

June 2002

Barcelona Short Course

Daniel Kaplan, Macalester College

Making and Using Surrogate Data

Objectives

This lab introduces surrogate data: how to generate it; how to use it; and how to interpret it. Since generating surrogate data is so simple, there will be a strong emphasis on the interpretation and on pitfalls and artifacts. But don't be too discouraged by the difficulties; whatever are the problems with the surrogate data technique, not using it is much worse! Surrogate data should be a standard part of your nonlinear time series analysis toolbox.

Tools

Three surrogate-generating programs are provided here:

Phase-randomized surrogates `fftsurr` produces a realization of phase-randomized surrogate data. Each time the program is called, it produces a new realization of surrogate data (controlled by the computer random number generator seed).

```
>> surr1 = fftsurr(data);
>> surr2 = fftsurr(data);
```

Note that `surr1` and `surr2` are different, but their power spectra (as estimated using the amplitude of the fft) are the same as the original time series:

```
>> psd1 = abs(fft(data));
>> psd2 = abs(fft(surr1));
```

Plot out one power spectrum versus the other (`plot(psd1,psd2,',')`) (or `plot(log(psd1), log(psd2),',')`) and you will get a straight line: the two spectra are identical.

Amplitude-adjusted surrogates `ampsurr` produces "amplitude-adjusted surrogates." In phase-randomized surrogates, the power spectrum of the test data and the surrogates are identical. However, the actual values in the signal are not the same. For example, try

```
>> plot( sort(data), sort(surr1), ',' )
```

or

```
>> subplot(2,1,1); hist(data); subplot(2,1,2); hist(surr1);
```

with phase-randomized surrogates.

In amplitude-adjusted surrogates,

```
>> asurr1 = ampsurr(data);
```

the amplitudes of the surrogate and data are identical; the surrogate is just a shuffled version of the points in the data. The shuffling has been cleverly done, however, so that the power spectrum of the signal and the surrogate are *almost* the same. Try comparing the power spectra of an amplitude-adjusted surrogate.

Polished surrogates (`polsurr`) produces surrogates that, like amplitude-adjusted surrogates, have the same amplitude distribution as the original data. But the shuffling has been more carefully done than in amplitude-adjusted data so the power spectrum of the surrogate is even closer to that of the original data.

There is also a program for shuffled surrogates, `surr = shuffledsurr(data);`. This is just like amplitude-adjusted surrogates except the power spectrum is white. The only time you would want to use such surrogates is when all you care about is the amplitude of the data, for example when you want to use shuffled residuals to drive a linear model. *Do not use shuffled surrogates for general purposes.*

To use surrogate data as part of a hypothesis test, you need a test statistic. Some examples of test statistics that are widely used are the correlation dimension and nonlinear predictability. The basic idea is to calculate the test statistic for the original data (called the “test data”) and then calculate the test statistic for a bunch of realizations of surrogate data. You compare the single value for the test data to the bunch of values for the surrogates, perhaps ultimately computing a p-value.

Two important points to keep in mind when considering a test statistic are:

1. It should generally be a single number: a scalar, not a vector. There are occasions when a vector test statistic makes sense, but they are more complicated than the ones we will consider here.
2. Exactly the same parameters and settings should be used for computing the test statistic for the test data as for the surrogate data. For example, it is *illegitimate* to pick some parameters for the test data (for instance, the scaling region for the correlation dimension calculation) and then apply those parameters to the surrogates.

If the above two restrictions have been honored, then it should be possible to implement the test statistic as a computer function that takes exactly one argument — a time series — and returns a single number.

Here are some test statistics that are implemented as programs: you can construct your own with the basic template

```
function result = myteststat(data)
result = complicatedcalculation(data, parameter1,parameter2, and so on);
```

This code would go in a file named `myteststat.m`. The complicated calculation might be nonlinear prediction, or whatever; you can put in whatever program you like to do the actual calculations. It is a matter of personal intellectual integrity that the parameters haven’t been established by looking at the data you are testing.

Here are two test statistics that are somewhat abstract, but have the great virtue of being extremely fast to calculate:

timerev(data,timescale) computes a statistic relating to the time-reversal asymmetry of the data. The parameter `timescale` is an integer describing a time scale of interest; set it to be perhaps one-quarter of the approximate period of oscillations. In order to avoid setting the parameter by looking at the data, you might want to generate a surrogate and base your estimate of the period from the surrogate. Then throw away the surrogate.

crinkle(data) computes a 4th moment of the time series. James Theiler invented this statistic to demonstrate some weaknesses of surrogate data.

To make it easy to do a hypothesis test, we provide a function `surrogatetest` that takes the data and a function implementing a test statistic, and runs many surrogates. `surrogatetest` returns the t-score and a non-parametric p-value (for a left-sided test, so subtract the result from 1 for a right-sided test). It also gives back the test statistic for the test data and the surrogates. An example (using the Baltimore measles data in `balt.dat`):

```
>> [t,r] = surrogatetest(balt, 'crinkle(x)', 'fftsurr') => ans:  t = 801
                                     r = 0.9500
```

The result is a strong indication of nonlinearity: the t-statistic is huge (a value of 1.73 would be statistically significant at the 95% level for a one-tailed test); the non-parametric p-value is 0.05 for the right-tailed test.

By default, 19 surrogates are used. (The number of surrogates is set by the 4th argument, if it is given.) If we ran the test again with more surrogates, the non-parametric p-value would be even less — it’s 1% for 99 surrogates. Wow! When have you ever seen a result that strong with so little effort!

An important note about programming. Note that in the above, the test statistic was `crinkle(x)`. The variable should always be `x` regardless of the name of the data set. `x` is just a dummy variable. You can also include parameters, but they must be numbers and not variables. For example: `quickde(x,2,3)` but NOT `quickde(x,dim,lag)`.

Questions

- Re-run the analysis of the Baltimore data with amplitude-adjusted surrogates and with polished surrogates. How and why do the results differ?
- Write a program for either nonlinear prediction or for the correlation dimension and apply it to the lorenz data. Remember, your program should take a data set as an input and provide a scalar number as an output. For the dimension calculation, you might want to use the rule-of-five method. For nonlinear prediction, perhaps take the median absolute value of the residual.
- Take a segment of Lorenz or Rossler data and try nonlinear prediction using both short and long prediction horizons. Now try the prediction on some surrogates generated from the data. How do the results differ for different prediction horizons?
- Let's make some data from a linear process with gaussian white noise inputs:
`>> y = randn(500,1);`
`>> x = arma([1.95 -.96], 1, y);`
 Use the ARMA parameters given. You'll see that the result is a smoothly varying oscillation.

This represents a signal for which surrogate data should give a negative result: the data satisfy the null hypothesis of surrogate data.

Test the linear data using surrogate data. Since this is a hypothesis test, you should get a result that causes you to reject the null hypothesis every once in a while; for a significance level of 5% you should anticipate a false rejection of the null 5% of the time. Repeat the test many times, using new realization of the noisy input `y` to generate new linear time series `x`. How often do you falsely reject the null?

- Try the same thing, but using `x.^3` or `x.^2` instead of `x`.
- Let's try a slightly different linear system:
`>> x = arma([1.5 -.7], 1, y);`
 This system has a fast-decaying impulse response.
 Rather than using gaussian white noise as the input, lets use some non-gaussian spikes:
`>> y = zeros(500,1); y(10) = 7; y(287) = -3; y(403) = -5;`
`>> x = arma([1.5 -.7], 1, y);`
 Test this system for "nonlinearity" using the crinkle test statistic. What violates the null hypothesis in this case? Is there a difference in the result for amplitude adjusted and for polished surrogates?
- Generate two ARMA signals with quite different power spectra (such as the two used above) and concatenate them to make a linear, nonstationary system. Test the nonstationary signal using surrogate data.