Import NumPy and SciPy (not needed when using --pylab)

```
In [ ]:
```

Load data from file

```
In [ ]:
```

```
In [4]: np.min() # Check bounds
```

```
In [5]: np.max()
```

Construct histogram from data

There are several histrogram commands: hist() will be fine here, but note the syntax below. Also note that the bin *edges* are returned, so that there will nbins+1 of these.

```
In [7]: nbins = 50; # Is this a good choice?
```

```
In [8]: n, bins, patches = hist() # With hist, one needs to (spuriously) request the patch objects as well
```

```
In [26]: x = bins[0:nbins] + (bins[2]-bins[1])/2; # Convert bin edges to centres, chopping the last
```

Interpolate histogram output -> p(z); n.b. that you can also use numerical quadrature to get $P(z)$ directly.

```
In [10]: # Import the function you need
```

```
In [11]: # Build an interpolation function for p(z) that accepts an arbitrary redshift z
```

```
In [13]: z = linspace(0,2,100); plot(z,p(z)) # Test your interpolation function out
```

Use numerical integration to get $P(z) = \int_0^\infty p(z')dz'$

```
In [14]: # Import the function you need
```

[redacted]

In [15]:
```
Pz = lambda : ... # Use integrate inside a lambda function to define P(z)?
```

In [16]:
```
total = Pz(5) # Get normalisation constant by evaluating P(z->\infty)
```

In [17]:
```
total # Check that this worked
```

Now, to test your integration you can build a vector that samples P(z).

In [18]:
```
Pz = 0 * z;
for i in range(len(z)):
    Pz[i] = Pz(z[i]) / total
```

In [19]:
```
plot(z,Pz,Pz,z) # Check plotting of P(z) and its inverse
```

Use interpolation again to define $P^{-1}(z)$ at arbitrary $z \in [0,1)$

In [20]:
```
```

Finally, generate uniform random variates and feed them to P^{-1}

In [22]:
```
```

In [24]:
```
hist(output,100); # Make a histogram of your output; does it look like the original one?
```