell in matrix

if (speciesinfo[reps][5] == 0)//if the species goes extinct - adjust
taxcount accordingly
{
taxcount = taxcount -1;//minus one to total number of extant species
}

}
```

```
}
}
}
}
}
//*********************************************************************
*
//END OF EXTINCTION


//START OF MIGRATION
//*********************************************************************
*
for (x = 0; x< matrix; x++)
{
for (y = 0; y< matrix; y++)
{
if (continent[reps][x][y] == 1)
{
pmigrate = rand();
pmigrate = pmigrate/RAND_MAX;//32767;
if (speciesinfo[reps][1] > pmigrate)
{
newcell = rand() % 8;
    if (newcell == 0)
    {
    yy = y-1;
    xx = x;
    }
    if (newcell == 1)
    {
    yy = y-1;
    xx = x-1;
    }
    if (newcell == 2)
    {
    yy = y-1;
    xx = x+1;
    }
    if (newcell == 3)
    {
    yy = y;
    xx = x-1;
    }
    if (newcell == 4)
    {
    yy = y;
    xx = x+1;
    }
    if (newcell == 5)
    {
    yy = y+1;
    xx = x;
    }
    if (newcell == 6)
    {
    yy = y+1;
    xx = x-1;
    }
    if (newcell == 7)
    {
    yy = y+1;
    xx = x+1;
    }

if (xx > (matrix-1) || xx<0 || yy > (matrix-1) || yy<0){}//do not move
else//new cells not outside range of matrix
{
if (continent[reps][xx][yy] == 0) //cell is unoccupid
{
continent[reps][xx][yy] = 2;
OccupiedCells = OccupiedCells + 1;
speciesinfo[reps][5]= speciesinfo[reps][5] + 1;
}//end if
}//end if

}//end if
}//end if
}//next y
}//next x
//END OF MOVEMENT


//set all occupied cells to 1
for (x = 0; x< matrix; x++)
{
for (y = 0; y< matrix; y++)
{
if (continent[reps][x][y] > 1)
{
continent[reps][x][y] = 1;
}//end if
}//next y
}//next x
//*********************************************************************
*
//END OF MIGRATION


// REMOVED PERIPATRIC SPECIATION EVENTS


//PARAPATRIC SPECIATION
//*********************************************************************
*
if (speciesinfo[reps][5] > 1)//then there are more than one occupied
cells in the matrix
{
randcells = int(speciesinfo[reps][5]);//number of occupied cells as an
integer
randcells = randcells -1;//number of occupied cells as integer -1
cellcount = rand() % randcells;//random number between 0 and (number of
occupied cells -1)
cellcount = cellcount + 1;//number of cells to split off (between one
and number of occupied cells - 1)
psplit = rand();//random number between 0 and rand_max (3276)
psplit = psplit/RAND_MAX;//random number between 0 and 1
splitting = 0;//initialise count for number of cells being split


if (speciesinfo[reps][6] > psplit)//then speciate
{
taxcount = taxcount + 1;//add one to total number of extant species

direction = rand() % 2;//split range vertically or horizontally -
random number (0 or 1)
AddCells = rand() % 2;//direction add new cells to newly created
species - random number (0 or 1)

if (direction == 0)//split horizontally
{
if (AddCells == 0)//addcells from left to right
{
for(x = 0; x < matrix; x++)
{
for (y = 0; y < matrix; y++)
{
if (splitting < cellcount)//create new species range until required
number of cells
{
continent[SpNo][x][y] = continent[reps][x][y];
splitting = splitting + continent[reps][x][y];//add up new species
cells
continent[reps][x][y] = 0;
}
else// after required number of cells obtained - fill the rest with 0
{
continent[SpNo][x][y] = 0;
}//end if
}//next y
}//next x
}//end if


else//add cells from right to left
{
for(x = 0; x < matrix; x++)
{
for (y = matrix-1; y >= 0; y--)
{
if (splitting < cellcount)//create new species range until required
number of cells
{
continent[SpNo][x][y] = continent[reps][x][y];
splitting = splitting + continent[reps][x][y];//add up new species
cells
continent[reps][x][y] = 0;
}
else// after required number of cells obtained - fill the rest with 0
{
continent[SpNo][x][y] = 0;
}//end if
}//next y
}//next x
}//end if


}//end if


if (direction == 1)//split vertically (see above)
{

if (AddCells == 0)//add cells from top to bottom
{
for(x = 0; x < matrix; x++)
{
for (y = 0; y < matrix; y++)
{
if (splitting < cellcount)
{
continent[SpNo][y][x] = continent[reps][y][x];
splitting = splitting + continent[reps][y][x];
continent[reps][y][x] = 0;
}
else
{
continent[SpNo][y][x] = 0;
}//end if
}//next y
}//next x
}//end if

else//AddCells from bottom to top
{
for(x = 0; x < matrix; x++)
{
for (y = matrix-1; y >= 0; y--)
{
if (splitting < cellcount)
{
continent[SpNo][y][x] = continent[reps][y][x];
splitting = splitting + continent[reps][y][x];
continent[reps][y][x] = 0;
}
else
{
continent[SpNo][y][x] = 0;
}//end if
}//next y
}//next x
}//end if

}//end if


//debugger
if (splitting>cellcount)
{
cout<<"splitting = "<<splitting<<endl;
cout<<"cellcount = "<<cellcount<<endl;
cin.get();
```

```
}


//give new values to the separate species
speciesinfo[SpNo][0] = speciesinfo[reps][0];// + (r/100000);//plus a
number drawn from a normal distribution
speciesinfo[SpNo][1] = speciesinfo[reps][1];// + (r/100000);//plus a
number drawn from a normal distribution
speciesinfo[SpNo][2] = speciesinfo[reps][4];//ancestors unique ID
speciesinfo[SpNo][3] = counter;//generation created
speciesinfo[SpNo][4] = SpNo;//unique ID number
speciesinfo[SpNo][5] = cellcount;

//create gaussian random number "r" with mean of 0 and standard
deviation "stdev"
static double V1, V2, S;
      static int phase = 0;
      double gauss;
      if(phase == 0) {
            do {
                  double U1 = (double)rand() / RAND_MAX;
                  double U2 = (double)rand() / RAND_MAX;
                  V1 = 2 * U1 - 1;
                  V2 = 2 * U2 - 1;
                  S = V1 * V1 + V2 * V2;
                  } while(S >= 1 || S == 0);
            gauss = V1 * sqrt(-2 * log(S) / S);
      } else
            gauss = V2 * sqrt(-2 * log(S) / S);
      phase = 1 - phase;
      r = gauss * stdev;

speciesinfo[SpNo][6] = exp(log(speciesinfo[reps][6]) + r);//probability
of speciating

//give new values to the original species
speciesinfo[reps][5] = speciesinfo[reps][5] - cellcount;

      if(phase == 0) {
            do {
                  double U1 = (double)rand() / RAND_MAX;
                  double U2 = (double)rand() / RAND_MAX;
                  V1 = 2 * U1 - 1;
                  V2 = 2 * U2 - 1;
                  S = V1 * V1 + V2 * V2;
                  } while(S >= 1 || S == 0);
            gauss = V1 * sqrt(-2 * log(S) / S);
      } else
            gauss = V2 * sqrt(-2 * log(S) / S);
      phase = 1 - phase;
      r = gauss * stdev;

speciesinfo[reps][6] = exp(log(speciesinfo[reps][6]) + r);//probability
of speciating


SpNo = SpNo + 1;//add a new species number = counter for creation of
new matrices
```

```
}//end if
}//end if
//*********************************************************************
*
//END OF PARAPATRIC SPECIATION


}// next reps

OldSpNo = SpNo;//sets number of matrices to loop through each
generation
counter = counter + 1;//counting generations


if (OccupiedCells == 0)//Gone extint
{
if (EternalLoop > 5 * replicates)//populations keep going extinct -
hence "runs" never going to completion
{
break;
}
else
{
runs = runs -1;
break;
}
}


extant = 0;
NeverMult = 0;
for (w = 0; w<SpNo; w++)
{
if (speciesinfo[w][5] > 0)//that matrix not extinct
{
extant = extant + 1;
if (speciesinfo[w][6] <=0.000001)//that matrix never multiplies
(practically)!!
{
NeverMult = NeverMult + 1;//all species have a negative prob of
speciating
}//end if
}//end if
}// next w


if (extant == NeverMult)// then all species never speciate
{
runs = runs -1;
break;
}


}//END OF DO-WHILE = OBTAINED NUMBER OF SPECIES


//OUTPUT DETAILS
```

```
//*********************************************************************
*
if (OccupiedCells > 0 && taxcount == speciesNo)//only OUTPUT if all
pops not gone extinct
{

//add up total species richness
for(x = 0; x < matrix; x++)
{
for (y = 0; y < matrix; y++)
{
SpeciesRichness[x][y] = continent[0][x][y];
}
}

for (w = 1; w < SpNo  ; w++)
{
for(x = 0; x < matrix; x++)
{
for (y = 0; y < matrix; y++)
{
if (continent[w][x][y] >0)//subsequent generation add aditional
occupied cells
{
SpeciesRichness[x][y] = SpeciesRichness[x][y] + 1;
}
}
}
}


//OUTPUT TO MAP FILE
//*********************************************************************
*
char sdrun[10];
char run[10];
char mrun[10];
char extrun[10];
sprintf(run,"%.2f",runs);//converts integer (runs) to string (run)
sprintf(mrun,"%.2f",migrate);
sprintf(extrun,"%.2f",death);
sprintf(sdrun,"%.2f",stdev);
char stemp[46];
strcpy (stemp,"map");
strcat (stemp,"StDev");
strcat (stemp,sdrun);
strcat (stemp,"Mig");
strcat (stemp,mrun);
strcat (stemp,"Ext");
strcat (stemp,extrun);
strcat (stemp,"Reps");
strcat (stemp,run);
strcat (stemp,".txt");

ofstream examplefile (stemp);
comm = 'd';
if (examplefile.is_open())
 {
```

```
 for(x=0; x<matrix; x++)
 {
 for (y = 0;y<matrix;y++)
 {
 {
 examplefile << SpeciesRichness[x][y];
  }
 if (y == (matrix - 1))
 {
 examplefile << endl;
 }
 else
 {
 examplefile<<comm;
 }
 }
 }
 examplefile.close();
}
//*********************************************************************
*
//END OF OUTPUT


//OUTPUT PHYLOGENY FILE
//*********************************************************************
*
char sstemp[46];//this bit outputs file numbered sequentially
strcpy (sstemp,"phylogeny");
strcat (sstemp,"StDev");
strcat (sstemp,sdrun);
strcat (sstemp,"Mig");
strcat (sstemp,mrun);
strcat (sstemp,"Ext");
strcat (sstemp,extrun);
strcat (sstemp,"Reps");
strcat (sstemp,run);
strcat (sstemp,".txt");

ofstream examplefile2 (sstemp);
if (examplefile2.is_open())
 {
 for (x = 0;x<SpNo;x++)
 {
 examplefile2 <<
speciesinfo[x][3]<<comm<<speciesinfo[x][2]<<comm<<speciesinfo[x][4]<<co
mm<<speciesinfo[x][5]<<comm<<SpNo<<comm<<speciesinfo[x][6]<<comm;
 examplefile2 << endl;
 }
  examplefile2.close();
 }
//*********************************************************************
*
//END OF TREE FILE OUPUT


}//end if (only do when all cells not extinct
//END OF OUTPUT DETAILS
```

```
//*********************************************************************
*


EternalLoop = EternalLoop + 1;
runs = runs + 1;
}//next replicate to run (while loop)

//stdev = stdev + 0.1;
//}//next stdev run


 return 0;
}
```

ELECTRONIC APPENDIX C

#R code (http://www.r-project.org) for several functions to manipulate and visualize #the output from the C++ program. Output requires the lattice library.

#VBAmaker reads in output from C++ code into a 100x100x1000 array

```
VBAmaker<-function(){
    dim3size<-length(dir())
    destin<-NULL
    the.big.array<-NULL
    for (i in dir()){
      currstrings<-scan(i, what="numeric",sep="d")
        destin<-append(destin,currstrings)
        print("Next file!")
                    }
    print("String to numeric conversion...")
    destin<-as.numeric(destin)
    print("Piping vector to array...")
    the.big.array<-array(destin, dim=c(100,100,dim3size))
    print("Returning array to workspace!")
    return(the.big.array)
    }
```

#This function calculates a rank symmetry score for a vector of length 100

```
rank_symmetry_scorer<-function(richnesses){
      how_many<-length(richnesses)
      negs<-sort(seq((how_many/2):1),decreasing=T)
      poss<-seq(from=how_many/2+1,to=how_many,by=1)
      my_ranks<-rank(richnesses)
      rank_difs<-NULL
      for (i in seq(1:(how_many/2))){
        curr_diff<-abs(my_ranks[negs[i]]-my_ranks[poss[i]])
            rank_difs<-append(rank_difs, curr_diff)

    }
}
```

#This function calculates the rank symmetry score for the rows for each of 1000

#replicates (3rd dimension of the array) for a variety of summary statistics applied #to the

columns.

```
variants_sum_calc<-function(the_array){
    num_slices<-dim(the_array)[3]
    the_maxs<-NULL
    the_means<-NULL
    the_medians<-NULL
```

```
     the_sums<-NULL
     the_mins<-NULL
     for ( i in seq(1:num_slices)){
          curr_max<-apply(the_array[,,i],1,max)
          curr_mean<-apply(the_array[,,i],1,mean)
          curr_median<-apply(the_array[,,i],1,median)
          curr_sum<-apply(the_array[,,i],1,sum)
          curr_min<-apply(the_array[,,i],1,min)
          the_maxs<-append(the_maxs,sum(rank_symmetry_scorer(curr_max)))
          the_means<-
append(the_means,sum(rank_symmetry_scorer(curr_mean)))
          the_medians<-
append(the_medians,sum(rank_symmetry_scorer(curr_median)))
          the_sums<-append(the_sums,sum(rank_symmetry_scorer(curr_sum)))
          the_mins<-append(the_mins,sum(rank_symmetry_scorer(curr_min)))
          }
     the_output<-
as.data.frame(cbind(the_maxs,the_means,the_medians,the_sums,the_mins))
     names(the_output)<-c("Rank symmetry of row maxima","Rank symmetry
of row means","Rank symmetry of row medians","Rank symmetry of row
sums","Rank symmetry of row minima")
     return(the_output)
     }
```

#this function produces a 100 x 100 matrix with the frequency with which each cell #was the maximum point of species richness in one of the 1000 replicates

```
midpoint_mapper<-function(the_array){
   all.maxs<-function(x){return(which(x==max(x)))}
   the_maxima<-unlist(apply(the_array,3,all.maxs))
   the_nulls<-rep(0,10000)
   for (i in the_maxima){
      the_nulls[i]<-1+the_nulls[i]
                        }
   return(matrix(the_nulls,100,100))
}
```

#graphical code to produce the smoothed output frequency surfaces. The output #from midpoint_mapper() must be written to file first.

#This function requires four input files (cases a,b,c,d) and produces pdf output.

```
plot4maps<-function(file1,file2,file3,file4,pdffile){
   the.map1<-as.matrix(read.delim(file1, header=F))
   the.map2<-as.matrix(read.delim(file2, header=F))
   the.map3<-as.matrix(read.delim(file3, header=F))
   the.map4<-as.matrix(read.delim(file4, header=F))
   trellis.device(pdf, theme=col.whitebg(), h=8, w=8, file=pdffile)
   mp1<-wireframe(image.smooth(the.map1, theta=4), drape=T, zlim=c(0,2.5), zlab="", ylab="Lat.",
xlab="Long.", colorkey=F, par.box=list(lwd=0))
   mp2<-wireframe(image.smooth(the.map2, theta=4), drape=T, zlim=c(0,2.5), zlab="", ylab="Lat.",
xlab="Long.", colorkey=F, par.box=list(lwd=0))
```

```
   mp3<-wireframe(image.smooth(the.map3, theta=4), drape=T, zlim=c(0,2.5), zlab="", ylab="Lat.",
xlab="Long.", colorkey=F, par.box=list(lwd=0))
   mp4<-wireframe(image.smooth(the.map4, theta=4), drape=T, zlim=c(0,2.5), zlab="", ylab="Lat.",
xlab="Long.", colorkey=F, par.box=list(lwd=0))
   print(mp1)
   print(mp2)
   print(mp3)
   print(mp4)
   dev.off()
}
```