Journal of Statistical Software

July 2016, Volume 71, Issue 7. doi: 10.18637/jss.v071.i07

rft1d: Smooth One-Dimensional Random Field Upcrossing Probabilities in Python

Todd Pataky
Shinshu University

Abstract

Through topological expectations regarding smooth, thresholded n-dimensional Gaussian continua, random field theory (RFT) describes probabilities associated with both the field-wide maximum and threshold-surviving upcrossing geometry. A key application of RFT is a correction for multiple comparisons which affords field-level hypothesis testing for both univariate and multivariate fields. For unbroken isotropic fields just one parameter in addition to the mean and variance is required: the ratio of a field's size to its smoothness. Ironically the simplest manifestation of RFT (1D unbroken fields) has rarely surfaced in the literature, even during its foundational development in the late 1970s. This Python package implements 1D RFT primarily for exploring and validating RFT expectations, but also describes how it can be applied to yield statistical inferences regarding sets of experimental 1D fields.

Keywords: random field theory, Gaussian random fields, multivariate analysis, time series, continuum analysis.

1. Introduction

1.1. Theory

Random field theory (RFT) extends Gaussian behavior to smooth, continuous n-dimensional (nD) processes, and in particular describes the probability that Gaussian fields (Figure 1) will reach particular heights when acting over arbitrary nD geometries (Adler and Taylor 2007). The primary application of this probability is to correct for multiple comparisons across the entire field, thereby affording field-level hypothesis testing (Worsley, Taylor, Tomaiuolo, and Lerch 2004; Friston, Ashburner, Kiebel, Nichols, and Penny 2007).

RFT inferences stem from the expected topological features of an nD Gaussian field which

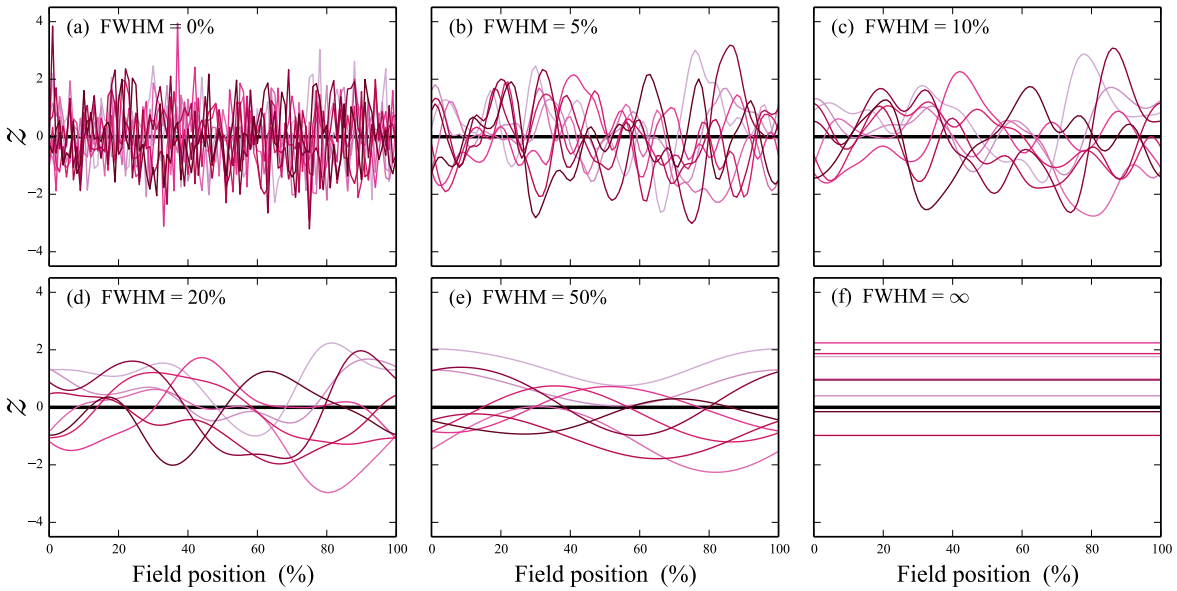

Figure 1: Gaussian 1D fields of varying smoothnesses; FWHM is the full-width-at-half maximum of a Gaussian smoothing kernel. (a) Uncorrelated Gaussian noise. (b–e) Same data as (a), but convolved with increasingly broad Gaussian kernels and rescaled to unit variance. (f) Infinitely smooth fields, equivalent to 0D Gaussian scalars.

has been thresholded at a value u. The two key topological features of interest are the Euler characteristic (EC) and the Hadwiger characteristic (HC), and in the 1D case both the EC and HC directly represent the number of threshold upcrossings (Figure 2). In the RFT literature, topological expectations were derived first for the EC and for two- and three-dimensional fields (Adler and Hasofer 1976), and were subsequently generalized to n-dimensional fields (Hasofer 1978, Equation 4) as follows:

$$\mathsf{E}(EC) = \lambda(S)(2\pi)^{-\frac{1}{2}(n+1)} |A|^{\frac{1}{2}} P(u) e^{-u^2/2\sigma^2}, \tag{1}$$

where:

- EC is the Euler characteristic at threshold u over the finite nD space S,
- $\lambda(S)$ is the Lebesgue measure of S,
- $|A|$ is the determinant of the covariance matrix of the field's partial derivatives,
- $P(u)$ is the Hermite polynomial:

$$P(u) = \sum_{j=0}^{[(n-1)/2]} (-1)^j \frac{(2j)!}{j!2^j} \binom{n-1}{2j} \sigma^{2j} u^{n-1-2j}. \tag{2}$$

Equation 1 suggests that only one additional parameter, in addition to the mean and standard deviation, is required to describe smooth nD Gaussian behavior: the ratio of the field size $\lambda(S)$ to its smoothness $|A|$.

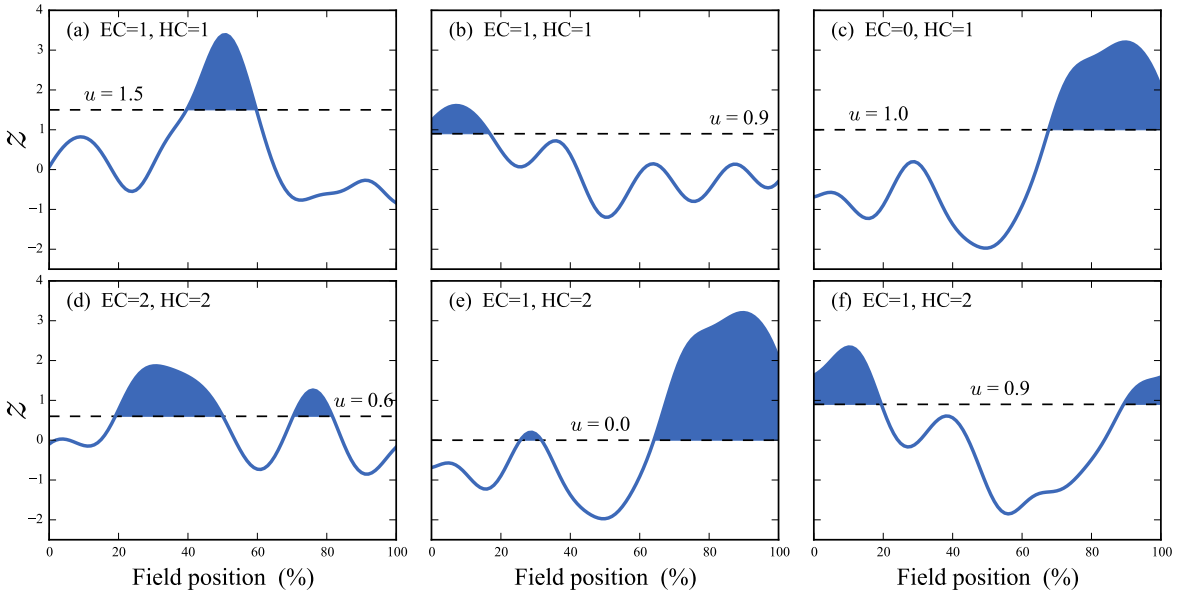

Figure 2: Topological features of smooth 1D fields which have been thresholded at u. EC = Euler characteristic, HC = Hadwiger characteristic. Whereas the HC increments for each upcrossing, the EC increments only if the upcrossing does not touch the right boundary.

Using the expected EC and HC in classical hypothesis testing is straightforward: One simply finds the height u^* for which the expected EC or HC is α, and the null hypothesis is rejected if the observed test statistic field traverses u^*. In practice a Poisson heuristic is often also used to correct for multiple upcrossings (Friston *et al.* 2007). Thus, when coupled with a Poisson heuristic, the expected EC or HC can directly control the family-wise error rate α across the whole field, thereby affording field-wide analysis in a single hypothesis test.

RFT has been applied most widely to 3D/4D functional neuroimaging analyses (Friston *et al.* 2007) via extensions of the expected EC to test statistic fields including t, F and χ^2 (Worsley 1995), Hotelling's T^2 (Cao and Worsley 1999), Roy's maximum root (Worsley *et al.* 2004) and Wilks's Λ (Carbonell, Worsley, and Galan 2011). Since the most common software implementations of RFT are also 3D/4D, RFT accessibility is naturally limited by two factors: data visualization and algorithmic sophistication (to cope with large data volumes). Fortunately, both limitations can be easily overcome by considering the simpler 1D case. In particular:

1. 1D Gaussian fields can be easily computed and visualized (Figure 1).
2. The key theoretical concept – expected topology – simplifies to the expected number of threshold-surviving maxima, which can also be easily visualized in 1D (Figure 2).
3. Theoretical validation, via random field generation, can be conducted effectively instantaneously.

Moreover, Equation 1 reduces to a surprisingly simple form when $n = 1$:

$$\mathsf{E}(EC) = \frac{S}{W}\frac{\sqrt{4\log 2}}{2\pi}\, e^{-u^2/2}, \tag{3}$$

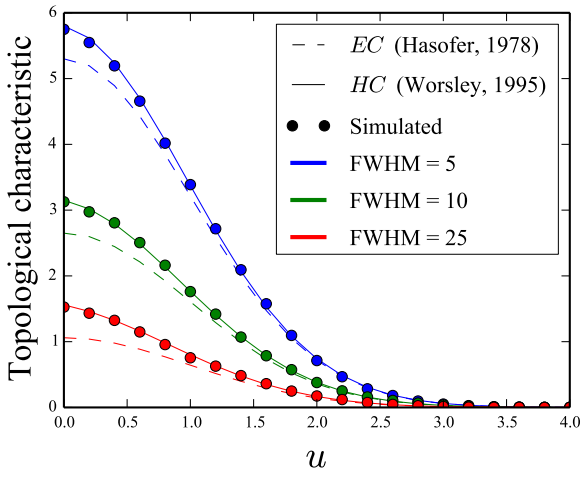

Figure 3: Validation of topological expectations for smooth, unbroken 1D Gaussian random fields with size $S = 100$. The variables EC and HC are the Euler (Equation 3) and Hadwiger (Equation 5) characteristics, respectively. These validations were performed using the **rft1d** random field generator: `rft1d.randn1d`.

where S is the field length and where W is the full-width-at-half-maximum (FWHM) of a Gaussian kernel used to smooth uncorrelated Gaussian fields (Figure 1):

$$W = \sqrt{\frac{4 \log 2}{A}}. \tag{4}$$

Although quite simple, Equation 3 quite accurately predicts the EC for high thresholds u (Figure 3). Its main limitation is that threshold upcrossings can touch the field boundary, and thereby affect the EC (Figure 2). Worsley (1995) solved this problem using the Hadwiger characteristic (HC), which is invariant to boundary touches and equivalent to the EC otherwise, and which more accurately describes Gaussian field topology (Figure 3). The solution given for $n = 2$ in Worsley (1995, Theorem 2) reduces to, for an unbroken, bounded field of unit variance and $n = 1$ to:

$$\mathsf{E}(HC) = \mathsf{P}(X \geq u) + \frac{S}{W} \frac{\sqrt{4 \log 2}}{2\pi} e^{-u^2/2}, \tag{5}$$

where $\mathsf{P}(X \geq u)$ is the usual Gaussian survival function. For broken Gaussian fields (Figure 4), for which S now represents the total unbroken field length, one needs simply to multiply the survival function by the number of unbroken field segments c (Worsley 1995):

$$\mathsf{E}(HC) = c\Big(\mathsf{P}(X \geq u)\Big) + \frac{S}{W} \frac{\sqrt{4 \log 2}}{2\pi} e^{-u^2/2}. \tag{6}$$

Broken-field expectations are also easily validated (Figure 5). The key result is thus that only two parameters (c and S/W) are required to accurately specify the topological behavior of both broken and unbroken Gaussian fields. These two parameters have been termed "resolution element counts" or "resel counts" (Worsley, Marrett, Neelin, Vandal, Friston, and Evans

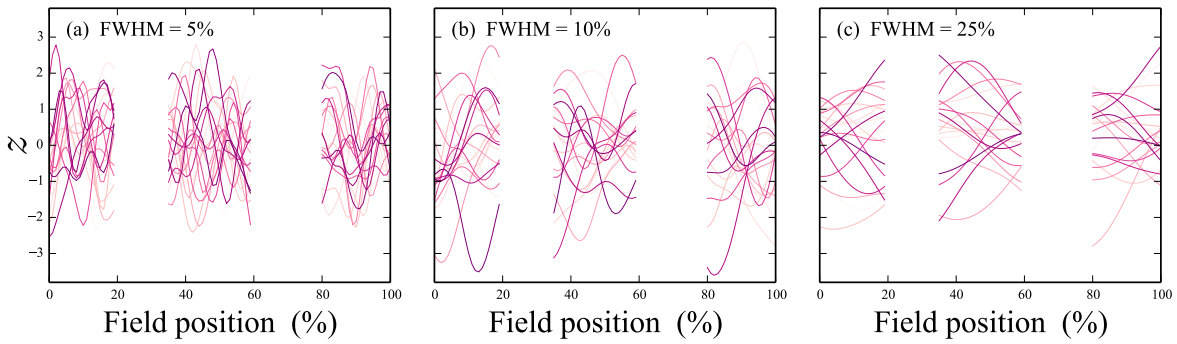

Figure 4: Broken (piecewise continuous) Gaussian 1D fields. These fields have undefined heights over the intervals $[20, 35\%]$ and $[60, 80\%]$, and thus a total of $c = 3$ unbroken field segments (see Equation 6).

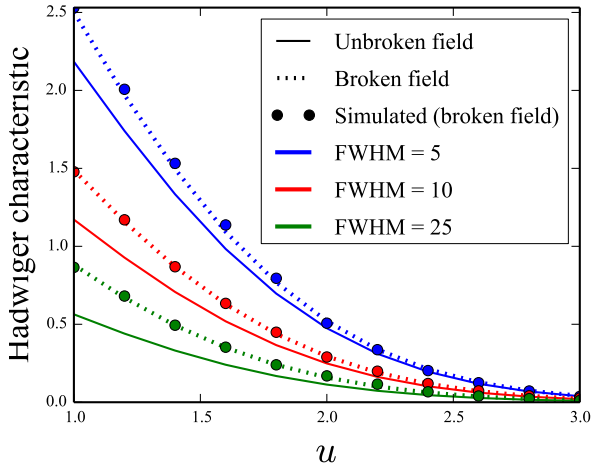

Figure 5: Validation of the expected Hadwiger characteristic (Equation 6) for the broken fields depicted in Figure 4. Although the number of unbroken field segments is actually $c = 3$, the "unbroken field" results use $c = 1$.

1996) to emphasize the fact that these parameters specify field geometry in a manner which is invariant to sampling frequency.

From this yearamental topological expectation stems many probabilities regarding Gaussian field behavior (Friston *et al.* 2007). Three key probabilities include:

1. *Height-based inference*: the probability that Gaussian fields or test statistic fields will reach a particular height.

2. *Cluster-level inference*: the probability that Gaussian fields or test statistic fields, when thresholded at u, will produce an upcrossing (or "cluster") with a particular extent k.

3. *Set-level inference*: the probability that Gaussian fields or test statistic fields, when thresholded at u, will produce at least c upcrossings with a minimum extent of k.

The overall goal of the **rft1d** software package is to make these RFT concepts accessible and

explorable for students and researchers who deal with 1D field data including time series.

1.2. Comparison with other software packages

The software packages most closely related to **rft1d** are:

1. **SPM12** (Friston *et al.* 2014);
2. **fmristat** (Worsley 2006);
3. **nipy** (Millman and Brett 2007);
4. **spm1d** (Pataky 2012).

The first two packages were written for MATLAB (The MathWorks, Inc. 2014) and the last two were written for Python (van Rossum 2014). The **SPM12**, **fmristat** and **nipy** packages primarily target the field of Neuroimaging, and in particular the analysis of 3D functional magnetic resonance images (fMRI). These packages offer a comprehensive suite of tools for fMRI data analysis, and all implement three-dimensional RFT. From the perspective of a user who wishes to learn RFT concepts, these three fMRI-related packages are non-ideal for a number of reasons:

- *Software bulk* – Since these packages target applied image analysis, a substantial portion of their functionality involves 3D image management, processing and visualization, so finding the RFT-related code and understanding how it interacts with the rest of the software can be challenging and time-consuming.
- *Experimental modeling complexity* – Experimental design matrices for most fMRI experiments require a time-dependent model of the brain's hemodynamic response which is observable in the raw fMRI data (Friston *et al.* 2007). This complicates the relation amongst single images, residual images, and test statistic images. Thus the connection between the original fMRI data and 3D Gaussian random fields is not evident until one appreciates how the model residuals are generated.
- *Three-dimensional RFT* – Although all three fMRI-related packages also support one- and two-dimensional RFT, they primarily target three-dimensional RFT. This naturally introduces two important barriers to generalized RFT adoption:
  - *Code complexity*: Three-dimensional code is naturally more complex than is needed for 1D and 2D analysis.
  - *Random field exploration*: It is much easier to visualize 1D random and 2D random fields than it is to visualize 3D random fields. Similarly, it is much easier to simultaneously visualize multiple 1D random fields (Figure 1) than it is to simultaneously visualize multiple 2D random fields. Thus the randomness that RFT models is clearest for 1D data.

The fourth package (**spm1d**) partially addresses these limitations. In particular, it compactly focusses on 1D data, and utilizes simple models which much more closely retain the connection

between the raw 1D data and 1D residuals. However, it is a preliminary applied package which only implements univariate analyses and which does not provide any conceptual aids for the RFT core which drives its probability computations. Moreover, it fails to implement a number of RFT-relevant computations including those pertaining to: (i) broken fields, (ii) multiple test statistic fields in conjunction, (iii) circular fields, and (iv) node-based vs. element-based field sampling.

The goals of the **rft1d** package are:

- To promote RFT learning through exclusive focus on one-dimensional fields;
- To simplify and accelerate existing one-dimensional RFT implementations;
- To introduce a novel high-level interface to core RFT expectations and probabilities which uses minimum input parameter sets, and which does not require special data structures as input arguments;
- To provide a set of scripts which emphasize RFT concepts and which validate RFT predictions;
- To implement RFT procedures for: (i) broken fields, (ii) multiple test statistic fields in conjunction, (iii) circular fields, and (iv) node-based vs. element-based field sampling;
- To provide HTML documentation regarding all aforementioned RFT concepts.

In addition to facilitating RFT explorations (Section 3) and validations (Section 4), the **rft1d** package supports practical statistical inference. An example of classical hypothesis testing, using both RFT and a mirroring non-parametric technique, is described in Section 5.

2. Python implementation

Dependencies of **rft1d** include: Python 2.7, **NumPy** 1.8 (van der Walt, Colbert, and Varoquaux 2011), **SciPy** 0.14 (Jones, Oliphant, Peterson *et al.* 2001) and, to run its example scripts: **matplotlib** 1.3 (Hunter 2007). Other versions of these dependencies should also work but have not yet been tested thoroughly. **rft1d** was developed using **Canopy** 1.4 (Enthought, Inc. 2014).

The **rft1d** package consists of four modules:

1. `distributions` – a **SciPy**-like interface to RFT expectations and probabilities;
2. `geom` – functions/classes for calculating geometrical features of fields/upcrossings;
3. `prob` – core RFT probability calculations along with a high-level interface;
4. `random` – smooth 1D Gaussian field generation.

The core module `rft1d.prob` translates, simplifies and accelerates existing MATLAB (The MathWorks, Inc. 2014) implementations of 3D RFT (**SPM8** and **SPM12b**) which are both open-source and available from: `http://www.fil.ion.ucl.ac.uk/spm`. Unlike the 3D RFT

implementation in the **SPM** software, **rft1d** exclusively focusses on 1D fields, and also provides a novel high-level interface that emphasizes which parameters are required for particular expectation and probability calculations.

Users interested in reproducing **SPM8** and **SPM12** results can do so using the function `rft1d.prob.rft`, but otherwise users are encouraged to initially interact with the high-level interface in `rft1d.distributions`, which is also accessible from the top-level package interface in a **SciPy**-like manner. For example, the Gaussian survival function at a height $u = 2.0$ can be computed as follows.

```
> from scipy import stats
> u = 2.0
> p = stats.norm.sf(u)
```

In **rft1d**, the Gaussian survival function requires two additional parameters: the number of discrete field nodes (`nNodes`) and the field smoothness (`FWHM`).

```
> import rft1d
> u = 3.0
> nNodes = 101
> FWHM = 10.0
> p = rft1d.norm.sf(u, nNodes, FWHM)
```

This yields $p = 0.030$.

rft1d also implements four test statistic distributions: Student's t, χ^2, F and Hotelling's T^2. Survival functions for these distributions require an additional degrees of freedom parameter which, to follow **SciPy** syntax, should follow the height u.

```
> p = rft1d.t.sf(u, 8, nNodes, FWHM)
> p = rft1d.chi2.sf(u, 8, nNodes, FWHM)
> p = rft1d.f.sf(u, (2, 15), nNodes, FWHM)
> p = rft1d.T2.sf(u, (2, 15), nNodes, FWHM)
```

The resulting probabilities are 0.180, 0.934, 0.744 and 0.964.

Inverse survival functions are accessible via the `isf` method:

```
> alpha = 0.05
> u = rft1d.norm.isf(alpha, nNodes, FWHM)
> u = rft1d.t.isf(alpha, 8, nNodes, FWHM)
> u = rft1d.chi2.isf(alpha, 8, nNodes, FWHM)
> u = rft1d.f.isf(alpha, (2, 15), nNodes, FWHM)
> u = rft1d.T2.isf(alpha, (2, 15), nNodes, FWHM)
```

The resulting critical heights are 2.826, 4.115, 24.433, 10.063 and 22.421, respectively.

For more convenient theoretical explorations (Figure 6), multiple heights can be submitted to the survival functions, and multiple Type I error rates can be submitted to the inverse survival functions as follows:

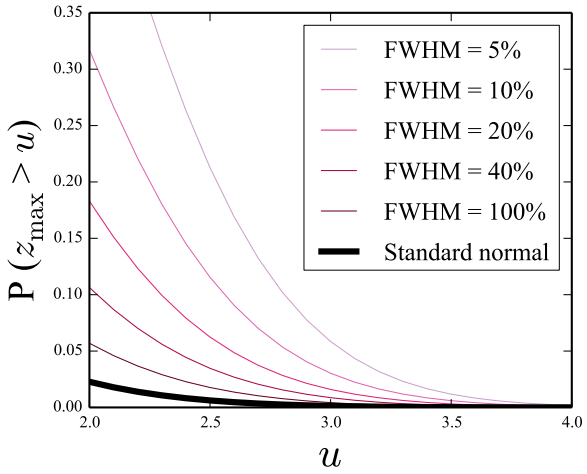

Figure 6: Survival functions for Gaussian field maxima ($z_{\max}$) as a function of smoothness.

```
> import numpy as np
> heights = np.linspace(2, 4, 51)
> sf = rft1d.norm.sf(heights, nNodes, FWHM)
> from matplotlib import pyplot
> pyplot.plot(heights, sf)
> alpha = [0.001, 0.01, 0.05, 0.1]
> u = rft1d.norm.isf(alpha, nNodes, FWHM)
```

"Cluster-level" inferences (Section 1) can similarly be accessed through the interface provided by `rft1d.distribution` using the observed cluster extent k, which should be in resels units. For example, the FWHM-dependent probability that Gaussian random fields will yield an upcrossing which spans 1.7 nodes is given by:

```
> FWHM = 8.0
> k = 1.7 / FWHM
> u = 3.0
> p = rft1d.norm.p_cluster(k, u, nNodes, FWHM)
```

The result is $p = 0.032$.

"Set-level" inferences (Section 1) require just one more parameter (c), the number of upcrossings. In this case k should be the minimum upcrossing extent (in resels units).

```
> c = 2
> FWHM = 8.0
> k = 0.7 / FWHM
> u = 3.0
> p = rft1d.norm.p_set(c, k, u, nNodes, FWHM)
```

The result is $p = 0.00067$.

rft1d implements a variety of additional functionality including, for example: a common interface for both unbroken and broken fields, and adjustments for very rough fields (for which

the Bonferroni correction may be less conservative). This functionality, along with complete API details, are described in the HTML documentation in the supplementary material.

3. Exploring RFT expectations and probabilities

rft1s implements two separate interfaces for exploring RFT expectations and probabilities. The first is a procedural-type interface, as briefly outlined above in Section 2. The second is an object-oriented interface which facilitates systematic explorations of particular field geometries. The object-oriented interface stems from objects of the '`RFTCalculator`' class as follows:

```
> STAT = "T"
> df = (1, 19)
> nNodes = 101
> FWHM = 10.0
> calc = rft1d.prob.RFTCalculator(STAT, df, nNodes, FWHM)
```

The `STAT` variable should be one of: `"Z"`, `"T"`, `"X2"`, `"F"` and `"T2"` which respectively reference the Gaussian, Student's t, χ^2, F and Hotelling's T^2 distributions.

Now that the calculator is instantiated, a variety of expectations can be accessed conveniently through its "expected" methods by specifying a height of u:

```
> u = 3.0
> c = calc.expected.number_of_upcrossings(u)
> k_nodes = calc.expected.nodes_per_upcrossing(u)
> k = calc.expected.resels_per_upcrossing(u)
> N = calc.expected.number_of_suprathreshold_nodes(u)
```

RFT probabilities can also be accessed in a similar fashion, using the number c and extent k of upcrossings. The probabilities of observing (i) at least one upcrossing at a height $u = 3.0$, (ii) an upcrossing at $u = 3.0$ with an extent of $k = 0.2$ resels, and (iii) $c = 2$ upcrossings at $u = 3.0$ with a minimum extent of $k = 0.2$ resels are given by the following code:

```
> u = 3.0
> k = 0.2
> c = 2
> p = calc.p.upcrossing(u)
> p = calc.p.cluster(k, u)
> p = calc.p.set(c, k, u)
```

This interface emphasizes that, in addition to the typical degrees of freedom parameter, only basic geometric information (`nNodes`, `FWHM`) is needed to generate rather comprehensive expectations and probabilities.

An important theoretical concept is the intersection of RFT and Bonferroni probabilities. Following Nyquist theory, when fields become very rough one runs the risk of under-sampling the field process. In this case a Bonferroni correction across field nodes may be less conservative than the RFT correction because numerical derivatives become poor estimates of the

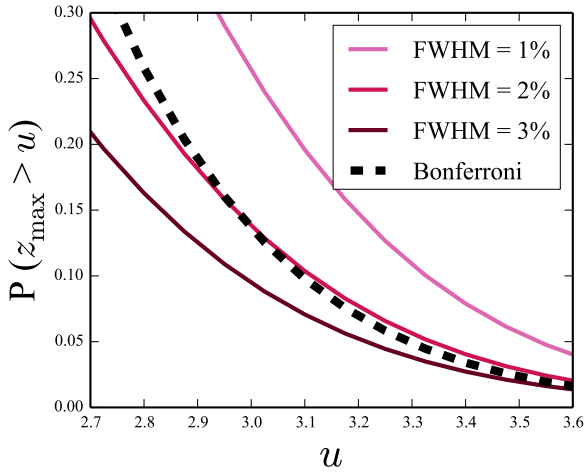

Figure 7: Survival functions for rough fields. The Bonferroni correction (across 101 nodes) is less conservative than the RFT correction for high thresholds ($u > 3$) when the fields are rough (FWHM $< 2\%$).

true local smoothness (Figure 7). To explore this issue one can first compute the critical Bonferroni-corrected height using `scipy.optimize` as follows:

```
> import scipy.optimize
> import rft1d
> STAT = "Z"
> df = None
> nNodes = 101
> alpha = 0.05
> p = lambda x: rft1d.prob.p_bonferroni(STAT, x, df, nNodes)
> objective_fn = lambda x:(p(x) - alpha) ** 2
> x0 = 5.0
> u = scipy.optimize.fmin(objective_fn, x0)
```

This yields a critical height of $u = 3.293$. Next, the RFT inverse survival function can be computed as a function of the FWHM using `rft1d.norm.isf` as outlined above in Section 2. From the resulting isoprobability contours (Figure 8) it is clear that the RFT and Bonferroni isoprobabilities cross at a particular FWHM value, and for smaller FWHMs it is more powerful to adopt the Bonferroni approach. All relevant **rft1d** procedures allow one to control acceptance of Bonferroni correction via the keyword `withBonf` as follows:

```
> u = 3.0
> nNodes = 101
> FWHM = 1.5
> p0 = rft1d.norm.sf(u, nNodes, FWHM, withBonf = False)
> p1 = rft1d.norm.sf(u, nNodes, FWHM, withBonf = True)
```

In this case the pure RFT probability, which ignores the Bonferroni correction, is $p_0 = 0.179$. The unified probability, which adopts the least conservative threshold, is $p_1 = 0.136$. For sufficiently smooth fields the keyword has no effect:

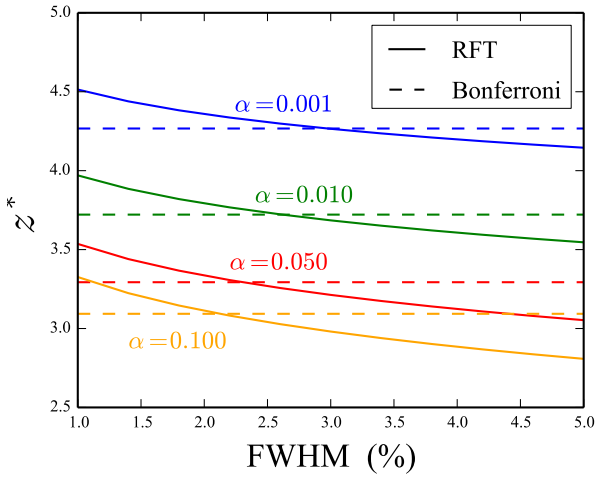

Figure 8: Inverse survival function isoprobabilities: RFT vs. Bonferroni approaches. The RFT functions are smoothness-dependent but the Bonferroni functions are not.

```
> u = 3.0
> nNodes = 101
> FWHM = 5.0
> p0 = rft1d.norm.sf(u, nNodes, FWHM, withBonf = False)
> p1 = rft1d.norm.sf(u, nNodes, FWHM, withBonf = True)
```

In this case $p_0 = p_1 = 0.058$.

4. Validating RFT expectations

While previously published 1D RFT validations exist for relatively long fields ($S > 4000$) with relatively low FWHMs (Friston, Worsley, Frackowiak, Mazziotta, and Evans 1994) there are no validation results for short fields, and in particular for $S = 100$, which is a natural choice for data interpolation (i.e., 0% to 100%).

4.1. Random field generation

Gaussian 1D fields can be generated using the `rft1d.randn1d` function:

```
> import rft1d
> y = rft1d.random.randn1d(20, 101, FWHM = 10)
```

The function can also be accessed from the top-level interface:

```
> y = rft1d.randn1d(20, 101, FWHM = 10)
```

This yields 20 Gaussian fields sampled at $Q = 101$ nodes ($S = 100$) with an FWHM of 10% (Figure 1c), where the output `y` is an (20×101) **NumPy** array. Reproducing the identical fields can be done using `numpy.random.seed` as follows:

```
> import numpy as np
> np.random.seed(0)
> yA = rft1d.randn1d(5, 101, 20.0)
> yB = rft1d.randn1d(5, 101, 20.0)
> np.random.seed(0)
> yC = rft1d.randn1d(5, 101, 20.0)
```

In this case `yA` and `yB` are different, but `yA` and `yC` are identical.

Broken fields (Figure 4) can be produced using a binary field array:

```
> b = np.array([ True ] * 101)
> b[20 : 30] = False
> b[60 : 80] = False
> y = rft1d.randn1d(20, b, FWHM = 10)
```

This yields 20 Gaussian fields with an FWHM of 10%, but with undefined heights over the intervals [20, 30%] and [60, 80%], similar to the fields depicted in Figure 1b.

4.2. Field-wide maxima

Since the fields are stored as **NumPy** arrays their maxima can be easily extracted using **NumPy** array methods:

```
> ymax = y.max(axis = 1)
```

and these maxima can be compared to the theoretical expectation via the **rft1d** survival functions as follows:

```
> nNodes = 101
> FWHM = 10.0
> u = 3.0
> p_simulated = (ymax > u).mean()
> p_expected = rft1d.norm.sf(u, nNodes, FWHM)
```

This procedure can easily be extended to validate the results of Figure 6, as depicted in Figure 9.

4.3. Cluster-level inference

Upcrossing extents can be easily computed using various **SciPy** functions like `scipy.ndimage.label`, but since upcrossing extent computations can involve various complications like interpolation (Figure 10b) and wrapping (for circular fields), it may be more convenient to use **rft1d**'s '`ClusterMetricCalculator`' class as follows:

```
> nNodes = 101
> FWHM = 10.0
> u = 3.0
> calc = rft1d.geom.ClusterMetricCalculator()
```

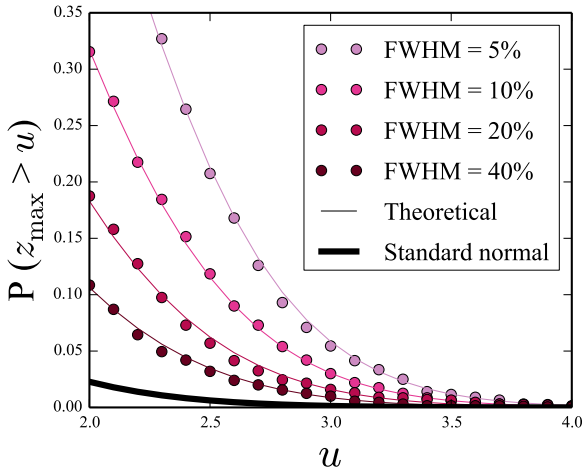

Figure 9: Validation of the RFT Gaussian survival function for field maxima ($z_{\max}$).

```
> interp = True
> wrap = False
> y = rft1d.randn1d(1000, nNodes, FWHM)
> kmax = [calc.max_cluster_extent(yy, u, interp, wrap) for yy in y]
```

These maximum cluster extents can then be compared to RFT expectations as follows:

```
> k0_nodes = 2.0
> k0_resels = k0_nodes / FWHM
> p_simulated = (np.array(kmax) >= k0_nodes).mean()
> p_expected = rft1d.norm.p_cluster(k0_resels, u, nNodes, FWHM)
```

Repeating this procedure for a variety of height and cluster extent thresholds shows close agreement between simulated and theoretical results (Figure 11).

4.4. Set-level inference

Set-level inference requires one to simply count the number of upcrossings which exceed a particular extent threshold $k_{\min}$ when thresholded at u. This can then be compared to the theoretical expectation as follows:

```
> nNodes = 101
> FWHM = 10.0
> u = 3.0
> kmin_resels = 2.0 / FWHM
> c = 2
> p_expected = rft1d.norm.p_set(c, kmin_resels, u, nNodes, FWHM)
```

Repeating for a variety of cluster extent thresholds ($k_{\min}$) and height thresholds (u) yields simulation results which follow theoretical expectation reasonably well (Figure 12) but as reported elsewhere (Barnes, Ridgway, Flandin, Woolrich, and Friston 2013) simulated probabilities tend to be lower than predicted. Since the prediction error is in the conservative

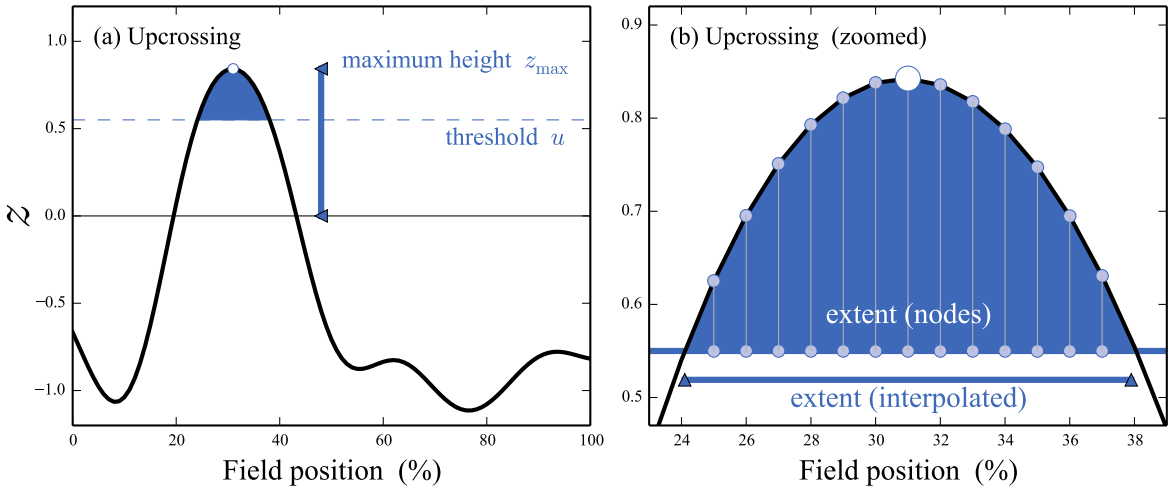

Figure 10: A threshold-surviving upcrossing. The maximum height metric $y_{\max}$ describes the whole field. When thresholded at a particular height u, surviving clusters can be characterized using various metrics including extent, integral, etc. Given the field smoothness W, two key probabilities are: $\mathsf{P}(z_{\max}|W)$ and $\mathsf{P}[k|(u,W)]$, where k is the upcrossing extent.

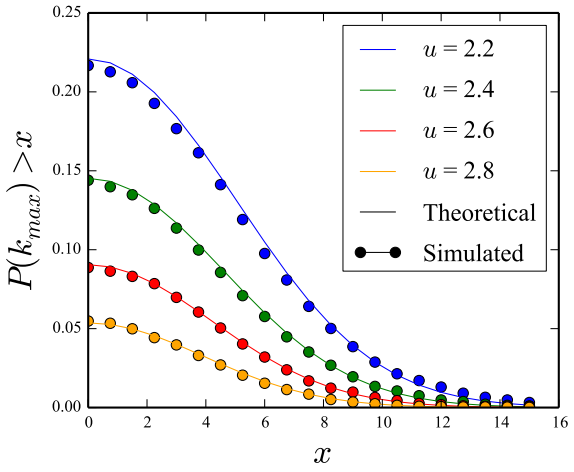

Figure 11: Validation of cluster-level inference: The probability that Gaussian fields will produce an upcrossing with an extent of x when thresholded at a height u.

direction it is not considered a serious limitation. More accurate set-level inference procedures have recently been developed (Barnes *et al.* 2013) and will be incorporated into future versions of **rft1d**.

4.5. Other validations

Only the main validations (field maxima, cluster-level inference and set-level inference) were described above, and only for unbroken Gaussian fields. Validations for (i) test statistic fields, (ii) broken fields, (iii) univariate and multivariate procedures, and (iv) test statistic fields in conjunction are available in the `./rft1d/examples` directory in scripts named "`val*.py`".

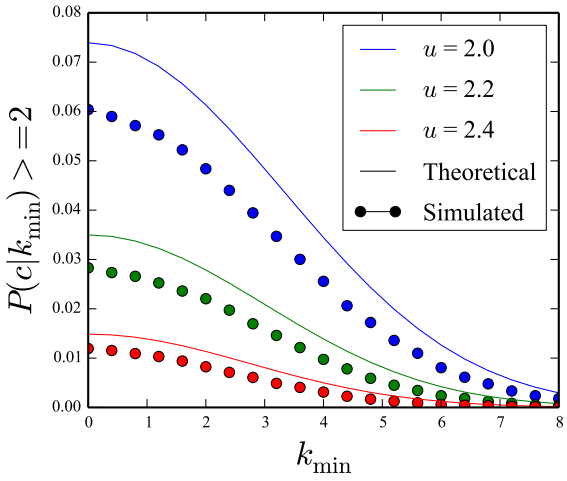

Figure 12: Validation of set-level inference: The probability that Gaussian fields will produce c or more upcrossings (here $c = 2$) with a minimum extent $k_{\min}$ when thresholded at u.

5. Example application

This section describes how RFT can be used for classical hypothesis testing on 1D fields, and also presents a non-parametric procedure which mirrors the parametric approach.

5.1. Dataset

A public weather dataset containing daily temperature data from four different regions of Canada (Figure 13) was downloaded on 2014-08-16 from `http://www.psych.mcgill.ca/misc/fda/downloads/FDAfuns/Matlab` (in the file `weather.zip`). A more thorough description of the dataset is available at `http://www.psych.mcgill.ca/misc/fda/ex-weather-a1.html`. For simplicity only the two regions with the largest number of fields are analyzed here:

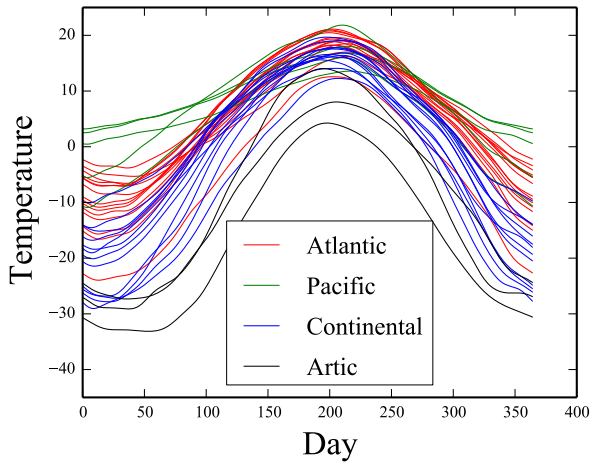

Figure 13: Weather dataset (Ramsay and Silverman 2005). Each field represents the recordings at one of 35 separate weather stations.

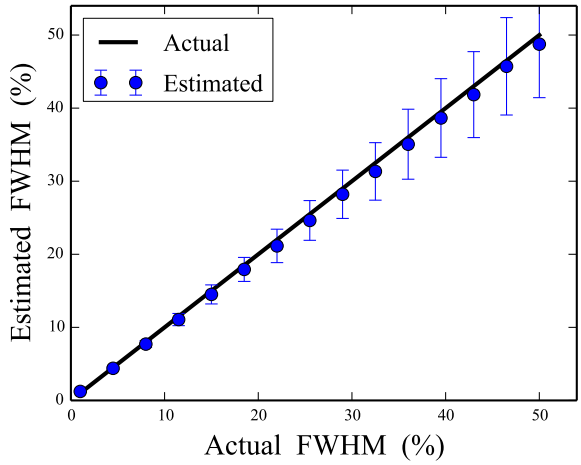

Figure 14: Field smoothness estimations from simulated null data with known FWHM (sample size = 10, $S = 100$). For each FWHM, 500 iterations were performed. Dots and error bars depict the mean and SD of the FWHM estimate across iterations.

the Atlantic and Continental regions, with 15 and 12 fields, respectively.

5.2. Smoothness estimation

Since the field FWHM is not known *a priori* it can be estimated from the model residuals. If the Atlantic and Continental fields are stored in **NumPy** arrays `yA` and `yB`, respectively, the model residuals can be computed by subtracting sample means:

```
> mA, mB = yA.mean(axis = 0), yB.mean(axis = 0)
> rA, rB = yA - mA, yB - mB
> residuals = np.vstack([rA, rB])
```

rft1d employs an unbiased FWHM estimation procedure (Kiebel, Poline, Friston, Holmes, and Worsley 1999) which can be accessed from the `rft1d.geom` module as follows:

```
> FWHM = rft1d.geom.estimate_fwhm(residuals)
```

For this dataset the estimated FWHM is 135.7. These FWHM estimates can be validated using `rft1d.randn1d` to generate residuals of known smoothness, then repeating for many iterations for various FWHMs (Figure 14).

5.3. Classical hypothesis testing

Assuming equal variance for simplicity, the two-sample test statistic field can be computed as follows:

```
> nA, nB = 15, 12
> mA, mB = yA.mean(axis = 0), yB.mean(axis = 0)
> sA, sB = yA.std(ddof = 1, axis = 0), yB.std(ddof = 1, axis = 0)
> s = np.sqrt(((nA - 1) * sA * sA + (nB - 1) * sB * sB) / (nA + nB - 2))
> t = (mA - mB) / (s * np.sqrt(1.0 / nA + 1.0 / nB))
```

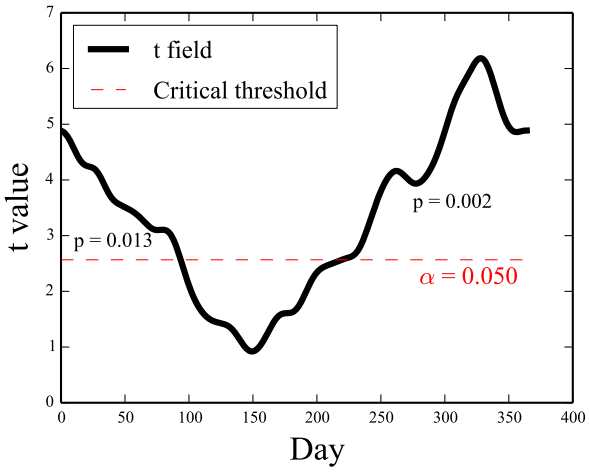

Figure 15: Classical hypothesis testing comparing the Atlantic and Continental regions from the weather dataset (Figure 13).

where `sA` and `sB` are sample standard deviations and `s` is the pooled standard deviation. The critical threshold can then be computed using the `rft1d.distribution` interface:

```
> alpha = 0.05
> df = nA + nB - 2
> nNodes = 365
> t_critical = rft1d.t.isf(alpha, df, nNodes, FWHM)
```

This yields a critical threshold of 2.566, and for comparison the uncorrected (0D) threshold is 1.708. Since the computed test statistic field exceeds the critical threshold (Figure 15), the null hypothesis is rejected at $\alpha = 0.05$.

5.4. Set- and cluster-level inferences

There are two upcrossings with extents of 0.689 and 1.067 resels, respectively. Set- and cluster-level inference can be conducted as follows:

```
> c = 2
> k = [0.689, 1.067]
> tstar = 2.566
> df = 25
> nNodes = 365
> FWHM = 135.7
> Pset = rft1d.t.p_set(c, min(k), tstar, df, nNodes, FWHM)
> Pcluster = [rft1d.t.p_cluster(kk, tstar, df, nNodes, FWHM) for kk in k]
```

The set-level p value is approximately 0.00008, implying that Gaussian fields of the identical smoothness would produce two t-field upcrossings with a minimum extent of 0.689 with a probability of 0.00008. The cluster-level p values are 0.013 and 0.002, respectively (Figure 15), implying that Gaussian fields of the identical smoothness would produce t-field upcrossings of the given extents with probabilities of 0.013 and 0.002, respectively.

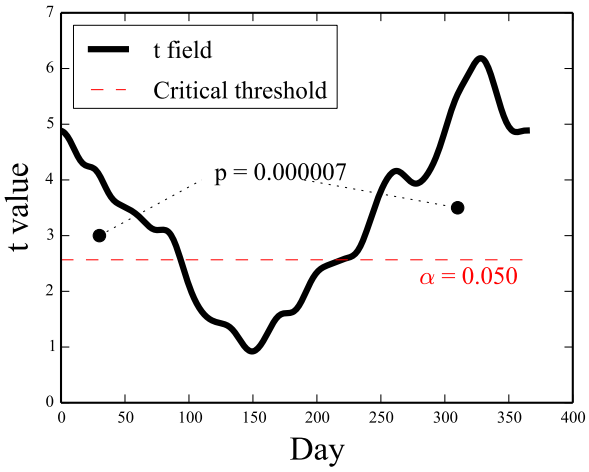

Figure 16: Classical hypothesis testing for the weather dataset, assuming a circular field.

It is important to note that the above analyses ignore the fact that this is a circular field: The last day of the year is followed by the first day of the next year. In this case the upcrossing should be wrapped from the end to the start of the field. This yields just one upcrossing with a much larger extent and thus a much lower p value (Figure 16).

5.5. Non-parametric inference

Classical hypothesis testing using RFT can be regarded as a two-stage procedure involving (i) the critical threshold computation based on the expected test statistic field maximum, followed by (ii) set-level and/or cluster-level inference. Nichols and Holmes (2002) propose a non-parametric permutation procedure which mirrors this two-stage RFT procedure:

1. *(Stage 1)* Randomly permute the labels;
2. Compute the test statistic field based on the new labels;
3. Extract and store the field maximum ($t_{\max}$);
4. Repeat (1)–(3) until all permutations have been exhausted or for a set number of random permutations;
5. Compute the critical test statistic (t^*) from the $t_{\max}$ distribution;
6. *(Stage 2)* Repeat (1)–(4), this time extracting the maximum upcrossing extent ($k_{\max}$) at threshold t^* for each permutation;
7. For cluster-based inference, compute upcrossing-specific p values based on the empirical $k_{\max}$ distribution.

This procedure yields a critical threshold (2.359) that is somewhat lower than the RFT threshold (2.566), but also yields upcrossings p values which are essentially identical (Figure 17). Since the results are qualitatively identical, it may be concluded that RFT's assumption of Gaussian field randomness is a reasonably good one for this dataset.

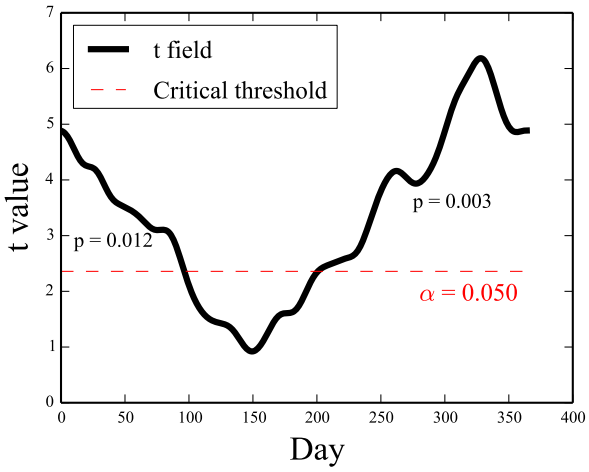

Figure 17: Non-parametric hypothesis testing results following Nichols and Holmes (2002).

6. Summary

This paper has described a Python package called **rft1d** which implements random field theory (RFT) expectations and probabilities regarding upcrossings in smooth 1D Gaussian and test statistic fields. These probabilities may be used to make a variety of statistical inferences regarding experimental fields, including classical univariate and multivariate hypothesis testing. Validations of RFT probabilities for relatively short fields (length = 100) are available in the example scripts in the supplementary material for unbroken and broken fields, and also for test statistic fields in conjunction. It is hoped that this 1D implementation will both help to make RFT procedures more accessible to researchers for 1D data analysis, and help to reduce conceptual barriers associated with RFT in higher-dimensional datasets.

Acknowledgments

This work was supported by Wakate A Grant 15H05360 from the Japan Society for the Promotion of Science.

References

Adler RJ, Hasofer AM (1976). "Level Crossings for Random Fields." *The Annals of Probability*, **4**(1), 1–12. doi:10.1214/aop/1176996176.

Adler RJ, Taylor J (2007). *Random Fields and Geometry*. Springer-Verlag. doi:10.1007/978-0-387-48116-6.

Barnes G, Ridgway G, Flandin G, Woolrich M, Friston K (2013). "Set-Level Threshold-Free Tests on the Intrinsic Volumes of SPMs." *NeuroImage*, **68**, 133–140. doi:10.1016/j.neuroimage.2012.11.046.

Cao J, Worsley K (1999). "The Detection of Local Shape Changes via the Geometry of Hotelling's T^2 Fields." *The Annals of Statistics*, **27**(3), 925–942. doi:10.1214/aos/1018031263.

Carbonell F, Worsley KJ, Galan L (2011). "The Geometry of the Wilks's Λ Random Field." *The Annals of the Institute of Statistical Mathematics*, **63**(1), 1–27. doi:10.1007/s10463-008-0204-2.

Enthought, Inc (2014). "**Canopy**: Scientific and Analytic Python Deployment with Integrated Analysis Environment." URL https://www.enthought.com/products/canopy/.

Friston KJ, Ashburner J, Barnes G, Flandin G, Gitelman D, Glauche V, Hutton C, Litvak V, Moran R, Oostenveld R, Penny W, Phillips C, Pinotsis D, Ridgway G, Seghier M, Stephan KE, Andersson J, Brett M, Buechel C, Chen CC, Chumbley J, Daunizeau J, Harrison L, Heather J, Henson R, Holmes A, Rosa M, Kiebel S, Kilner J, Mattout J, Nichols T, Poline JB, Worsley KJ (2014). "Statistical Parametric Mapping." The Wellcome Trust Centre for Neuroimaging, URL http://www.fil.ion.ucl.ac.uk/spm/.

Friston KJ, Ashburner J, Kiebel S, Nichols T, Penny W (2007). *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. Elsevier, London.

Friston KJ, Worsley KJ, Frackowiak RSJ, Mazziotta JC, Evans AC (1994). "Assessing the Significance of Focal Activations Using Their Spatial Extent." *Human Brain Mapping*, **1**(3), 210–220. doi:10.1002/hbm.460010306.

Hasofer A (1978). "Upcrossings of Random Fields." *Advances in Applied Probability*, **10**, 14–21. doi:10.2307/1427002.

Hunter J (2007). "**Matplotlib**: A 2D Graphics Environment." *Computing in Science & Engineering*, **9**(3), 90–95. doi:10.1109/mcse.2007.55.

Jones E, Oliphant T, Peterson P, *et al.* (2001). "**SciPy**: Open Source Scientific Tools for Python." URL http://www.scipy.org/.

Kiebel SJ, Poline JB, Friston KJ, Holmes AP, Worsley KJ (1999). "Robust Smoothness Estimation in Statistical Parametric Maps Using Standardized Residuals from the General Linear Model." *NeuroImage*, **10**(6), 756–766. doi:10.1006/nimg.1999.0508.

Millman K, Brett M (2007). "Analysis of Functional Magnetic Resonance Imaging in Python." *Computing in Science and Engineering*, **9**(3), 52–55. doi:10.1109/mcse.2007.46.

Nichols TE, Holmes AP (2002). "Nonparametric Permutation Tests for Functional Neuroimaging: A Primer with Examples." *Human Brain Mapping*, **15**(1), 1–25. doi:10.1002/hbm.1058.

Pataky T (2012). "One-Dimensional Statistical Parametric Mapping in Python." *Computer Methods in Biomechanics and Biomedical Engineering*, **15**(3), 295–301. doi:10.1080/10255842.2010.527837.

Ramsay J, Silverman B (2005). *Functional Data Analysis*. Springer-Verlag, New York. doi:10.1007/978-1-4757-7107-7.

The MathWorks, Inc (2014). "MATLAB – The Language of Technical Computing, Version R2014a." The MathWorks, Inc., Natick, Massachusetts, URL http://www.mathworks.com/products/matlab/.

van der Walt S, Colbert S, Varoquaux G (2011). "The **NumPy** Array: A Structure for Efficient Numerical Computation." *Computing in Science and Engineering*, **13**, 22–30. doi:10.1109/mcse.2011.37.

van Rossum G (2014). "The Python Library Reference Release 2.7.8." URL https://docs.python.org/2/library/.

Worsley KJ (1995). "Estimating the Number of Peaks in a Random Field Using the Hadwiger Characteristic of Excursion Sets, with Applications to Medical Images." *The Annals of Statistics*, **23**(2), 640–669. doi:10.1214/aos/1176324540.

Worsley KJ (2006). "**FMRISTAT**: A General Statistical Analysis for fMRI Data." URL http://www.math.mcgill.ca/keith/fmristat/.

Worsley KJ, Marrett S, Neelin P, Vandal A, Friston KJ, Evans AC (1996). "A Unified Statistical Approach for Determining Significant Signals in Images of Cerebral Activation." *Human Brain Mapping*, **4**(1), 58–73. doi:10.1002/(sici)1097-0193(1996)4:1<58::aid-hbm4>3.3.co;2-l.

Worsley KJ, Taylor J, Tomaiuolo F, Lerch J (2004). "Unified Univariate and Multivariate Random Field Theory." *NeuroImage*, **23**, S189–S195. doi:10.1016/j.neuroimage.2004.07.026.

Affiliation:

Todd C. Pataky
Institute for Fiber Engineering
Department of Bioengineering
Shinshu University
Tokida 3-15-1, Ueda, Nagano, Japan
E-mail: tpataky@shinshu-u.ac.jp
URL: http://www.tpataky.net/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2016, Volume 71, Issue 7 *Submitted:* 2014-08-25
doi:10.18637/jss.v071.i07 *Accepted:* 2015-12-17