1 Using NumPy to generate random numbers, or shuffle arrays

NumPy has an extensive list of methods to generate random arrays and single numbers, or to randomly shuffle arrays.

```
import numpy as np

# Optionally you may set a random seed to make sequence of random numbers
# repeatable between runs (or use a loop to run models with a repeatable
# sequence of random numbers in each loop, for example to generate replicate
# runs of a model with different random seeds)

# If seed is not used then a seed will be set using the clock, and so each
# run will have a different seed

np.random.seed(0)
```

1.1 Some examples of random numbers

NumPy has an extensive collection of different types of random distribution.

1.1.1 An array of numbers between 0 and 1

```
# 3x4 array of random numbers between 0 and 1

print (np.random.rand(3,4))

OUT:

[[0.5488135  0.71518937 0.60276338 0.54488318]
 [0.4236548  0.64589411 0.43758721 0.891773  ]
 [0.96366276 0.38344152 0.79172504 0.52889492]]
```

For all methods if the array shape is left out then a single number is returned:

```
print (np.random.rand())

OUT:

0.5680445610939323
```

1.1.2 An array of integers between 10 and 20

Note that the *randint* method uses the 'half open' format where the range includes the lower number, but not the higher number. There is an alternative method *random.random_integers* where the range includes the higher number.

```
print (np.random.randint(10,21,(3,4)))

OUT:

[[15 19 18 19]
 [14 13 10 13]
 [15 10 12 13]]
```

1.1.3 An array of normally distributed numbers

```
# 3x4 array of normally distributed data with mean 10, and SD 1

print (np.random.normal(10,1,(3,4)))

OUT:

[[ 9.81841743 11.41020463  9.62552831 10.27519832]
 [ 9.03924539 10.37692697 10.03343893 10.68056724]
 [ 8.43650331  9.43330238  9.75785049 11.51439128]]
```

1.1.4 List of distributions

From `https://docs.scipy.org/doc/numpy/reference/routines.random.html`

Simple:

rand(d0, d1, , dn) Random values in a given shape.

randn(d0, d1, , dn) Return a sample (or samples) from the standard normal distribution.

randint(low[, high, size, dtype]) Return random integers from low (inclusive) to high (exclusive).

random_integers(low[, high, size]) Random integers of type np.int between low and high, inclusive.

random_sample([size]) Return random floats in the half-open interval [0.0, 1.0).

random([size]) Return random floats in the half-open interval [0.0, 1.0).

ranf([size]) Return random floats in the half-open interval [0.0, 1.0).

sample([size]) Return random floats in the half-open interval [0.0, 1.0).

choice(a[, size, replace, p]) Generates a random sample from a given 1-D array

bytes(length) Return random bytes.

More complex:

beta(a, b[, size]) Draw samples from a Beta distribution.

binomial(n, p[, size]) Draw samples from a binomial distribution.

chisquare(df[, size]) Draw samples from a chi-square distribution.

dirichlet(alpha[, size]) Draw samples from the Dirichlet distribution.

exponential([scale, size]) Draw samples from an exponential distribution.

f(dfnum, dfden[, size]) Draw samples from an F distribution.

gamma(shape[, scale, size]) Draw samples from a Gamma distribution.

geometric(p[, size]) Draw samples from the geometric distribution.

gumbel([loc, scale, size]) Draw samples from a Gumbel distribution.

hypergeometric(ngood, nbad, nsample[, size]) Draw samples from a Hypergeometric distribution.

laplace([loc, scale, size]) Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).

logistic([loc, scale, size]) Draw samples from a logistic distribution.

lognormal([mean, sigma, size]) Draw samples from a log-normal distribution.

logseries(p[, size]) Draw samples from a logarithmic series distribution.

multinomial(n, pvals[, size]) Draw samples from a multinomial distribution.

multivariate_normal(mean, cov[, size,) Draw random samples from a multivariate normal distribution.

negative_binomial(n, p[, size]) Draw samples from a negative binomial distribution.

noncentral_chisquare(df, nonc[, size]) Draw samples from a noncentral chi-square distribution.

noncentral_f(dfnum, dfden, nonc[, size]) Draw samples from the noncentral F distribution.

normal([loc, scale, size]) Draw random samples from a normal (Gaussian) distribution.

pareto(a[, size]) Draw samples from a Pareto II or Lomax distribution with specified shape.

poisson([lam, size]) Draw samples from a Poisson distribution.

power(a[, size]) Draws samples in [0, 1] from a power distribution with positive exponent a - 1.

rayleigh([scale, size]) Draw samples from a Rayleigh distribution.

standard_cauchy([size]) Draw samples from a standard Cauchy distribution with mode = 0.

standard_exponential([size]) Draw samples from the standard exponential distribution.

standard_gamma(shape[, size]) Draw samples from a standard Gamma distribution.

standard_normal([size]) Draw samples from a standard Normal distribution (mean=0, stdev=1).

standard_t(df[, size]) Draw samples from a standard Students t distribution with df degrees of freedom.

triangular(left, mode, right[, size]) Draw samples from the triangular distribution over the interval [left, right].

uniform([low, high, size]) Draw samples from a uniform distribution.

vonmises(mu, kappa[, size]) Draw samples from a von Mises distribution.

wald(mean, scale[, size]) Draw samples from a Wald, or inverse Gaussian, distribution.

weibull(a[, size]) Draw samples from a Weibull distribution.

zipf(a[, size]) Draw samples from a Zipf distribution.

1.2 Shuffle arrays

NumPy can randomly shuffle arrays.

```
x = [0, 1, 2, 3, 4, 5, 6, 7 ,8 ,9, 10]
print ('Original Array:')
print (x)
np.random.shuffle(x)
print ('\nShuffled array:')
print (x)

OUT:

Original Array:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Shuffled array:
[4, 10, 5, 0, 9, 2, 8, 6, 3, 7, 1]
```

1.2.1 Shuffling rows or columns

What if you just wanted to shuffle the rows or columns?

In that case we would create an array of index numbers for the rows or columns, shuffle that, and then use that to reorder the rows.

```
x = np.arange(1,21)
x = x.reshape (5,4)
print ('Original array:')
print (x)

# Shuffle rows
number_of_rows = x.shape[0]
index = list(np.arange(number_of_rows))
np.random.shuffle(index)
print('\nShuffled row index:')
print (index)
# Apply index
y = x[index,:]
print('\nArray with shuffled rows:')
print (y)

# Shuffle columns
number_of_cols = x.shape[1]
index = list(np.arange(number_of_cols))
np.random.shuffle(index)
print('\nShuffled column index:')
print (index)
# Apply index
z = x[:, index]
print('\nArray with shuffled columns:')
print (z)

OUT:

Original array:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]

Shuffled row index:
[3, 0, 2, 1, 4]

Array with shuffled rows:
[[13 14 15 16]
 [ 1  2  3  4]
 [ 9 10 11 12]
 [ 5  6  7  8]
 [17 18 19 20]]

Shuffled column index:
[2, 1, 0, 3]

Array with shuffled columns:
[[ 3  2  1  4]
 [ 7  6  5  8]
 [11 10  9 12]
 [15 14 13 16]
 [19 18 17 20]]
```