6A

Machine Learning II

ID: 5684926 Tristan Scheidemann

6A-1

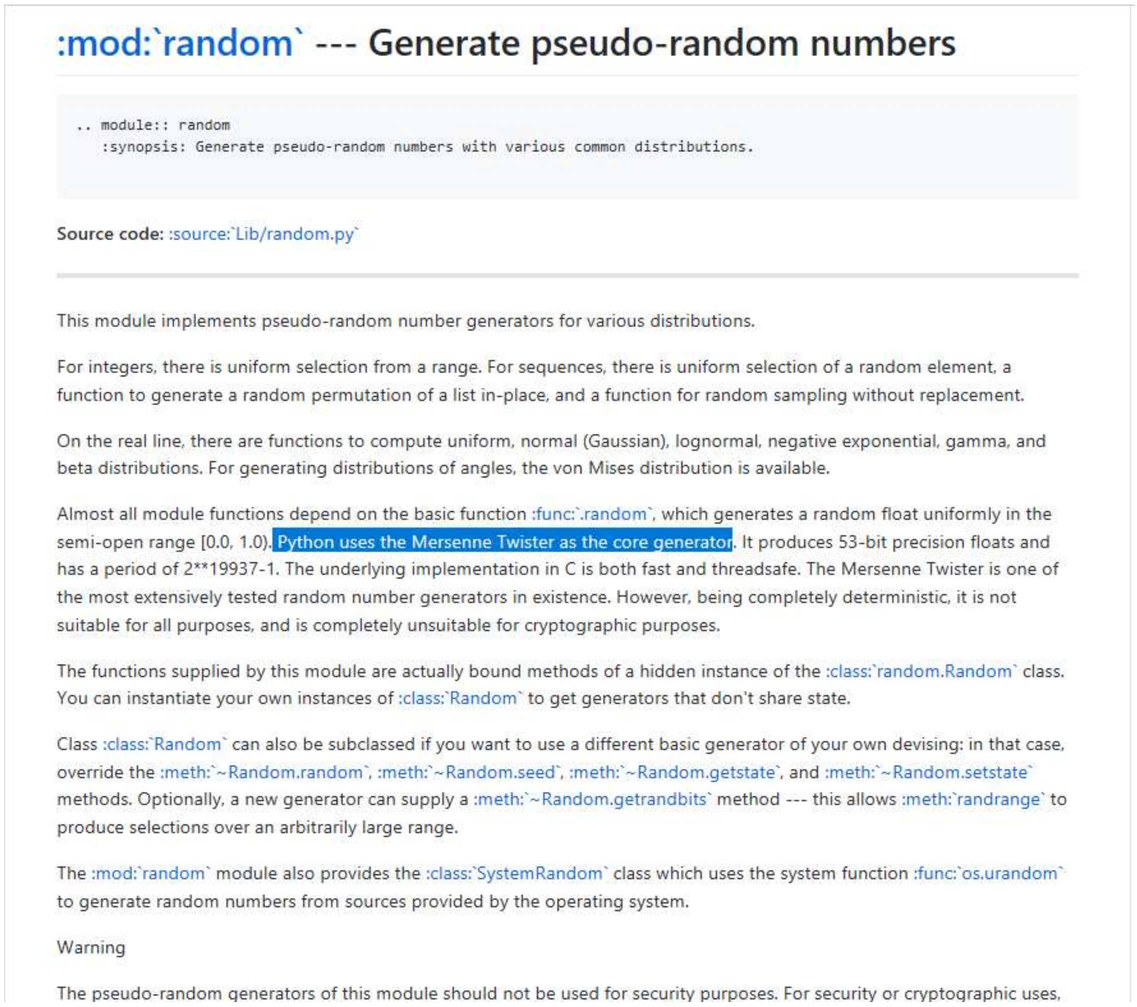
Note: $f(x_i)$ are independent, identically distributed random quantities.

$$\begin{aligned} E\left[\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right] - \mu &= \frac{1}{N}\sum_{i=1}^{N} E[f(x_i)] - \mu \\ &= \frac{1}{N}[N\mu] - \mu \\ &= 0. \end{aligned}$$

6A-2

Language: CPython 3.8.0 alpha 0

Uniform random number generation	
Link	Documentation Implementation



:mod:`random` --- Generate pseudo-random numbers

```
.. module:: random
   :synopsis: Generate pseudo-random numbers with various common distributions.
```

Source code: :source:`Lib/random.py`

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function :func:`.random`, which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

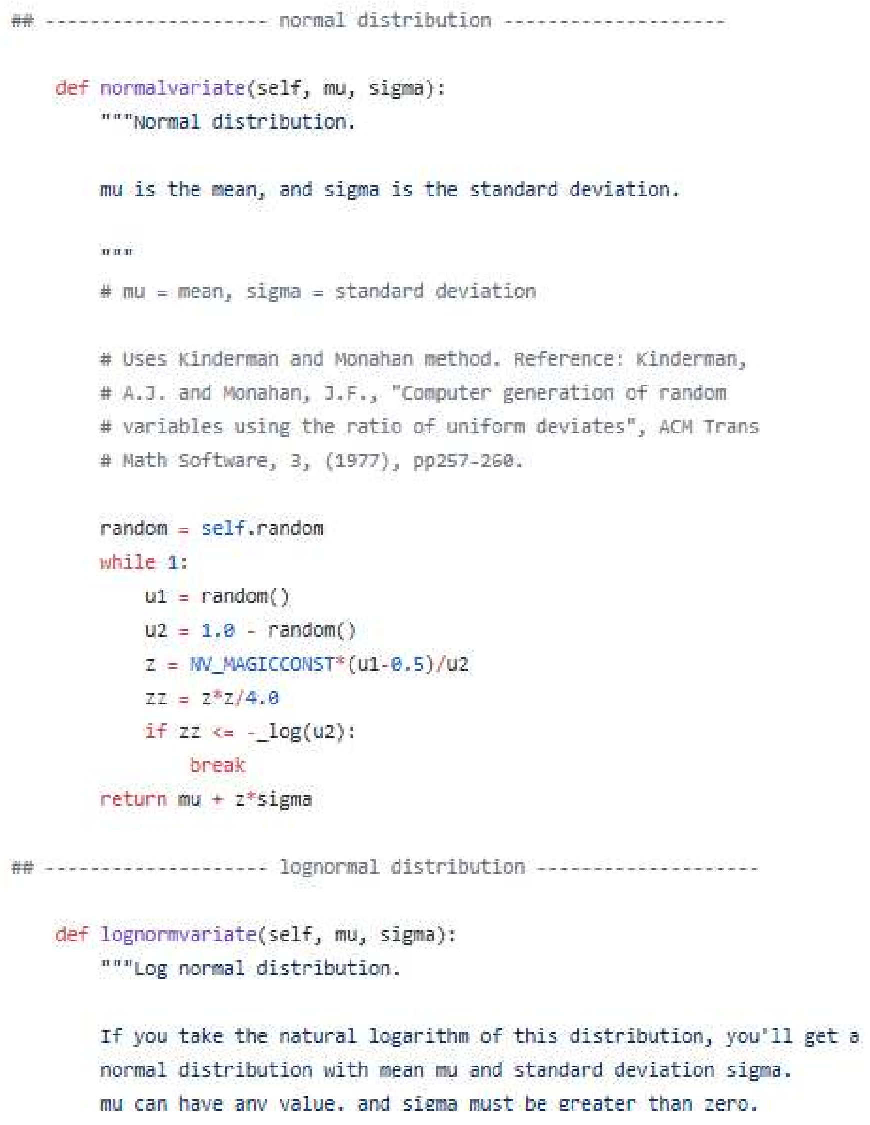
The functions supplied by this module are actually bound methods of a hidden instance of the :class:`random.Random` class. You can instantiate your own instances of :class:`Random` to get generators that don't share state.

Class :class:`Random` can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the :meth:`~Random.random`, :meth:`~Random.seed`, :meth:`~Random.getstate`, and :meth:`~Random.setstate` methods. Optionally, a new generator can supply a :meth:`~Random.getrandbits` method --- this allows :meth:`randrange` to produce selections over an arbitrarily large range.

The :mod:`random` module also provides the :class:`SystemRandom` class which uses the system function :func:`os.urandom` to generate random numbers from sources provided by the operating system.

Warning

The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses,



```
## -------------------- normal distribution --------------------

    def normalvariate(self, mu, sigma):
        """Normal distribution.

        mu is the mean, and sigma is the standard deviation.

        """
        # mu = mean, sigma = standard deviation

        # Uses Kinderman and Monahan method. Reference: Kinderman,
        # A.J. and Monahan, J.F., "Computer generation of random
        # variables using the ratio of uniform deviates", ACM Trans
        # Math Software, 3, (1977), pp257-260.

        random = self.random
        while 1:
            u1 = random()
            u2 = 1.0 - random()
            z = NV_MAGICCONST*(u1-0.5)/u2
            zz = z*z/4.0
            if zz <= -_log(u2):
                break
        return mu + z*sigma

## -------------------- lognormal distribution --------------------

    def lognormvariate(self, mu, sigma):
        """Log normal distribution.

        If you take the natural logarithm of this distribution, you'll get a
        normal distribution with mean mu and standard deviation sigma.
        mu can have any value, and sigma must be greater than zero.
```

Auxiliary calculation

We explain parts of (A. J. KINDERMAN, 1977).

Because $T: (u, v) \to \left(\frac{v}{u}, u\right)$, the pdf of

$$C_f = \left\{(u, v): 0 \le u \le f\left(\frac{v}{u}\right)^{\frac{1}{2}}\right\}$$

scales accordingly:

$$
\begin{aligned}
f_T\left(\frac{v}{u}, u\right) &= f_U(T^{-1}) \cdot |\det T^{-1}| \\
&= f_U(T^{-1}) \cdot \left| \underbrace{-y}_{pg.258,} \right| \\
&= y\, f_U(T^{-1}). \\
&= cy, \qquad c \in \mathbb{R}.
\end{aligned}
$$

To have the marginal in equation (2.2) of (A. J. KINDERMAN, 1977) sum to one, c must be equal to 2, which means C_f always has an area of $1/2$.

The (A. J. KINDERMAN, 1977) algorithm is an acceptance/rejection method, because sampling in C_f is intractable.

6A-3

Auxiliary calculation

The joint distribution of N independent standard normal $\mathbf{z} = (z_1, \dots, z_N)$ is

$$f_Z(\mathbf{z}) = (2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}\mathbf{z}^T I \mathbf{z}}.$$

Furthermore, given $g\colon \mathbb{R}^n \to \mathbb{R}^n$ with $Y = g(Z)$ consider the cube

$$C := I_{z_1, z_1+\delta} \times \dots \times I_{z_n, z_n+\delta}.$$

For small δ, our target

$$g(Z) \approx g(\mathbf{z}) + M(\mathbf{z})[Z - \mathbf{z}]$$

behaves linearly in C, where $M(\mathbf{z})$ is g's Jacobian evaluated at $\mathbf{z}$.
In this neighborhood $g(C)$ describes a parallelepiped with Volume $|\det M(\mathbf{z})| \cdot \delta^n$.

Likewise, assuming $f_Z(\mathbf{z})$ is continuous at $\mathbf{z}$, we have

$$
\begin{aligned}
P(Z \in C) &= \int_C f_Z(\mathbf{z})\, dz \\
&= \int_{g(C)} f_Z\big(\mathrm{g}^{-1}(\mathbf{y})\big)\, dy \\
&\approx f_Z\big(\mathrm{g}^{-1}(\mathbf{y})\big)\delta^n.
\end{aligned}
$$

This leads to:

$$\begin{aligned} f_Z(\mathrm{g}^{-1}(\mathbf{y}))\delta^n &\approx P(Z \in C) \\ &= P\left(\underbrace{g(Z) \in g(C)}_{image\ to\ parallelepiped}\right) \\ &\approx f_Y(y) \cdot vol(g(c)) \\ &\approx f_Y(y)\ |\det M(\mathbf{z})| \cdot s\delta^n. \end{aligned}$$

Solving for $f_Y(y)$ leads to

$$f_Y(y) = f(\mathrm{g}^{-1}(\mathbf{y}))\frac{1}{|\det M(\mathbf{z})|}.$$

Starting off with the whitening transformation, we rearrange for invertible A:

$$\begin{aligned} & Y &&= \mathrm{g}(Z) \\ \Leftrightarrow\quad & Y &&= AZ + \boldsymbol{\mu} \\ \Leftrightarrow\quad & A^{-1}(Y - \boldsymbol{\mu}) &&= Z. \end{aligned}$$

Now, utilizing change of measure:

$$\begin{aligned} f_Y(\mathbf{y}) &= \mathrm{f}_Z(\mathrm{g}^{-1}(\mathbf{y})) \cdot \left|\det\frac{\mathrm{d}}{\mathrm{d}\mathbf{y}}\mathrm{g}^{-1}(\mathbf{y})\right| \\ &= \mathrm{f}_Z(\mathrm{A}^{-1}(\mathbf{y} - \boldsymbol{\mu})) \cdot \left|\det\frac{\mathrm{d}}{\mathrm{d}\mathbf{y}}[\mathrm{A}^{-1}(\mathrm{Y} - \boldsymbol{\mu})]\right| \\ &= (2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}[\mathrm{A}^{-1}(\mathbf{y}-\boldsymbol{\mu})]^T\mathrm{A}^{-1}(\mathbf{y}-\boldsymbol{\mu})} \cdot |\det A^{-1}| \\ &= (2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\mathrm{A}^{-2}(\mathbf{y}-\boldsymbol{\mu})} \cdot |\det A^{-1}|. \end{aligned}$$

Comparing coefficients with a $\mathrm{N}(\mathbf{y}|\boldsymbol{\mu}, \Sigma)$ distributed random variable:

$$\begin{aligned} & \mathrm{A}^{-2} &&= \Sigma^{-1} \\ \Longrightarrow\quad & \mathrm{A} &&= \Sigma^{\frac{1}{2}}. \end{aligned}$$

Substituting the result into the original equation:

$$\begin{aligned} f_Y(\mathbf{y}) &= (2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\mathrm{A}^{-2}(\mathbf{y}-\boldsymbol{\mu})} \cdot |\det A^{-1}| \\ &= (2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\mathrm{A}^{-1}(\mathbf{y}-\boldsymbol{\mu})} \cdot \left|\underbrace{\det\Sigma^{-\frac{1}{2}}}_{>0, positive\ definiteness}\right| \\ &= (2\pi)^{-\frac{1}{2}}\det\Sigma^{-\frac{1}{2}}\ e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\mathrm{A}^{-2}(\mathbf{y}-\boldsymbol{\mu})}. \end{aligned}$$

6A-4

Auxiliary calculation

A sample y from a continuous distribution must fulfill:

$$P(a \leq y \leq b) = F_X(b) - F_X(a)$$

.

If u is uniform, since $Y = F_X(a)$ is also uniform we have

$$P(a \leq F_X^{-1}(u) \leq b) = P\big(F_X(a) \leq u \leq F_X(b)\big) = F_X(b) - F_X(a).$$

So $y = F_X^{-1}(u)$ fulfills our sample criterium.