Rare-event simulation: Code demo PyMC3

Patrick Laub
March 26, 2020

[1]:
```python
import numpy as np
import pandas as pd
import pymc3 as pm

%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

[2]:
```python
import sys
print("Python version:", sys.version)
print("Numpy version:", np.__version__)
print("PyMC3 version:", pm.__version__)
```

Python version: 3.7.7 (default, Mar 23 2020, 23:19:08) [MSC v.1916 64 bit (AMD64)]
Numpy version: 1.18.1
PyMC3 version: 3.8

[3]:
```python
df = pd.read_csv("intervals.csv")
```

[4]:
```python
df.head()
```

<table>
<thead>
<tr>
<th>EL</th>
<th>ER</th>
<th>SL</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>45.999306</td>
<td>49.0</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>43.999306</td>
<td>44.0</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>49.375000</td>
<td>53.0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>35.999306</td>
<td>35.0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>43.999306</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Figure 1 from Reich et al. (2009), *Estimating incubation period distributions with coarse data*.

[5]: `Tmin = np.array(np.maximum(df["SL"]-df["ER"], 0))`
    `Tmax = np.array(df["SR"]-df["EL"])`

[6]: `plt.hist(Tmin);`

[7]: `plt.hist(Tmax);`
xs = np.linspace(0, 75, 500)
ys = np.zeros(len(xs))

nCases = len(Tmin)
for i in range(nCases):
    ys[(xs >= Tmin[i]) & (xs <= Tmax[i])] += 1 /
        (nCases * (Tmax[i] - Tmin[i]))

plt.plot(xs, ys);
\[
T_{avgs} = \frac{(T_{min} + T_{max})}{2}
\]

```python
%%time
with pm.Model() as model:
    μ = pm.Uniform('μ', lower=-25, upper=25)
    σ = pm.Uniform('σ', lower=0, upper=25)
    T = pm.Lognormal('T', mu=μ, sigma=σ, observed=Tavgs)
trace = pm.sample(10**4)

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [σ, μ]
Sampling 4 chains, 0 divergences:
100%|█████████████████████████████████████████████████████████| 42000/42000
[00:19<00:00, 2124.38draws/s]

Wall time: 53 s
```

```python
pm.plot_trace(trace);
```
This is assuming we have observations $T = (T_1, \ldots, T_n)$ which gives us a likelihood of

$$L(\mu, \sigma \mid T) = \prod_{i=1}^{n} f(T_i; \mu, \sigma)$$

where $f(x; \mu, \sigma)$ is the p.d.f. of the LogNormal($\mu, \sigma^2$) distribution.

However we don’t have observations, just intervals. Say each unobserved period fell into $T_i \in [T_i^-, T_i^+]$. Our likelihood becomes

$$L(\mu, \sigma \mid T^-, T^+) = \prod_{i=1}^{n} [F(T_i^+; \mu, \sigma) - F(T_i^-; \mu, \sigma)]$$

where $F(x; \mu, \sigma)$ is the c.d.f. of the LogNormal($\mu, \sigma^2$) distribution.

[13]: `import theano.tensor as tt`

# Taken from PyMC3’s pymc3/distributions/dist_math.py file
# starting at line 346.
def zvalue(x, sigma, mu):
    ""
    Calculate the z-value for a normal distribution.
    ""
    return (x - mu) / sigma

# Taken from PyMC3's pymc3/distributions/continuous.py file
# starting at line 1849.
def cdf(x, mu, sigma):
    ""
    Compute the log of the cumulative distribution function for
    Lognormal distribution
    at the specified value.
    Parameters
    ----------
    x: numeric
        Value(s) for which log CDF is calculated. If the log CDF for
    multiple
        values are desired the values must be provided in a numpy array
    or theano tensor.
    Returns
    -------
    TensorVariable
    ""
    z = zvalue(np.log(x), mu=mu, sigma=sigma)
    return tt.switch(
        tt.lt(z, -1.0),
        tt.erfcx(-z / tt.sqrt(2.)) / 2. * np.exp(-tt.sqr(z) / 2),
        tt.erfc(-z / tt.sqrt(2.)) / 2.
    )

With Potential we have to add log-terms to the likelihood. So

\[
\log[L(\mu, \sigma \mid T^-, T^+)] = \sum_{i=1}^{n} \log[F(T_i^+; \mu, \sigma) - F(T_i^-; \mu, \sigma)] .
\]

[14]: %time
with pm.Model() as model:
    μ = pm.Uniform('μ', lower=-25, upper=25)
    σ = pm.Uniform('σ', lower=0, upper=25)
    pm.Potential('T', tt.sum(tt.log( cdf(Tmax, μ, σ) - cdf(Tmin, μ, σ) )))
trace = pm.sample(10**5, step=pm.Metropolis())
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
> Metropolis: [σ]
> Metropolis: [μ]
Sampling 4 chains, 0 divergences:
100% [03:16<00:00, 2040.71draws/s]
The number of effective samples is smaller than 10% for some parameters.
Wall time: 3min 41s

[15]: \texttt{pm.stats.ess(trace["μ"]), pm.stats.ess(trace["σ"])}

[15]: (30595.927940502946, 55537.902861184826)

[16]: \texttt{pm.plot_trace(trace)};

[17]: \texttt{pm.plot_posterior(trace)};

[18]: \texttt{trace["μ"].mean()}
[18]: 1.633400315715902

[19]: trace["o"].mean()

[19]: 0.384109415517976