Week 3: Python Worksheet 3, Numbers

Barry Rowlingson

November 28, 2021

1 Intro

For this workshop do everything in a single Jupyter Lab notebook. This gives a development environment that we know is complete and we don't have to worry about reloading modules all the time.

2 Raised Incidence Modelling

Are there more lung cases near this incinerator? Questions like that form some of the earliest applications of spatial statistics to health data. As an example, Peter J. Diggle and Barry S. Rowlingson (1994), A Conditional Approach to Point Process Modelling of Elevated Risk. Journal of the Royal Statistical Society. Series A (Statistics in Society), 157(3), 433–440. doi:10.2307/2983529

describes a case-control point process model for elevated risk around a specified point source.

The data are a set of case locations corresponding to the disease in question and a set of control locations. The controls should be a random sample from the population at risk. This could be a random sample of the whole population, or, in the example above, cases of a disease that has a similar age profile as the disease of interest but with no suspected strong association with the source. In the Chorley-Ribble data, the cases of interest are larynx cancer, the controls are lung cancer, and the possible source is an incinerator.

To do a statistical analysis we need a statistical model. If we consider cases as "ones" and controls as "zeroes" then we can fit a Bernoulli model based on distance to the source. For this class of model we need a function that returns a probability (between 0 and 1) given some set of model parameters at any given distance.

A good statistical model has as few parameters as necessary to fit the requirements. In this case we need at least three to describe three features of raised incidence - the level of raised incidence at the source, the background level of incidence at large distance from the source, and a parameter for how quickly the effect of the source (if any) drops off with distance. The paper suggests the following model for the probability of a datapoint being a case at a distance d, and there are three model parameters:

$$p_1(d;\alpha,\beta,\rho) = \rho f/(1+\rho f)$$

where

$$f(d,\alpha,\beta) = 1+\alpha\exp(-(d/\beta)^2)$$

This model has the features we need - at large distance the exponential term goes to zero and the probability is a constant $\rho/(1+\rho)$. At zero distance the $f()$ expression becomes $1+\alpha$ and the probability becomes $\rho(1+\alpha)/(1+\rho(1+\alpha))$. The probability decreases with distance at a rate governed by the β parameter.

Other model formulations for $p_1(.)$ are possible, but often in statistics we don't have enough data to really justify a more complex model, so we start simple and see where that gets us.

3 Probability Decay Function

Make sure you `import numpy as np` and `import matplotlib.pyplot as plt` at the start of your notebook.

Next write the `f(.)` function. Start like this, and fill in the return value as described by the formula above. Assume the distance, `d`, is passed in as a `numpy` array so you can use `np.exp` for the exponential. The `alpha` and `beta` parameters will be scalars.

```
def f(d, alpha, beta):
   return ...
```

Check that the following calls to your `f()` function produce the same results as this:

```
f(0, alpha=1, beta=1) = 2.0000 (zero distance)
f(1, alpha=1, beta=1) = 1.3679 (unit distance)
f(1000, alpha=1, beta=1) = 1.0000 (long distance)
```

Next write a function called `p` that works out the case probability using the formula. It will call your `f` function and then work out `p`. It will start like this:

```
def p(d, alpha, beta, rho):
  fd = f(d, alpha, beta)
  ...
```

Assume the `d` distances are a `numpy` array so you don't need to loop over the values in any computation.

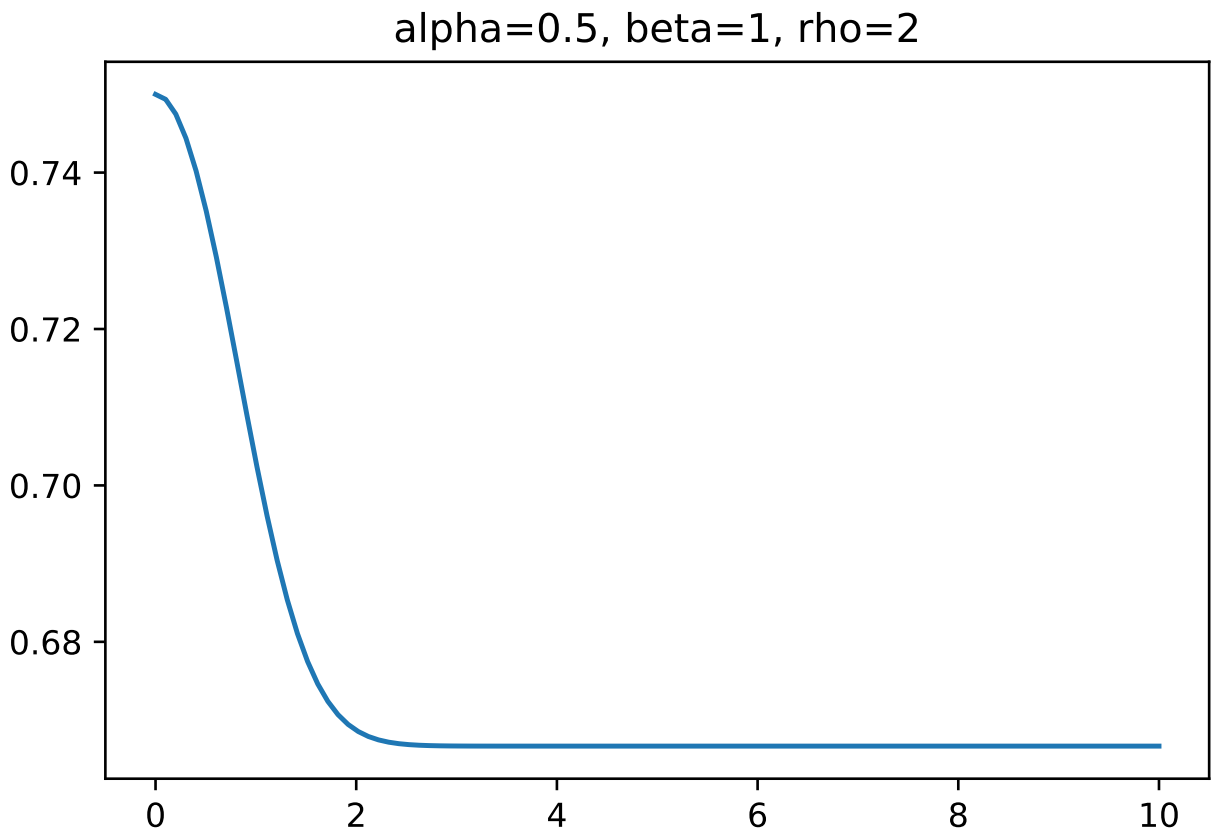
Check that your `p` function returns the same values as these, and confirm that the probabilities in the zero- and long-distance cases correspond to the expressions for those limits in the description of the model above.

```
p(0, alpha=0.5, beta=1, rho=2) = 0.7500 (zero distance)
p(1, alpha=0.5, beta=1, rho=2) = 0.7031 (unit distance)
p(1000, alpha=0.5, beta=1, rho=2) = 0.6667 (long distance)
```

4 Graphs

Let's look at how this probability varies with distance by plotting it. First we'll create a one-dimensional array of 100 distances from 0 to 10, and then compute and plot the probability.

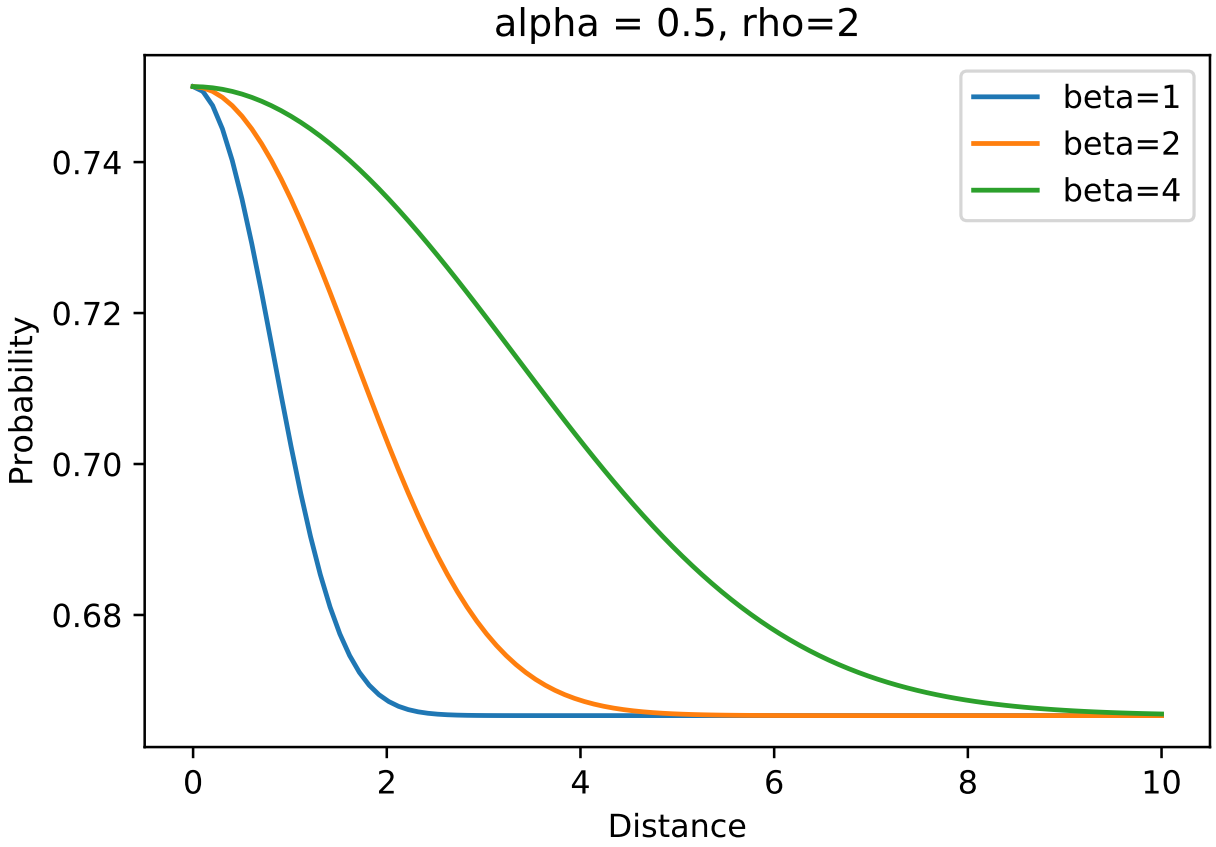
```
d = np.linspace(0, 10, 100)
pv = p(d, alpha=0.5, beta=1, rho=2)
fig = plt.plot(d, pv)
plot_title = plt.title("alpha=0.5, beta=1, rho=2")
```



4.1 Multiple Graphs

Next let's see how the function changes with varying β values. This controls the distance-decay aspect of the model. We'll create a plot with three lines on it for β values in $\{1, 2, 4\}$.

```
d = np.linspace(0, 10, 100)
pv1 = p(d, alpha=0.5, beta=1, rho=2)
fig1 = plt.plot(d, pv1, label="beta=1")
pv2 = p(d, alpha=0.5, beta=2, rho=2)
fig2 = plt.plot(d, pv2, label="beta=2")
pv3 = p(d, alpha=0.5, beta=4, rho=2)
fig3 = plt.plot(d, pv3, label="beta=4")
plt.xlabel("Distance")
plt.ylabel("Probability")
title = plt.title("alpha = 0.5, rho=2")
legend = plt.legend()
```



There's a lot of repeated code there. Change it so that you can plot for any sequence of `beta` values. Your code should look like this:

```
alpha = 0.5
betas = [1, 2, 4]
rho = 2
for beta in betas:
    pv = p(d, alpha, beta, rho)
    # plt.plot(d, pv, label=...)
# plt.xlabel(...)
# plt.ylabel(...)
# plt.title(...)
# legend = plt.legend()
```

4.2 A note on "f" strings

Python's formatted strings are extremely useful when you want to print the value of a variable or an expression. They will help when making labels and titles for graphs where you want the text to reflect the value of a variable. For example:

```
alpha = 99
beta = -12.67263
gamma = 0.00023
# put the value in place where the { } are:
print(f"The value of alpha is {alpha}")
# use an equals sign to label the value too:
print(f"{alpha=} and {beta = }")
# use an expression and round to two decimals:
print(f"{gamma*1000 = :.2f}")
```

```
The value of alpha is 99
alpha=99 and beta = -12.67263
gamma*1000 = 0.23
```

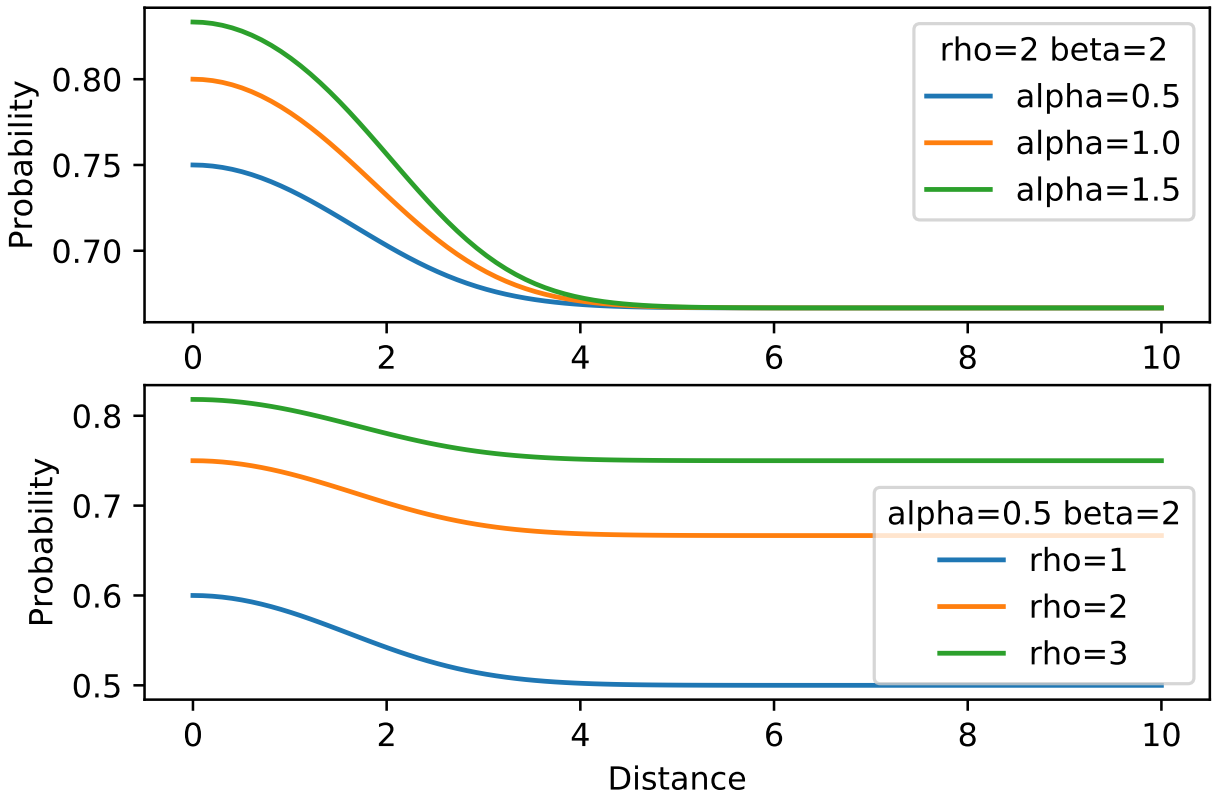
4.3 More Graphs

Now show the effect of setting α to the values $\{0.5, 1, 1.5\}$ while keeping $\rho = 2$ and $\beta = 2$. This can be done with a variation on the loop above. Then show the effect of changing ρ while keeping $\alpha = 0.5$ and $\beta = 2$ in another graph.

You can put both these plots in the same figure using the `subplots` feature:

```
fig, ax = plt.subplots(2,1)
```

and then instead of plotting using `plt.plot(...)` you can plot using `ax[0].plot(...)` and `ax[1].plot(...)`. Try adding the title to the legend as well.

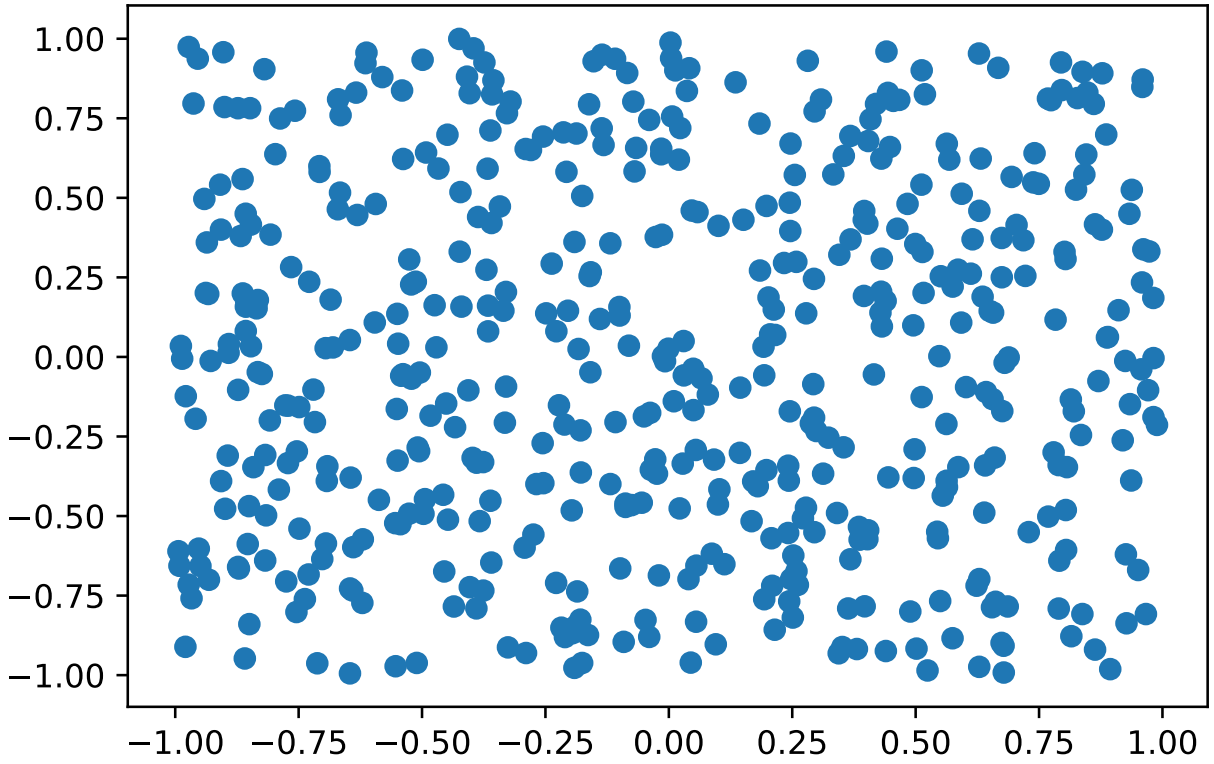


5 Data Generation

Knowing the probability at any distance gives us a method for generating synthetic data from the model for a given set of parameters. We'll use the random number generators from `numpy` to do this. All the code for this section is given here but you could also attempt to improve it my annotating plot labels, titles, legends, etc.

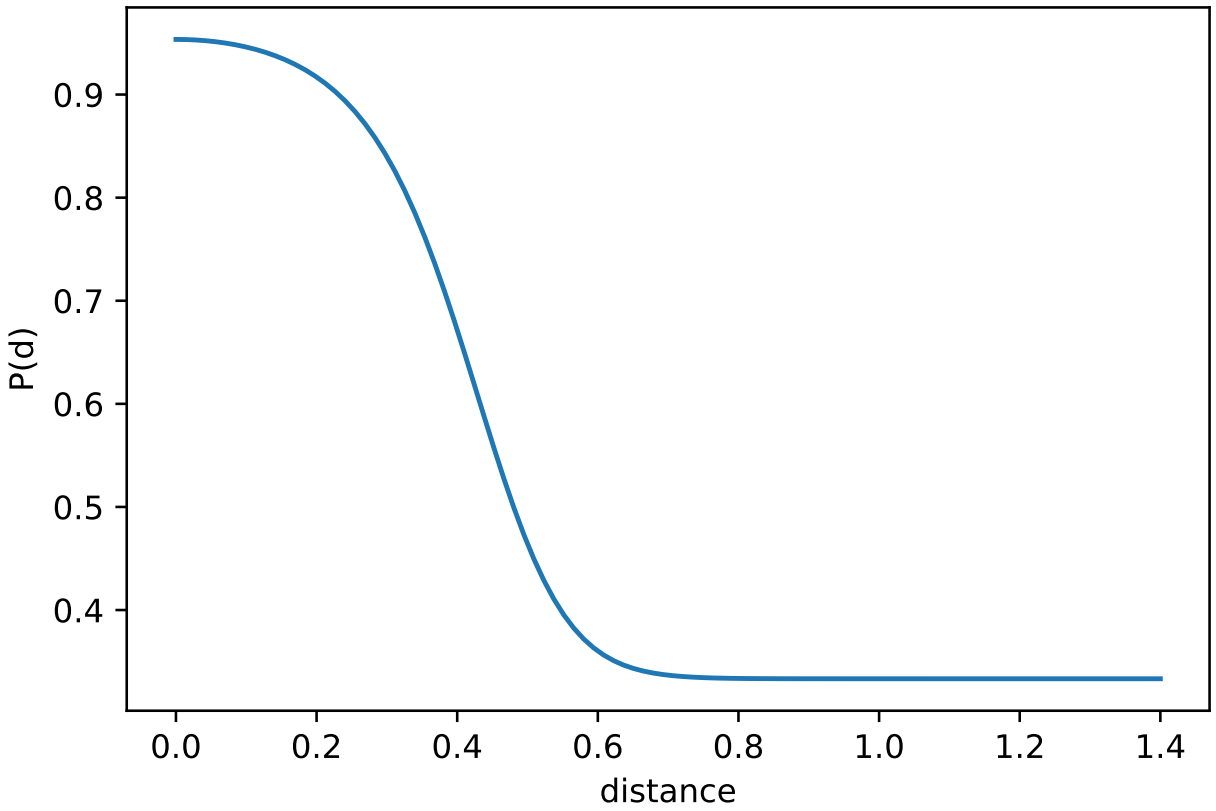
First we'll generate some points in a square bounded by -1,1 on both axes. The `numpy` random number generator system works by you first creating an instance of `default_rng()` and then calling methods from that. In the following code we create the X, Y coordinates of points and plot them.

```
from numpy.random import default_rng
npts = 500
rng = default_rng()
x,y = (rng.uniform(-1, 1, npts), rng.uniform(-1, 1, npts))
xy = plt.scatter(x,y)
```



To test our data generation is working we should try it with a probability curve that makes it clear if we are generating raised incidence near the source or not. Here's a plot with some extreme parameter values at the limits.

```
d = np.linspace(0, 1.4, 100)
alpha = 40
beta = 0.25
rho = .5
probs = p(d, alpha=alpha, beta=beta, rho=rho)
fig = plt.plot(d, probs)
xl = plt.xlabel("distance")
yl = plt.ylabel("P(d)")
```



Next we compute the distance from the origin (0,0) to each point and get the corresponding case probability from the function `p` that we wrote earlier.

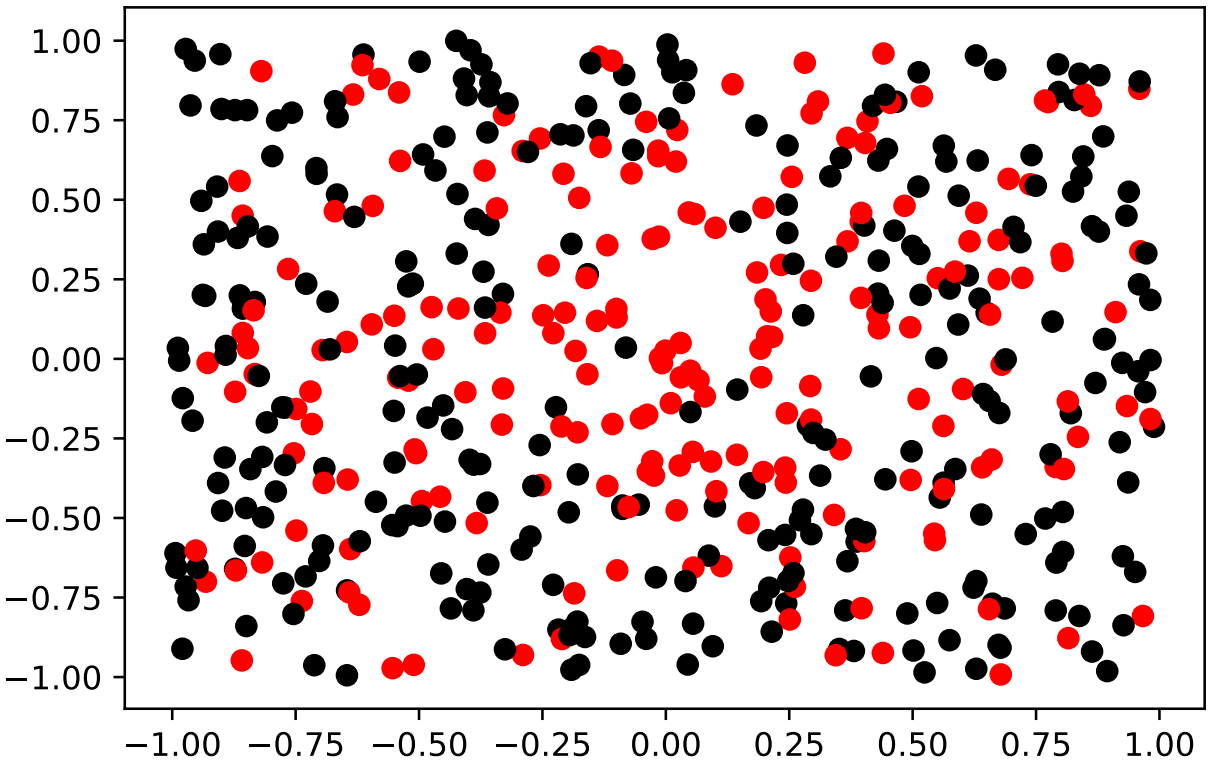
```
dist = np.sqrt(x**2 + y**2)
pdist = p(dist, alpha=alpha, beta=beta, rho=rho)
```

Now we want to toss a *weighted* coin for each of these points, where the weight is the probability just computed. If the weighted coin comes up heads, we call this a case, otherwise its a control. This is a binomial random variate of size 1, and we can compute that with `rng.binomial`:

```
cc = rng.binomial(1, pdist)
```

Finally we can plot a simple map and colour the points black for zero (control) and red for ones (cases):

```
xyplot = plt.scatter(x,y)
cols = np.array(["black","red"])
xyplot.set_color(cols[cc])
```



Does this look like there's a greater number of red points near the centre? You could investigate this by dividing the points into distance bands and computing the ratio of cases to controls in each band. Or you can do as in the paper, and find the parameters that maximise the likelihood for this data. But that is for another time...

6 Conclusion

You should now have a notebook that you can run from the start with no errors and that generates data and plots for this raised incidence model.

Think about what might be worth taking out of this notebook into a standalone Python module and how you would interface to that from this notebook.