Import relevant libraries

```python
# 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

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

```python
[2]: xs = np.linspace(-5, 5, 500)
plt.plot(xs, stats.laplace.pdf(xs), 'r');
```
def sample(R):
    rng = rnd.default_rng(1)
    
    \( \pi \) = stats.laplace.pdf

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

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

        \( \alpha \) = \( \frac{\pi(Y)}{\pi(X[n-1])} \)

        if rng.uniform() < \( \alpha \):
            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

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

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

### 1.2 Check improvements one-by-one

Replace built-in Laplace p.d.f. with a version we have made.
```python
[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.])

[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:
```

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

return X

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

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

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

Wall time: 16.3 s
Wall time: 987 ms

[25]:
16.3 / 0.987

[25]:
16.51469098277609

[26]:
%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

<table>
<thead>
<tr>
<th>Line #</th>
<th>Hits</th>
<th>Time</th>
<th>Per Hit</th>
<th>% Time</th>
<th>Line Contents</th>
</tr>
</thead>
</table>
| 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     |        |        |         |        |               |
Let's try vectorising the random number generation

```python
[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.   -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

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

Timer unit: 1e-07 s
Let's try getting rid of the exponential in the p.d.f.

```python
[33]: samplePrev = sample

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

    logpi = 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.

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

\[ Y = X[n-1] + \text{jumps}[n-1] \]

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

\[
\text{if } \text{exponentials}[n-1] < \log \alpha: \\
X[n] = Y \\
\text{else:} \\
X[n] = X[n-1]
\]

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

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

[40]: 0.4680446804468045

[41]: samplePrev = sample

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

    \( \pi_{\text{Un}} \) = 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.
\[ \alpha = \pi \text{Un}(Y) / \pi \text{Un}(X[n-1]) \]

```python
if uniforms[n-1] < \alpha:
    X[n] = Y
else:
    X[n] = X[n-1]

return X
```

```python
[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]
```

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

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

```python
[45]: 14.6 / 4.11
[45]: 3.552311435523114
```

1.4 Try compiling the algorithm with numba

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

[47]: samplePrev = sample

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

    \( \pi \text{Un} = \text{lambda } x: \text{np.exp(-np.abs(x))} \)

    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
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)
\[ \pi_{Un} = \lambda x : \text{np.exp(-np.abs(x))} \]

```python
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.
    \[ \alpha = \pi_{Un}(Y) / \pi_{Un}(X[n-1]) \]

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

```bash
[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 \( \pi \) 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]: from IPython.display import Image
Image("numpy_update.png")
```

NumPy 1.18.2 Release Notes

This small release 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