Rare-event simulation: High-performance Python

Patrick Laub

March 27, 2020

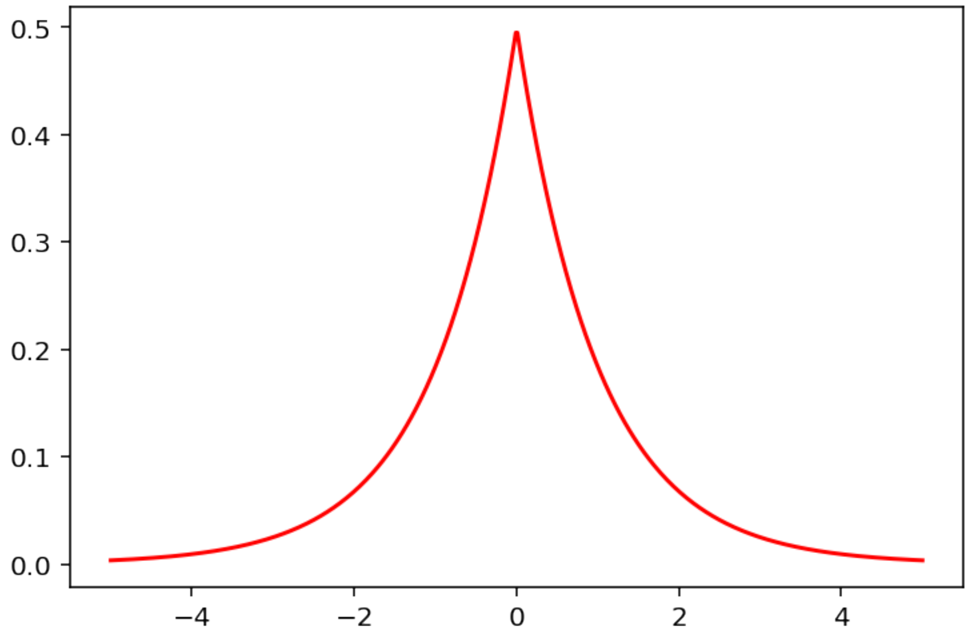
Import relevant libraries

[1]:
```
# numpy is the 'Numerical Python' package
import numpy as np

# Numpy's methods for pseudorandom number generation
import numpy.random as rnd

# For plotting
import matplotlib.pyplot as plt

# scipy is the 'Scientific Python' package
# We'll use the stats package to get some p.d.f.s.
from scipy import stats

%config InlineBackend.figure_format = 'retina'
```

1 Sampling a Laplace distribution with MCMC

$$X \sim \mathsf{Laplace}(\mu, \lambda) \quad \Rightarrow \quad f_X(x) = \frac{1}{2\lambda} \exp\left\{ \frac{|x-\mu|}{\lambda} \right\}.$$

[2]:
```
xs = np.linspace(-5,5, 500)
plt.plot(xs, stats.laplace.pdf(xs), 'r');
```



[3]:
```
def sample(R):
    rng = rnd.default_rng(1)

    π = stats.laplace.pdf

    X = np.empty(R)
    X[0] = 0

    for n in range(1, R):
        Y = X[n-1] + rng.normal()

        α = π(Y) / π(X[n-1])

        if rng.uniform() < α:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

1.1 Measure the problem

Before timing any code, put turn off battery saver modes.

[4]:
```
%time X = sample(10**2)
```

```
Wall time: 26.5 ms
```

[5]:
```
26.5 / 1000 * 100
```

[5]:
```
2.65
```

[6]:
```
%time X = sample(10**4)
```

```
Wall time: 1.68 s
```

[7]:
```
1.68 * 100 / 60
```

[7]:
```
2.8
```

[8]:
```
%timeit X = sample(1)
```

```
29.4 µs ± 727 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[9]:
```
%load_ext line_profiler
```

[10]:
```
%lprun -f sample sample(10**4)
```

```
Timer unit: 1e-07 s

Total time: 2.88904 s
File: <ipython-input-3-0ab92f3542ac>
Function: sample at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def sample(R):
     2         1       1618.0   1618.0      0.0      rng = rnd.default_rng(1)
     3
     4         1         30.0     30.0      0.0      π = stats.laplace.pdf
     5
     6         1         66.0     66.0      0.0      X = np.empty(R)
     7         1         15.0     15.0      0.0      X[0] = 0
     8
     9     10000      42983.0      4.3      0.1      for n in range(1, R):
    10      9999     406224.0     40.6      1.4          Y = X[n-1] + rng.normal()
    11
    12      9999   27920074.0   2792.3     96.6          α = π(Y) / π(X[n-1])
    13
    14      9999     440077.0     44.0      1.5          if rng.uniform() < α:
    15      7043      48084.0      6.8      0.2              X[n] = Y
    16                                                   else:
    17      2956      31274.0     10.6      0.1              X[n] = X[n-1]
    18
```

```
    19         1          3.0      3.0      0.0      return X
```

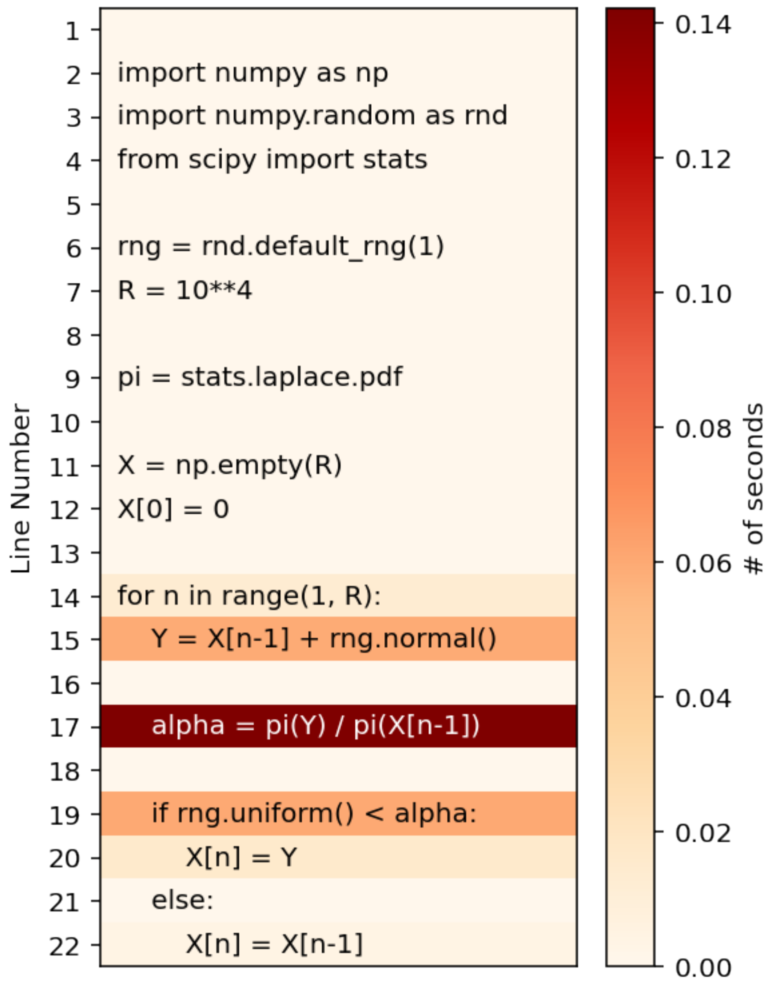
[11]:
```
%lprun -f stats.laplace.pdf sample(10**4)
```

```
Timer unit: 1e-07 s

Total time: 2.79672 s
File: C:\Users\patri\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py
Function: pdf at line 1714

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
  1714                                           def pdf(self, x, *args, **kwds):
  1715                                               """
  1716                                               Probability density function at x
  1717
  1718                                               Parameters
  1719                                               ----------
  1720                                               x : array_like
  1721                                                   quantiles
  1722                                               arg1, arg2, arg3,... : array_like
  1723                                                   The shape parameter(s) for th
  1724                                                   instance object for more info
  1725                                               loc : array_like, optional
  1726                                                   location parameter (default=0
  1727                                               scale : array_like, optional
  1728                                                   scale parameter (default=1)
  1729
  1730                                               Returns
  1731                                               -------
  1732                                               pdf : ndarray
  1733                                                   Probability density function
  1734
  1735                                               """
  1736     19998     244063.0     12.2      0.9      args, loc, scale = self._parse_ar
  1737     19998     805908.0     40.3      2.9      x, loc, scale = map(asarray, (x,
  1738     19998     199397.0     10.0      0.7      args = tuple(map(asarray, args))
  1739     19998    6459118.0    323.0     23.1      dtyp = np.find_common_type([x.dty
  1740     19998     881695.0     44.1      3.2      x = np.asarray((x - loc)/scale, d
  1741     19998    1069852.0     53.5      3.8      cond0 = self._argcheck(*args) & (
  1742     19998    1017517.0     50.9      3.6      cond1 = self._support_mask(x, *ar
  1743     19998     580429.0     29.0      2.1      cond = cond0 & cond1
  1744     19998     715135.0     35.8      2.6      output = zeros(shape(cond), dtyp)
  1745     19998    1573239.0     78.7      5.6      putmask(output, (1-cond0)+np.isna
  1746     19998    2280964.0    114.1      8.2      if np.any(cond):
  1747     19998    9581439.0    479.1     34.3          goodargs = argsreduce(cond, *
  1748     19998     215533.0     10.8      0.8          scale, goodargs = goodargs[-1
```

```
  1749     19998     2063593.0    103.2      7.4              place(output, cond, self._pdf
  1750     19998      141475.0      7.1      0.5          if output.ndim == 0:
  1751     19998      137848.0      6.9      0.5              return output[()]
  1752                                                    return output
```

[12]:
```
%load_ext heat
```

[13]:
```
%%heat

import numpy as np
import numpy.random as rnd
from scipy import stats

rng = rnd.default_rng(1)
R = 10**4

pi = stats.laplace.pdf

X = np.empty(R)
X[0] = 0

for n in range(1, R):
    Y = X[n-1] + rng.normal()

    alpha = pi(Y) / pi(X[n-1])

    if rng.uniform() < alpha:
        X[n] = Y
    else:
        X[n] = X[n-1]
```



[14]:
```
%load_ext snakeviz
```

[15]:
```
%snakeviz X = sample(10**4)
```

```
*** Profile stats marshalled to file
'C:\\Users\\patri\\AppData\\Local\\Temp\\tmpn9il9v6r'.
Embedding SnakeViz in this document…
```

```
<IPython.core.display.HTML object>
```

1.2 Check improvements one-by-one

Replace built-in Laplace p.d.f. with a version we have made.

[16]:
```
xs = np.linspace(-5, 5, 11)
old = stats.laplace.pdf(xs)
new = np.exp(-np.abs(xs))/2
old - new
```

[16]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[17]:
```
xs = np.linspace(-5, 5, 10**5)
%timeit stats.laplace.pdf(xs)
%timeit np.exp(-np.abs(xs))  # Don't need normalising constant
```

```
5.58 ms ± 315 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
1.2 ms ± 35 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

[18]:
```
5.58 / 1.2
```

[18]:
```
4.65
```

[19]:
```
xs = np.linspace(-5, 5, 10**5)
%timeit [stats.laplace.pdf(x) for x in xs]
%timeit [np.exp(-np.abs(x)) for x in xs]
```

```
7.37 s ± 211 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
233 ms ± 1.83 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[20]:
```
7.37 / 0.233
```

[20]:
```
31.630901287553648
```

[21]:
```
samplePrev = sample
```

[22]:
```
def sample(R):
    rng = rnd.default_rng(1)

    π = lambda x: np.exp(-np.abs(x))

    X = np.empty(R)
    X[0] = 0

    for n in range(1, R):
        Y = X[n-1] + rng.normal()

        α = π(Y) / π(X[n-1])

        if rng.uniform() < α:
            X[n] = Y
        else:
```

```python
            X[n] = X[n-1]

    return X
```

[23]:
```python
print(samplePrev(5))
print(sample(5))
```

```
[ 0.          0.          0.          0.         -0.53695324]
[ 0.          0.          0.          0.         -0.53695324]
```

[24]:
```python
%time X = samplePrev(10**5)
%time X = sample(10**5)
```

```
Wall time: 16.3 s
Wall time: 987 ms
```

[25]:
```python
16.3 / 0.987
```

[25]:
```
16.51469098277609
```

[26]:
```python
%lprun -f sample sample(10**5)
```

```
Timer unit: 1e-07 s

Total time: 1.38244 s
File: <ipython-input-22-2f3c9d85c13d>
Function: sample at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def sample(R):
     2         1       1803.0   1803.0      0.0      rng = rnd.default_rng(1)
     3
     4         1         10.0     10.0      0.0      π = lambda x: np.exp(-np.abs(x))
     5
     6         1        160.0    160.0      0.0      X = np.empty(R)
     7         1         15.0     15.0      0.0      X[0] = 0
     8
     9    100000     425389.0      4.3      3.1      for n in range(1, R):
    10     99999    3331726.0     33.3     24.1          Y = X[n-1] + rng.normal()
    11
    12     99999    6631665.0     66.3     48.0          α = π(Y) / π(X[n-1])
    13
    14     99999    2774220.0     27.7     20.1          if rng.uniform() < α:
    15     70184     421547.0      6.0      3.0              X[n] = Y
    16                                                   else:
    17     29815     237841.0      8.0      1.7              X[n] = X[n-1]
    18
```

```
    19          1           3.0      3.0      0.0      return X
```

Let's try vectorising the random number generation

[27]:
```
samplePrev = sample
```

[28]:
```
def sample(R):
    rng = rnd.default_rng(1)

    π = lambda x: np.exp(-np.abs(x))

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    uniforms = rng.uniform(size=R-1)

    for n in range(1, R):
        Y = X[n-1] + jumps[n-1]

        α = π(Y) / π(X[n-1])

        if uniforms[n-1] < α:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

[29]:
```
print(samplePrev(5))
print(sample(5))
```

```
[ 0.          0.          0.          0.         -0.53695324]
[ 0.          0.34558419  1.16720234  1.16720234 -0.1359549 ]
```

[30]:
```
%time X = samplePrev(10**6)
%time X = sample(10**6)
```

```
Wall time: 9.98 s
Wall time: 6.14 s
```

[31]:
```
9.98 / 6.14
```

[31]: 1.6254071661237786

[32]:
```
%lprun -f sample sample(10**6)
```

```
Timer unit: 1e-07 s
```

```
Total time: 9.0506 s
File: <ipython-input-28-f0fc8c08d600>
Function: sample at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def sample(R):
     2         1       1906.0   1906.0      0.0      rng = rnd.default_rng(1)
     3
     4         1         21.0     21.0      0.0      π = lambda x: np.exp(-np.abs(x))
     5
     6         1        406.0    406.0      0.0      X = np.empty(R)
     7         1         21.0     21.0      0.0      X[0] = 0
     8
     9         1     224605.0 224605.0      0.2      jumps = rng.normal(size=R-1)
    10         1     109040.0 109040.0      0.1      uniforms = rng.uniform(size=R-1)
    11
    12   1000000    4178819.0      4.2      4.6      for n in range(1, R):
    13    999999    9092839.0      9.1     10.0          Y = X[n-1] + jumps[n-1]
    14
    15    999999   64142919.0     64.1     70.9          α = π(Y) / π(X[n-1])
    16
    17    999999    6992107.0      7.0      7.7          if uniforms[n-1] < α:
    18    700380    3681116.0      5.3      4.1              X[n] = Y
    19                                                   else:
    20    299619    2082243.0      6.9      2.3              X[n] = X[n-1]
    21
    22         1          3.0      3.0      0.0      return X
```

Let's try getting rid of the exponential in the p.d.f.

[33]:
```
samplePrev = sample
```

[34]:
```
def sample(R):
    rng = rnd.default_rng(1)

    logπ = lambda x: -np.abs(x)

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    exponentials = np.log(rng.uniform(size=R-1)) # Seems faster than rng.
 ↪exponential

    for n in range(1, R):
```

```
        Y = X[n-1] + jumps[n-1]

        logα = logπ(Y) - logπ(X[n-1])

        if exponentials[n-1] < logα:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

[35]:
```
print(samplePrev(5))
print(sample(5))
```

```
[ 0.          0.34558419  1.16720234  1.16720234 -0.1359549 ]
[ 0.          0.34558419  1.16720234  1.16720234 -0.1359549 ]
```

[36]:
```
%time X = samplePrev(10**6)
%time X = sample(10**6)
```

```
Wall time: 6.06 s
Wall time: 3.5 s
```

[37]:
```
6.06 / 3.5
```

[37]: 1.7314285714285713

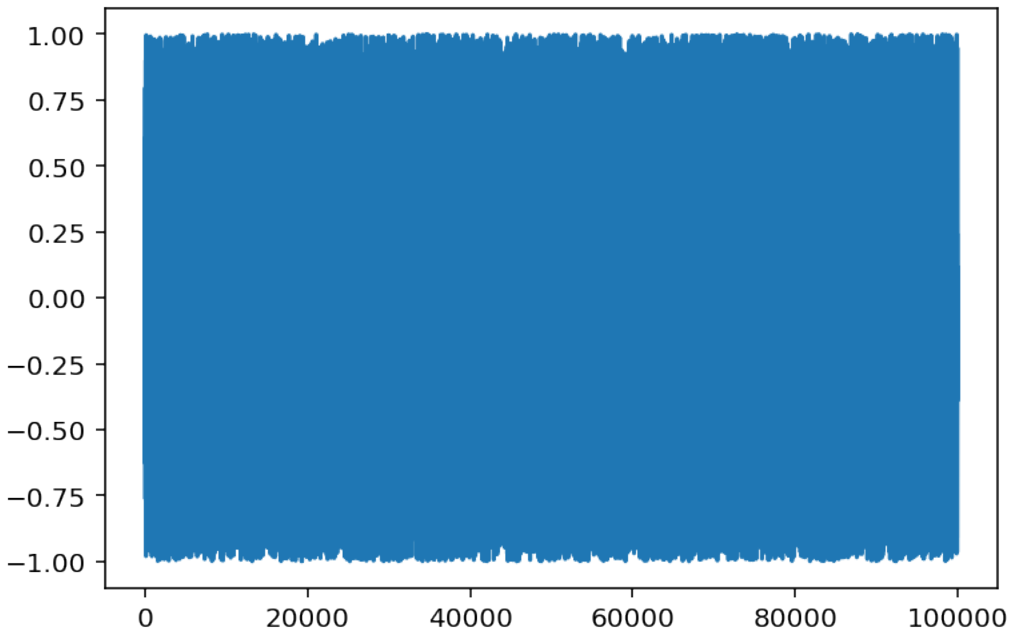
1.3 Sample from a truncated Laplace distribution

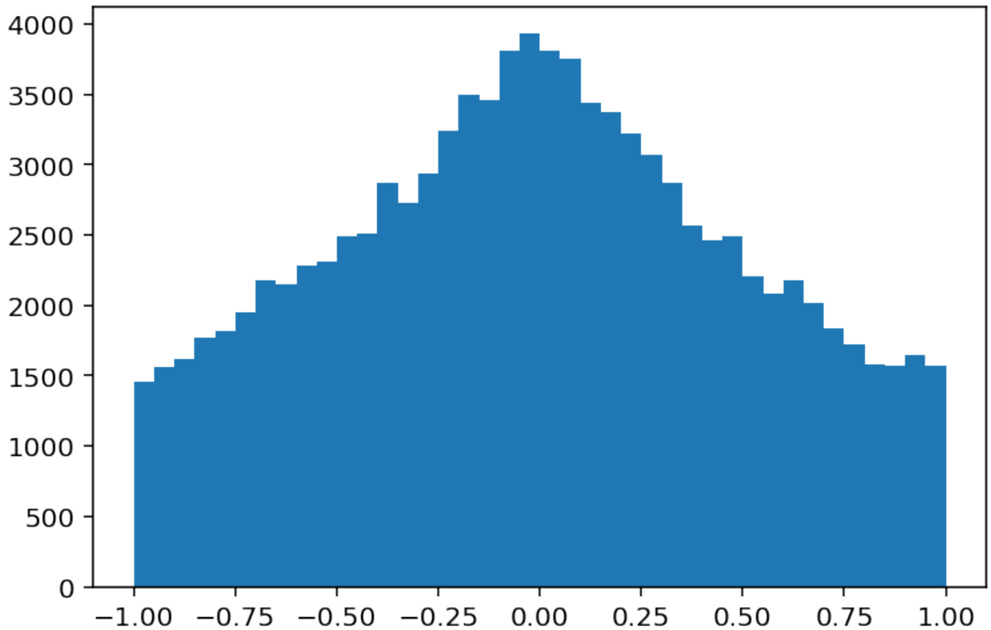
[38]:
```
def sample(R):
    rng = rnd.default_rng(1)

    π = lambda x: (x > -1) * (x < 1) * np.exp(-np.abs(x))

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    uniforms = rng.uniform(size=R-1)

    for n in range(1, R):
        Y = X[n-1] + jumps[n-1]

        α = π(Y) / π(X[n-1])

        if uniforms[n-1] < α:
            X[n] = Y
```

```
        else:
            X[n] = X[n-1]

    return X
```

[39]:
```
%time X = sample(10**5)

plt.plot(X)
plt.show()

plt.hist(X, 40);
```

```
Wall time: 1.45 s
```





[40]:
```
np.mean(np.diff(X) == 0)
```

[40]: 0.4680446804468045

[41]:
```
samplePrev = sample
```

[42]:
```
def sample(R):
    rng = rnd.default_rng(1)

    πUn = lambda x: np.exp(-np.abs(x))

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    uniforms = rng.uniform(size=R-1)

    for n in range(1, R):
        Y = X[n-1] + jumps[n-1]

        # Check the constraint first
        if Y <= -1 or Y >= 1:
            X[n] = X[n-1]
            continue

        # Then, if a valid proposal,
```

```
        # calculate the acceptance prob.
        α = πUn(Y) / πUn(X[n-1])

        if uniforms[n-1] < α:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

[43]:
```
print(samplePrev(5))
print(sample(5))
```

```
[ 0.          0.34558419  0.34558419  0.34558419 -0.95757304]
[ 0.          0.34558419  0.34558419  0.34558419 -0.95757304]
```

[44]:
```
%time X = samplePrev(10**6)
%time X = sample(10**6)
```

```
Wall time: 14.6 s
Wall time: 4.11 s
```

[45]:
```
14.6 / 4.11
```

[45]: 3.552311435523114

1.4 Try compiling the algorithm with numba

[46]:
```
from numba import njit
```

[47]:
```
samplePrev = sample
```

[48]:
```
@njit
def sample(R):
    rng = rnd.default_rng(1)

    πUn = lambda x: np.exp(-np.abs(x))

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    uniforms = rng.uniform(size=R-1)

    for n in range(1, R):
        Y = X[n-1] + jumps[n-1]
```

```
        # Check the constraint first
        if Y <= -1 or Y >= 1:
            X[n] = X[n-1]
            continue

        # Then, if a valid proposal,
        # calculate the acceptance prob.
        α = πUn(Y) / πUn(X[n-1])

        if uniforms[n-1] < α:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

[49]:
```
sample(5)
```

```
      ↪
↪---------------------------------------------------------------------------

      TypingError                               Traceback (most recent call↪
↪last)

      <ipython-input-49-dfc5eee7c6c4> in <module>
  ----> 1 sample(5)


      ~\Anaconda3\lib\site-packages\numba\dispatcher.py in↪
↪_compile_for_args(self, *args, **kws)
      399                     e.patch_message(msg)
      400
  --> 401                 error_rewrite(e, 'typing')
      402             except errors.UnsupportedError as e:
      403                 # Something unsupported is present in the user code,↪
↪add help info


      ~\Anaconda3\lib\site-packages\numba\dispatcher.py in error_rewrite(e,↪
↪issue_type)
      342                     raise e
      343                 else:
  --> 344                     reraise(type(e), e, None)
      345
      346             argtypes = []
```

```
    ~\Anaconda3\lib\site-packages\numba\six.py in reraise(tp, value, tb)
    666                 value = tp()
    667             if value.__traceback__ is not tb:
--> 668                 raise value.with_traceback(tb)
    669             raise value
    670


    TypingError: Failed in nopython mode pipeline (step: nopython
↪frontend)
  Unknown attribute 'default_rng' of type Module(<module 'numpy.random'
↪from 'C:
↪\\Users\\patri\\Anaconda3\\lib\\site-packages\\numpy\\random\\__init__.
↪py'>)

  File "<ipython-input-48-bced36de9aed>", line 3:
  def sample(R):
      rng = rnd.default_rng(1)
      ^

  [1] During: typing of get attribute at <ipython-input-48-bced36de9aed> (3)

  File "<ipython-input-48-bced36de9aed>", line 3:
  def sample(R):
      rng = rnd.default_rng(1)
      ^
```

[50]:

```
def sample(R):
    rng = rnd.default_rng(1)

    X = np.empty(R)
    X[0] = 0

    jumps = rng.normal(size=R-1)
    uniforms = rng.uniform(size=R-1)

    sample_jit(X, jumps, uniforms)

    return X

@njit
def sample_jit(X, jumps, uniforms):
    R = len(X)
```

```
    πUn = lambda x: np.exp(-np.abs(x))

    for n in range(1, R):
        Y = X[n-1] + jumps[n-1]

        # Check the constraint first
        if Y <= -1 or Y >= 1:
            X[n] = X[n-1]
            continue

        # Then, if a valid proposal,
        # calculate the acceptance prob.
        α = πUn(Y) / πUn(X[n-1])

        if uniforms[n-1] < α:
            X[n] = Y
        else:
            X[n] = X[n-1]
```

[51]:
```
%time X = sample(10**6)
%time X = sample(10**6)
```

```
Wall time: 242 ms
Wall time: 41 ms
```

[52]:
```
print(samplePrev(5))
print(sample(5))
```

```
[ 0.          0.34558419  0.34558419  0.34558419 -0.95757304]
[ 0.          0.34558419  0.34558419  0.34558419 -0.95757304]
```

[53]:
```
%time X = samplePrev(10**6)
%time X = sample(10**6)
```

```
Wall time: 4.67 s
Wall time: 41.9 ms
```

[54]:
```
4.67 / 0.0419
```

[54]: 111.45584725536993

[55]:
```
from numba import int64, float64
```

[56]:
```
samplePrev = sample
```

[57]:
```
@njit(float64[:](int64))
def sample(R):
```

```
    rnd.seed(123)
    X = np.empty(R)
    X[0] = 0
    for n in range(1, R):
        Y = X[n-1] + rnd.normal(0, 1)

        α = (Y > -1) * (Y < 1) * np.exp(-np.abs(Y)+np.abs(X[n-1]))

        if rnd.uniform(0, 1) < α:
            X[n] = Y
        else:
            X[n] = X[n-1]

    return X
```

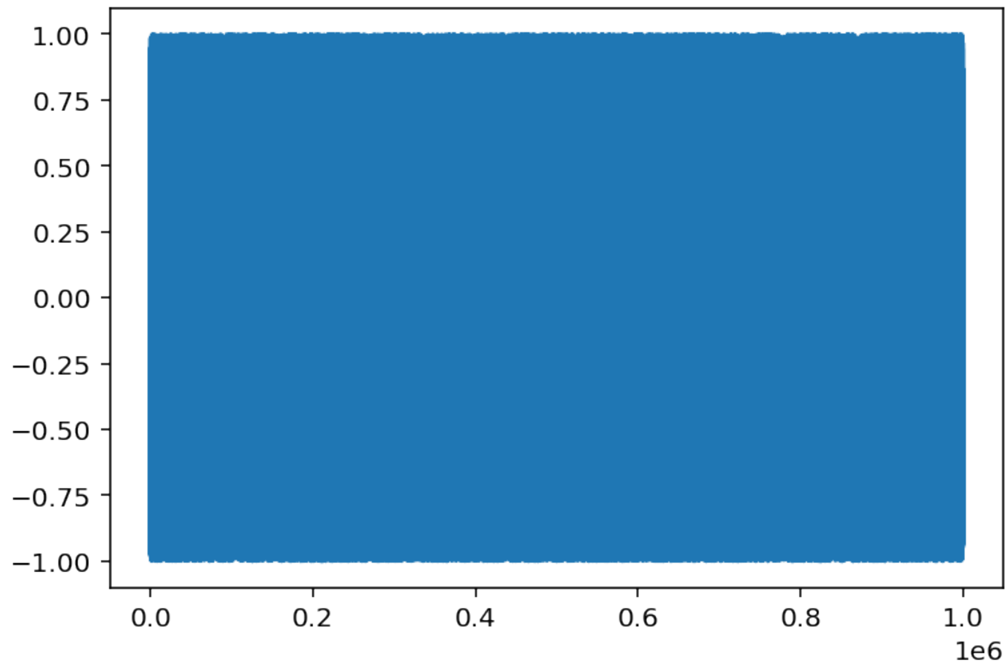
[58]:
```
%time X = sample(10**7)
%time X = sample(10**7)
```

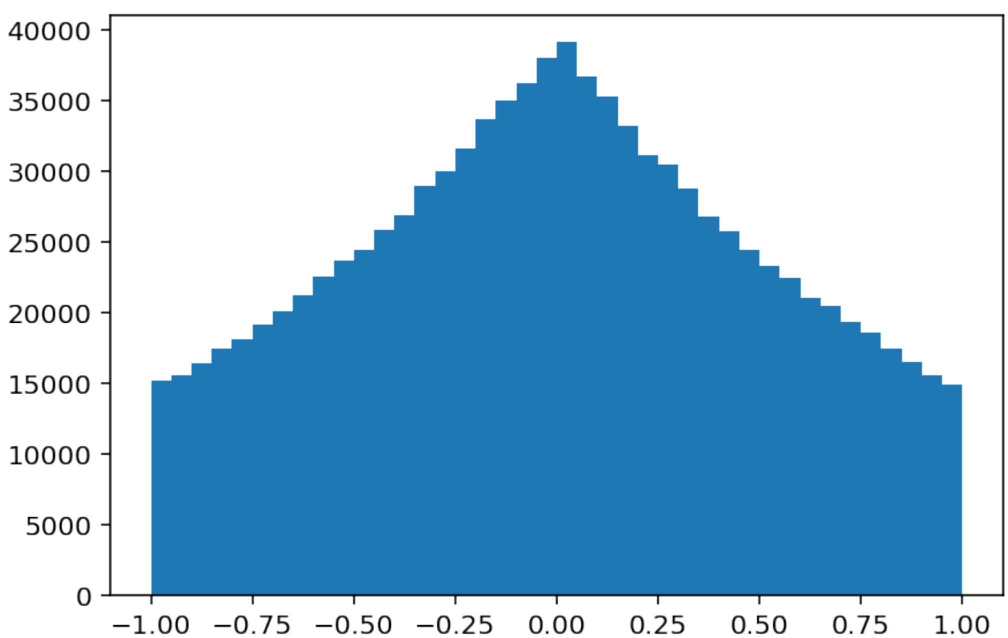
```
Wall time: 572 ms
Wall time: 584 ms
```

[59]:
```
%timeit X = samplePrev(10**7)
%timeit X = sample(10**7)
```

```
400 ms ± 8.55 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
578 ms ± 31 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[60]:
```
plt.plot(X[:10**6])
plt.show()

plt.hist(X[:10**6], 40);
```





Can get a little faster by noticing that each π function call is called (at least) twice with the same arguments. If the result is stored/cached, then we get faster but uglier code, so I'll stop here. Similarly, one can try to simulate using a truncated proposal so that invalid points are never proposed.

1.5 Keep in mind

Improvements to the algorithm and your choice of hyperparameters are often a better starting point than going down a rabbit-hole of performance optimisations!

Updating Python and its packages may give you a free small speed boost (or maybe it will slow things down). With this numpy update, I tested CMC before and after and the time went from 5m 4s down to 3m 54s.

[61]:
```python
from IPython.display import Image
Image("numpy_update.png")
```

[61]:

NumPy 1.18.2 Release Notes

This small elease contains a fix for a performance regression in numpy/random and several bug/maintenance updates.

The Python versions supported in this release are 3.5-3.8. Downstream developers should use Cython >= 0.29.15 for Python 3.8 support and OpenBLAS >= 3.7 to avoid errors on the Skylake architecture.

Contributors

A total of 5 people contributed to this release. People with a "+" by their names contributed a patch for the first time.

- Charles Harris
- Ganesh Kathiresan +
- Matti Picus
- Sebastian Berg
- przemb +

Pull requests merged

A total of 7 pull requests were merged for this release.

- #15675: TST: move _no_tracing to testing._private
- #15676: MAINT: Large overhead in some random functions
- #15677: TST: Do not create gfortran link in azure Mac testing.
- #15679: BUG: Added missing error check in ndarray.__contains__
- #15722: MAINT: use list-based APIs to call subprocesses