Methods in Ecology and Evolution

British Ecological Society

Methods in Ecology and Evolution 2015, **6,** 164–168 doi: 10.1111/2041-210X.12308

APPLICATION

NLMpy: a PYTHON software package for the creation of neutral landscape models within a general numerical framework

Thomas R. Etherington[1,2]*, E. Penelope Holland[3,4] and David O'Sullivan[5]

[1]*School of Applied Sciences, Auckland University of Technology, Private Bag 92006, Auckland 1010, New Zealand;* [2]*School of Environment, The University of Auckland, Private Bag 92019, Auckland 1142, New Zealand;* [3]*Landcare Research, P.O. Box 69040, Lincoln 7640, New Zealand;* [4]*Department of Biology, University of York, Heslington, York, YO10 5DD,UK; and* [5]*Department of Geography, University of California, Berkeley, 507 McCone Hall #4740, Berkeley, CA 94720, USA*

Summary

1. Neutral landscape models (NLMs) are widely used to model ecological patterns and processes across landscapes. However, the ability to generate NLMs is often made available through standalone bespoke software packages that have platform limitations.

2. We have developed a PYTHON package that brings together some of the more popular NLM algorithms using a general numerical framework.

3. The resulting NLMpy package: (i) allows for the creation of NLMs directly within a PYTHON modelling workflow or by other modelling software capable executing a PYTHON script, (ii) enables the first opportunity to create a NLM that combines different algorithms, (iii) provides easy integration with geographic information system data and (iv) creates a framework for developing other NLMs.

Key-words: artificial landscape, geographic information system, GIS, landscape generator, NLM, NumPy, SciPy, virtual landscape

Introduction

A neutral landscape model (NLM) is a simulated landscape that is used to examine and test observations of, and techniques designed to measure, the ecological patterns and processes across landscapes (With & King 1997; Turner, Gardner & O'Neill 2001, pp. 135–156). The successful application of NLMs has led to the development of software designed to create NLMs using a variety of algorithms. Examples include standalone software such as RULE (Gardner 1999), the subsequent QRULE (Gardner & Urban 2007) and SIMMAP (Saura & Martínez-Millán 2000), as well as software packages such as the ecomodtools package for R (Chipperfield, Dytham & Hovestadt 2011).

However, our own experience of using NLMs has required a more flexible software approach. Using a general numeric framework, we have developed the NLMpy PYTHON package that uses functions from the NumPy and SciPy packages (Oliphant 2007). NLMpy has several advantages over existing NLM software: (i) it is open source so it can be easily adapted or developed for specific modelling requirements, (ii) being cross-platform it can be used on any computer system, (iii) it brings together a much wider range of NLM algorithms, including some that are not available elsewhere, (iv) it can be easily combined with geographic information system (GIS) data, (v) it enables novel combinations and integrations of different NLM algorithms, and (vi) it can be embedded into larger modelling frameworks based on software that is capable of executing a PYTHON script. This study briefly describes the NLM algorithms integrated into NLMpy and demonstrates the methodological flexibility of the software.

*Correspondence author. E-mail: thomas.etherington@aut.ac.nz

NLM algorithms

We only intend to supply a summary description of the NLM algorithms. Potential users are advised to refer to the literature cited for a full description of the NLM algorithms and their previous application, and to the Supporting Information and source code for details on NLM computation. All the NLM algorithms presented produce a two-dimensional NumPy array with specified row and column dimensions, in which the elements of the array at each row and column position contain a value from 0 to 1. The difference between the NLM algorithms is the spatial autocorrelation of the element values.

In the 'random' NLM (Fig. 1a), there is no spatial autocorrelation as each element in the array is independently assigned a value (Palmer 1992), which in our case was a uniformly distributed random value ranging from 0 to 1. At the other extreme are NLMs that represent a constant environmental gradient, such as latitude, that have complete spatial autocorrelation. The simplest of these is the 'planar gradient' (Palmer 1992) that produces a linear gradient sloping in a specified or

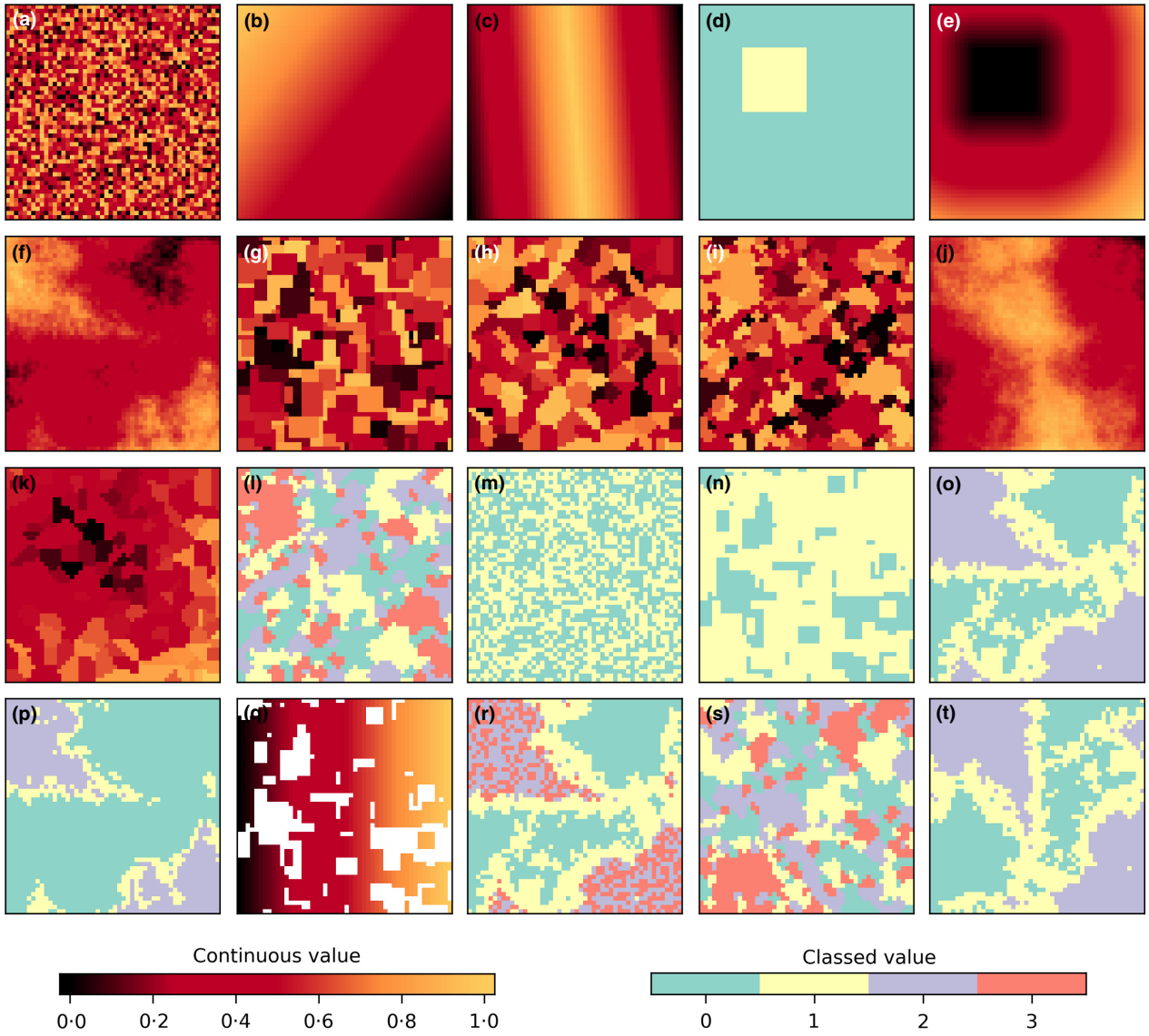

Fig. 1. Bestiary of neutral landscape models (NLMs): (a) random NLM, (b) planar gradient NLM, (c) edge gradient NLM, (d) an example of a binary mask used to create (e) a distance gradient NLM, (f) mid-point displacement NLM, (g) random rectangular cluster NLM, (h) random element nearest-neighbour NLM and (i) random cluster nearest-neighbour NLM. NLMs can also be a blend of (j) the NLMs in f and c, and (k) the NLMs in h and e. The continuous values in any NLM can be classified to produce NLMs such as (l) four equal classes from i (m) a binary percolation NLM based on a, (n) a binary NLM based on g, (o) a classified version of f and (p) a classified version of f in which the class proportions are determined using d as a classification mask. Masking can also be used to (q) limit the extent of an NLM to a binary mask such as n, and NumPy functions can (r) create an NLM that is hierarchical combination of NLMs o and m. NumPy functions can also be used to further manipulate the arrays, to produce for example (s) a rotation of l or (t) a transposition of o. Please see the supporting information for the PYTHON code used to create these NLMs.

random direction (Fig. 1b). The 'edge gradient' (Travis & Dytham 2004) produces a linear gradient orientated on a specified or random direction that has a central peak that runs perpendicular to the gradient direction (Fig. 1c). NLMpy makes use of binary arrays (Fig. 1d) with values of either 0 or 1 that act as masks to parameterise several functions. For example, we include a third gradient, the 'distance gradient', in which distances from the centres of elements classified as 1 in a binary array are calculated (Fig. 1e).

These NLMs with extremes of spatial autocorrelation do not produce realistic-looking landscape simulations – although this is a property that makes them useful for comparison with real landscapes, and as we will see later, for modifying other NLMs. To produce NLMs that have realistic patterns, the spatial autocorrelation needs to be controlled gradually.

The 'mid-point displacement' NLM (Palmer 1992) uses an algorithm (Fournier, Fussell & Carpenter 1982) in which the level of autocorrelation can be controlled (Fig. 1f). However, the mid-point displacement algorithm can only be applied to square arrays of specific sizes. Therefore, to enable a two-dimensional array of any size to be created, our algorithm creates an array that is larger than the desired extent but is the minimum-sized square that will cover desired extent, from which a random slice of the required dimensions is then extracted.

All of the NLMs introduced so far have assigned the values in an array on a per element basis to produce a pattern of gradual change across space. However, some landscapes, especially those dominated by human processes, have patterns that resemble a mosaic of uniform patches that tessellate together to form abrupt edges. An orthogonal pattern can be created using randomly sized and placed rectangular patches (Gustafson & Parker 1992). The 'random rectangular cluster' NLM randomly places rectangular clusters with random dimensions within a specified range. On each occasion, a uniformly distributed random value ranging from 0 to 1 is applied to the underlying elements, overwriting any previously assigned values, and this process is continued until all elements have a value (Fig. 1g). NLMs can be generated from point patterns by creating a nearest-neighbour (or Dirichlet, Thiessen or Voronoi) tessellation of polygons (Gaucherel 2008), but a similar approach can be taken on an array through the use of nearest-neighbour interpolation around random elements. The 'random element nearest-neighbour' NLM assigns a uniformly distributed random value ranging from 0 to 1 to a number of randomly selected elements, with nearest-neighbour interpolation used to interpolate these values to the remaining elements to create irregularly shaped and reasonably consistently sized patches (Fig. 1h).

Patchy landscapes can also occur in natural settings, and the modified random cluster (MRC) algorithm (Saura & Martínez-Millán 2000) creates patchy NLMs with more naturalistic patterns. The 'random cluster nearest-neighbour' NLM is an adaption of the MRC algorithm. As with the MRC algorithm, a series of random clusters are generated while controlling for both the proportion of the array that is within a cluster and the neighbourhood rule that affects the size and directional bias of those clusters. The MRC algorithm assigns a probability based class to each cluster, and then assigns classes to the remaining non-cluster elements using a modal filter followed by further probability based class assignment. Our 'random cluster nearest-neighbour' algorithm uses a slightly different approach to assign values to non-cluster elements that sits within our general code framework. We assign each cluster a uniformly distributed random value ranging from 0 to 1, and then use nearest-neighbour interpolation to assign values to the remainder of the array (Fig. 1i).

Blending

The NLM arrays can be easily blended to create even more patterns. For example, blending a gradient with an NLM can produce a new NLM useful for simulating ecotones (Milne *et al.* 1996) or range margins (Travis & Dytham 2004). Starting with a primary array a_p, any number n of additional arrays a can be added to the primary array with optional scaling factors s (assumed to equal 1 if not given) used to control the influence of the additional arrays (Travis & Dytham 2004). The 'blend array' function uses this approach to create a blended array $a_b = a_p + \Sigma\, a_n \times s_n$ that is then rescaled to range from 0 to 1 (Fig. 1j). When a_p has element values grouped into clusters, to retain the cluster structure of a_p the 'blend cluster array' function should be used. Before blending the arrays, this function groups the elements of a_n into the same pattern of clusters in a_p, and assigns a new value to each cluster in a_n that is the mean of the elements of a_n in each cluster (Fig. 1k).

Classification

Some NLMs consist of integers that are interpreted as nominal or ordinal landscape classes (With & Crist 1995; With, Gardner & Turner 1997; Saura & Martínez-Millán 2000). However, classifying each element or element cluster independently does not guarantee that the desired proportions are achieved (Gardner 1999; Saura & Martínez-Millán 2000). Therefore, NLMpy has a 'classify array' function based upon a list of ordered class weights w that dictate the proportion of the NLM that is assigned to each class. Once w is converted into a vector of cumulative proportions c, and once all of the elements N in the array have been rank-ordered, the index of the element at the upper class boundary b for each class i can be determined by $b_i = c_i \times N$ (With, Gardner & Turner 1997; Gardner 1999). As our algorithms all produce NLMs with values ranging from 0 to 1, this classification process can be applied to any of the NLMs to create landscape class proportions that are as close to the desired proportions as each NLM algorithm allows (Fig. 1l). We even use this classification approach to create binary NLMs developed out of percolation theory (Gardner *et al.* 1987; Gustafson & Parker 1992). Rather than using a probability parameter p to control the proportion of the landscape classified as 0 or 1, we classify a random NLM using $w = [1 - p, p]$ (Fig. 1m,n).

While the classification can be based on the full extent of the NLM (Fig. 1o), we also allow for a binary mask to be included as a classification parameter. The classification mask is used to limit which elements in the array are included when calculating the upper class boundaries (included = 1, not included = 0). This is useful as it enables the proportion of different landscape classes to be controlled within a subsection of the landscape, while at the same time enabling this subsection to be a consistent part of the whole landscape's pattern (Fig. 1p). The classification mask needs to be used with care, as it can produce landscapes dominated by the lowest and highest ranked classes when the elements within the mask have values that only cover part of the 0 to 1 range.

Masking

QRULE (Gardner & Urban 2007) introduced a masking constraint for producing NLMs within a specified spatial extent. This is useful as it enables NLMs to be created that mimic the extent of real landscapes, for example not including within the NLM areas of open water. We have incorporated this extent masking within all of our NLMs through an optional parameter that is a binary array that identifies which parts of an array are within the desired extent (Fig. 1q). When blending masked arrays, it is advisable to ensure all arrays were created using the same mask.

Hierarchical combination

The need for NLMs that mimic the hierarchical nature of landscapes has been long recognized (O'Neill, Gardner & Turner 1992). As NLMpy brings together multiple NLM algorithms within a single framework, NumPy logic functions can be used to create a hierarchy consisting of a combination of different NLMs. For example, a mid-point displacement NLM could be used to represent a continuous environmental gradient such as elevation upon which a hierarchical combination could be based. By classifying this continuous gradient, a series of sequential landscape classes can be created, but in which the lowest class may contain a finer scale pattern from a different NLM (Fig. 1r). This unique ability of NLMpy to enable the easy creation of a hierarchy consisting of a combination of different NLMs is very useful. Comparisons of different NLMs with real landscapes indicate that there is no single best NLM (Li *et al*. 2004), and we would imagine this problem would be compounded with real landscapes that have clear hierarchical structure.

Integration with GIS

The NumPy arrays that form the foundation of our approach are of course aspatial. However, a two-dimensional NumPy array data structure is equivalent to that of a GIS raster. Therefore, once a geographical position and extent is defined, it is a simple process to export a NumPy array as a raster by simply treating the value of each element in an array as the value of a cell in a raster. The 'export ASCII grid' NLMpy function will export an array as a GIS raster in the widely supported ASCII grid format. However, both open-source and proprietary GIS software now commonly use PYTHON as a scripting language, so it is a trivial task to convert between NumPy arrays and other GIS compatible raster data sets. Such conversions can also be performed solely with PYTHON using the Geospatial Data Abstraction Library (GDAL) package (Warmerdam 2008). Therefore, GIS data can easily form an input for creating a NLM that is based upon conditions for a real landscape (Fig. 2).

Discussion

We believe that our NLMpy PYTHON package will be a very useful addition to the software that modellers can use to generate NLMs. NLMpy allows NLMs to be created directly within a PYTHON modelling workflow, or by other software capable of executing a PYTHON script. NLMpy also enables the first opportunities to easily create a NLM that combines different algorithms and to use or produce GIS data. Finally, as NLMpy is based upon a series of functions that are common steps within the different NLM algorithms, it is quite likely that other NLM algorithms could be developed using the functionality contained within NLMpy. It is also worth noting that as NLMpy works within a NumPy array framework, any NLM array could be readily manipulated further using any of the vast number of methods available in NumPy and SciPy. Simple

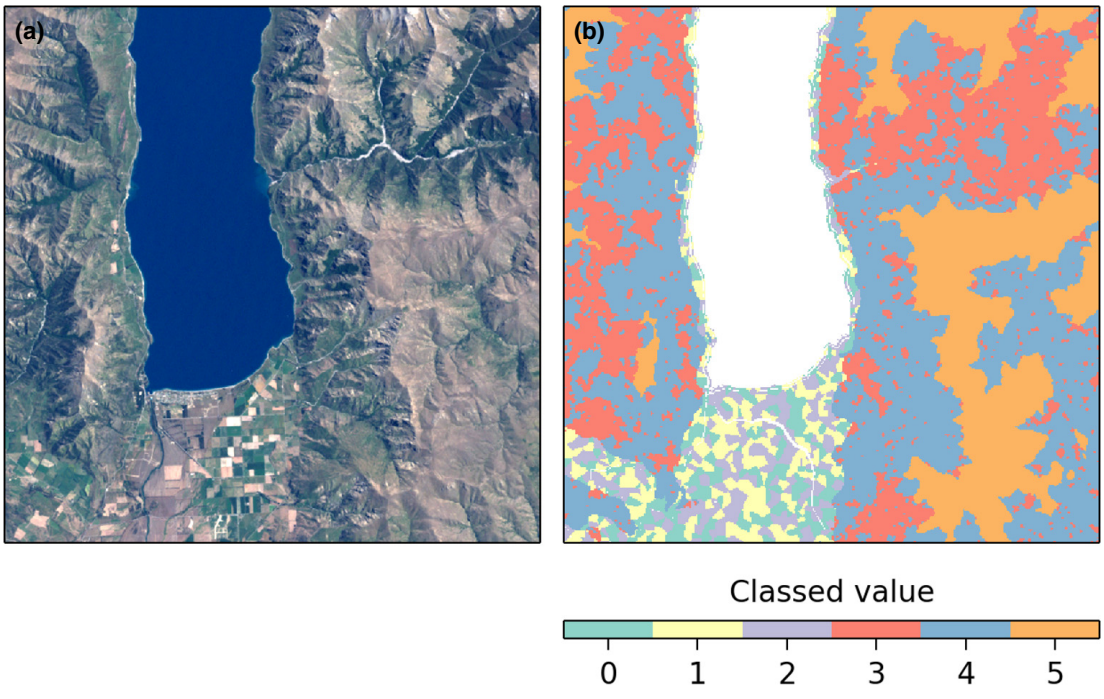

Fig. 2. An example of using geographic information system (GIS) data to create a neutral landscape model (NLM) that mimics a real landscape. The real landscape in (a) shows a hierarchical structure with a mosaic of angular patches in the valley bottom, a mixture of irregularly shaped forestry and pasture on the mountain slopes, and rough grassland on top of the mountains. Using GIS elevation data as an input, the NLM in (b) can be created to mimic these patterns by applying different NLMs at different elevation ranges, while also ignoring the area cover by the lake. Please see the supporting information for the PYTHON code used to create this NLM.

examples would include the ability to rotate (Fig. 1s), flip or transpose (Fig. 1t) the NLM array.

In creating a PYTHON package rather than a standalone bespoke piece of software that is operated via a graphical user interface, we are assuming that potential users will have the necessary scripting skills for languages such as PYTHON or R to make use of the functionality we have created. However, we do not think this is an unreasonable assumption as using NLMpy to create and export a mid-point displacement NLM can be achieved with only three lines of code:

```
>>> import nlmpy
>>> nlm=nlmpy.mpd(nRow=50,nCol=50,h=0.75)
>>> nlmpy.exportASCIIGrid("raster.asc",nlm)
```

We believe this simplicity should make NLMpy an accessible method for generating NLMs, even though we cannot give exhaustive instructions on how to use NLMpy that will meet the requirements of all potential users. However, to encourage the use of NLMpy, we provide as supporting information a function reference card and the PYTHON scripts and GIS data that generate our example NLMs (Figs 1 and 2b) that can be copied, extended or adapted to specific needs.

Acknowledgements

This work was funded by the University of Auckland Faculty Research Development Fund 3702237.

Data accessibility

The NLMpy reference card and example PYTHON scripts and GIS data are available as supporting information. NLMpy and the example scripts were developed using: PYTHON 2.7.3, NumPy 1.8.0, SciPy 0.13.2 and Matplotlib 1.3.1 – all of which are freely available online for all computer platforms. NLMpy itself has been made freely available under the MIT License and is available for installation as a package via the PYTHON Package Index (https://pypi.python.org/pypi/nlmpy).

References

Chipperfield, J.D., Dytham, C. & Hovestadt, T. (2011) An updated algorithm for the generation of neutral landscapes by spectral synthesis. *PLoS One*, **6**, e17040.

Fournier, A., Fussell, D. & Carpenter, L. (1982) Computer rendering of stochastic models. *Communications of the ACM*, **25**, 371–384.

Gardner, R.H. (1999) RULE: map generation and a spatial analysis program. *Landscape Ecological Analysis: Issues and Applications* (eds J.M. Klopatek & R.H. Gardner), pp. 280–303. Springer-Verlag, New York.

Gardner, R.H. & Urban, D.L. (2007) Neutral models for testing landscape hypotheses. *Landscape Ecology*, **22**, 15–29.

Gardner, R.H., Milne, B.T., Turner, M.G. & O'Neill, R.V. (1987) Neutral models for the analysis of broad-scale landscape pattern. *Landscape Ecology*, **1**, 19–28.

Gaucherel, C. (2008) Neutral models for polygonal landscapes with linear networks. *Ecological Modelling*, **219**, 39–48.

Gustafson, E.J. & Parker, G.R. (1992) Relationships between landcover proportion and indices of landscape spatial pattern. *Landscape Ecology*, **7**, 101–110.

Li, X.Z., He, H.S., Wang, X.G., Bu, R.C., Hu, Y.M. & Chang, Y. (2004) Evaluating the effectiveness of neutral landscape models to represent a real landscape. *Landscape and Urban Planning*, **69**, 137–148.

Milne, B.T., Johnson, A.R., Keitt, T.H., Hatfield, C.A., David, J. & Hraber, P.T. (1996) Detection of critical densities associated with piñon-juniper woodland ecotones. *Ecology*, **73**, 805–821.

Oliphant, T.E. (2007) Python for scientific computing. *Computing in Science & Engineering*, **9**, 10–20.

O'Neill, R.V., Gardner, R.H. & Turner, M.G. (1992) A hierarchical neutral model for landscape analysis'. *Landscape Ecology*, **7**, 55–61.

Palmer, M.W. (1992) The coexistence of species in fractal landscapes. *The American Naturalist*, **139**, 375–397.

Saura, S. & Martínez-Millán, J. (2000) Landscape patterns simulation with a modified random clusters method. *Landscape Ecology*, **15**, 661–678.

Travis, J.M.J. & Dytham, C. (2004) A method for simulating patterns of habitat availability at static and dynamic range margins. *Oikos*, **104**, 410–416.

Turner, M.G., Gardner, R.H. & O'Neill, R.V. (2001) *Landscape Ecology in Theory and Practice: Pattern and Process*. Springer, New York.

Warmerdam, F. (2008) The geospatial data abstraction library. *Open Source Approaches in Spatial Data Handling* (eds G.B. Hall & M.G. Leahy), pp. 87–104. Springer, Berlin.

With, K.A. & Crist, T.O. (1995) Critical thresholds in species' responses to landscape structure. *Ecology*, **76**, 2446–2459.

With, K.A., Gardner, R.H. & Turner, M.G. (1997) Landscape connectivity and population distributions in heterogeneous environments. *Oikos*, **78**, 151–169.

With, K.A. & King, A.W. (1997) The use and misuse of neutral landscape models in ecology. *Oikos*, **79**, 219–229.

Received 29 September 2014; accepted 5 November 2014
Handling Editor: Timothée Poisot

Supporting Information

Additional Supporting Information may be found in the online version of this article.

Data S1. A reference card detailing the functions of NLMpy.

Data S2. Python script to generate the NLMs shown in Fig. 1.

Data S3. Python script to generate the NLM shown in Fig. 2b.

Data S4. Elevation data used by the Data S3 Python script.