Machine Learning
MSE FTP MachLe
Christoph Würsch

OST
Ostschweizer
Fachhochschule

Bayesian Regression using `pymc3`

- Author: Christoph Würsch
- Subject: FTP_MachLe (MSE)

Motivation from Cam Davidson Pilon

"The **Bayesian method** is the *natural approach to inference*, yet it is hidden from readers behind chapters of slow, mathematical analysis. The typical text on Bayesian inference involves two to three chapters on probability theory, then enters what Bayesian inference is. Unfortunately, due to mathematical intractability of most Bayesian models, the reader is only shown simple, artificial examples. [...]

If Bayesian inference is the destination, then mathematical analysis is a particular path towards it. On the other hand, computing power is cheap enough that we can afford to take an alternate route via probabilistic programming.

The latter path is much more useful, as it denies the necessity of mathematical intervention at each step, that is, we remove often-intractable mathematical analysis as a prerequisite to Bayesian inference. Simply put, this latter computational path proceeds via small intermediate jumps from beginning to end, where as the first path proceeds by enormous leaps, often landing far away from our target. Furthermore, without a strong mathematical background, the analysis required by the first path cannot even take place."

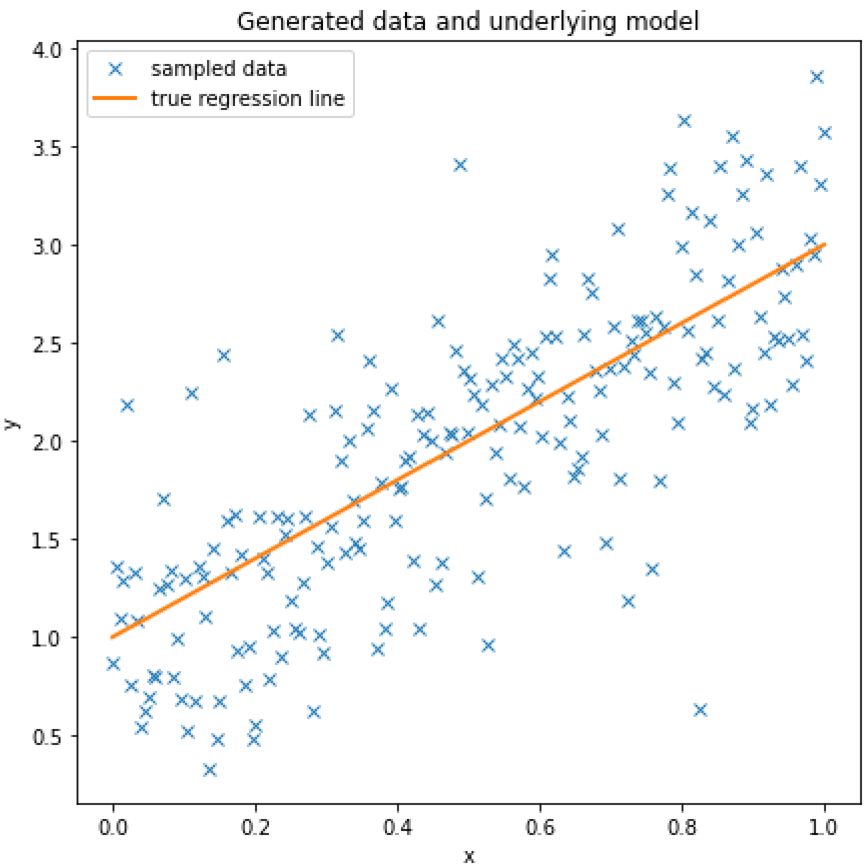
References:

https://docs.pymc.io/ https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers https://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/ https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/zipball/master

In [1]:
```
from pymc3 import HalfCauchy, Normal, sample, plot_trace, plot_posterior_predictive_
import pymc3 as pm
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
size = 200
true_intercept = 1
true_slope = 2

x = np.linspace(0, 1, size)
# y = a + b*x
true_regression_line = true_intercept + true_slope * x
```

```
# add noise
y = true_regression_line + np.random.normal(scale=.5, size=size)

data = dict(x=x, y=y)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(111, xlabel='x', ylabel='y', title='Generated data and underlyi
ax.plot(x, y, 'x', label='sampled data')
ax.plot(x, true_regression_line, label='true regression line', lw=2.)
plt.legend(loc=0);
plt.savefig('LinearRegressionData_pymc3.pdf')
```



Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) algorithm that avoids the random walk behavior and sensitivity to correlated parameters that plague many MCMC methods *by taking a series of steps informed by first-order gradient information*. These features allow it to converge to high-dimensional target distributions much more quickly than simpler methods such as **random walk, Metropolis or Gibbs sampling**.

The **No-U-Turn Sampler (NUTS)** is an extension to HMC that eliminates the need to set a number of steps L. NUTS uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution, stopping automatically when it starts to double back and retrace its steps. Empirically, NUTS performs at least as efficiently as and sometimes more efficiently than a well tuned standard HMC method, without requiring user intervention or costly tuning runs.

The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo
Matthew D. Hoffman, Andrew Gelman https://arxiv.org/abs/1111.4246

In [3]:

```
#%% Bayesian regression

with Model() as model: # model specifications in PyMC3 are wrapped in a with-stateme
```

```
    #tau = precision = 1/variance = 1/sigma^2
    # Define priors
    sigma     = HalfCauchy('sigma', beta=10, testval=1.)
    intercept = Normal('Intercept', 0, tau=1/20**2)
    x_coeff   = Normal('x', 0, tau=1/20**2)

    # Define likelihood
    likelihood = Normal('y', mu=intercept + x_coeff * x,
                        tau=1/sigma**2, observed=y)
    # Inference!
    trace = sample(6000, cores=2,return_inferencedata=False) # draw 3000 posterior s
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [x, Intercept, sigma]
```

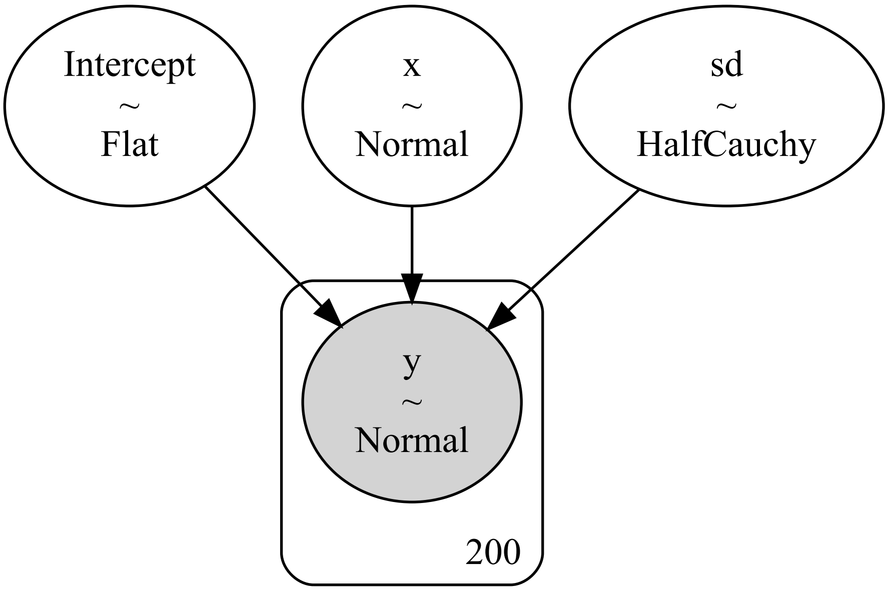
100.00% [14000/14000 00:13<00:00 Sampling 2 chains, 0 divergences]

```
Sampling 2 chains for 1_000 tune and 6_000 draw iterations (2_000 + 12_000 draws tot
al) took 22 seconds.
```

In [7]:

```
pm.model_to_graphviz(model)
```

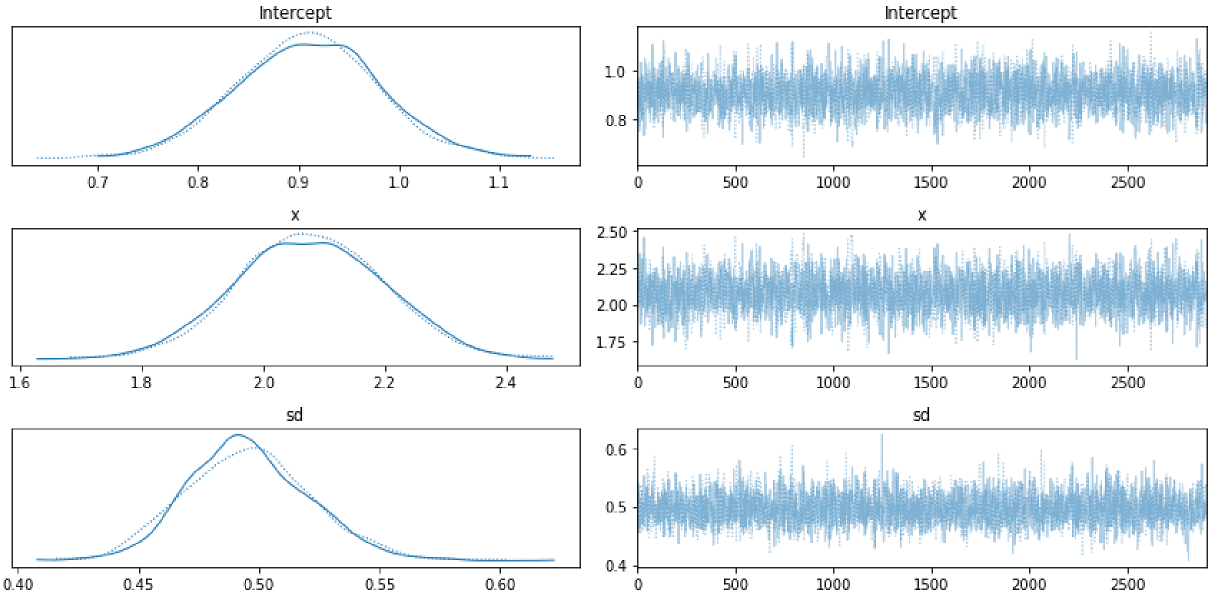
Out[7]:



In [8]:

```
plt.figure(figsize=(7, 7))
plot_trace(trace[100:])
plt.tight_layout();
plt.savefig('pymc3_posterior.pdf')
```

```
Got error No model on context stack. trying to find log_likelihood in translation.
C:\Users\wurc\.conda\envs\pymc3\lib\site-packages\arviz\data\io_pymc3_3x.py:102: Fut
ureWarning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Make sur
e you use the model argument or call from_pymc3 within a model context.
  FutureWarning,
Got error No model on context stack. trying to find log_likelihood in translation.
<Figure size 504x504 with 0 Axes>
```



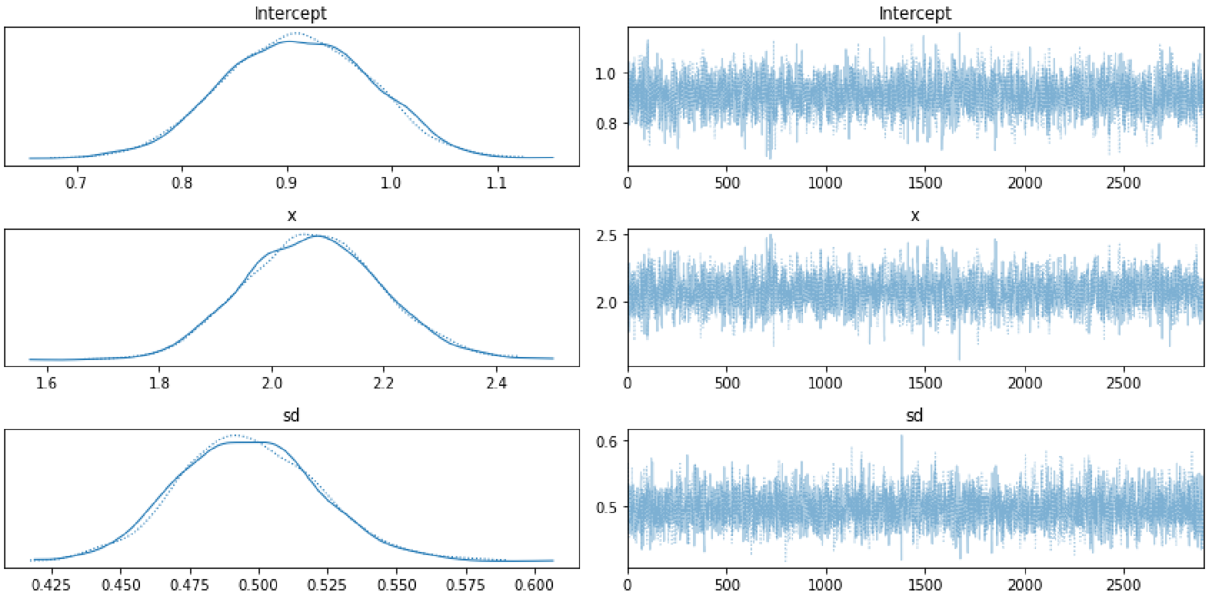
In [9]:

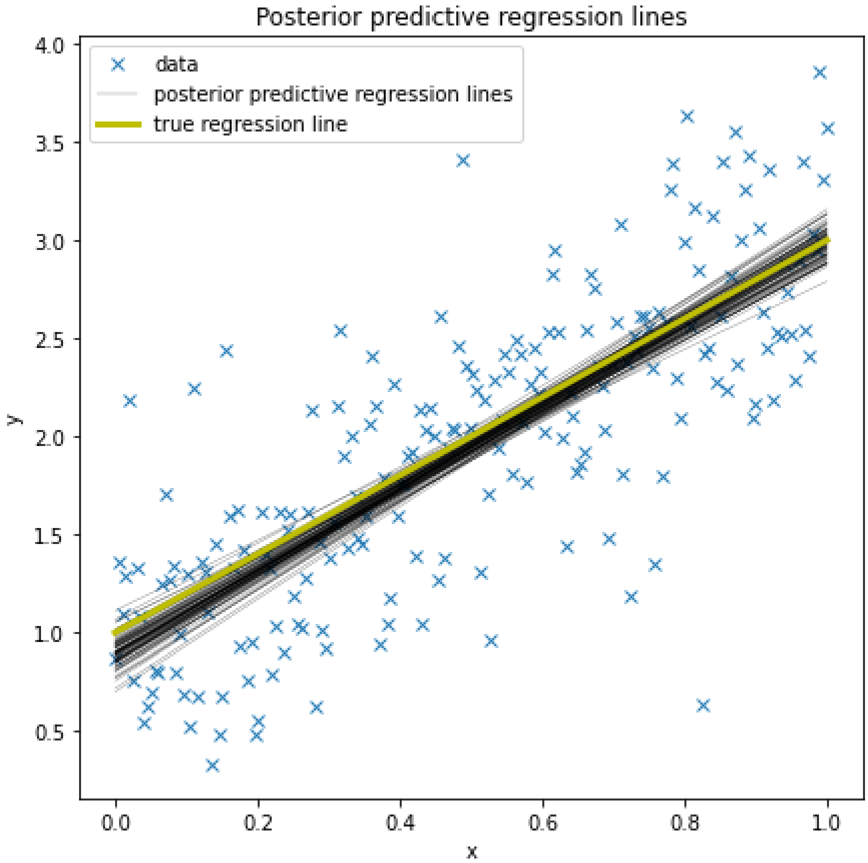
```
#%% glm
#    The new glm() function instead takes a Patsy linear model specifier from
#    which it creates a design matrix. glm() then adds random variables for
#    each of the coefficients and an appopriate likelihood to the model.


with Model() as model:
    # specify glm and pass in data. The resulting linear model, its likelihood and
    # and all its parameters are automatically added to our model.
    glm.GLM.from_formula('y ~ x', data)
    trace = sample(3000, cores=2) # draw 3000 posterior samples using NUTS sampling


#%% glm


plt.figure(figsize=(7, 7))
plot_trace(trace[100:])
plt.tight_layout();

plt.figure(figsize=(7, 7))
plt.plot(x, y, 'x', label='data')
plot_posterior_predictive_glm(trace, samples=100,
                              label='posterior predictive regression lines')
plt.plot(x, true_regression_line, label='true regression line', lw=3., c='y')

plt.title('Posterior predictive regression lines')
plt.legend(loc=0)
plt.xlabel('x')
plt.ylabel('y');
```

```
The glm module is deprecated and will be removed in version 4.0
We recommend to instead use Bambi https://bambinos.github.io/bambi/
C:\Users\wurc\.conda\envs\pymc3\lib\site-packages\ipykernel_launcher.py:11: FutureWa
rning: In v4.0, pm.sample will return an `arviz.InferenceData` object instead of a `
MultiTrace` by default. You can pass return_inferencedata=True or return_inferenceda
ta=False to be safe and silence this warning.
  # This is added back by InteractiveShellApp.init_path()
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [sd, x, Intercept]
```

100.00% [8000/8000 00:09<00:00 Sampling 2 chains, 0 divergences]

```
Sampling 2 chains for 1_000 tune and 3_000 draw iterations (2_000 + 6_000 draws tota
l) took 17 seconds.
Got error No model on context stack. trying to find log_likelihood in translation.
C:\Users\wurc\.conda\envs\pymc3\lib\site-packages\arviz\data\io_pymc3_3x.py:102: Fut
ureWarning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Make sur
e you use the model argument or call from_pymc3 within a model context.
  FutureWarning,
Got error No model on context stack. trying to find log_likelihood in translation.
C:\Users\wurc\.conda\envs\pymc3\lib\site-packages\pymc3\plots\posteriorplot.py:62: D
eprecationWarning: The `plot_posterior_predictive_glm` function will migrate to Arvi
z in a future release.
Keep up to date with `ArviZ <https://arviz-devs.github.io/arviz/>`_ for future updat
es.
  DeprecationWarning,
<Figure size 504x504 with 0 Axes>
```




Posterior predictive regression lines
data
posterior predictive regression lines
true regression line
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
y
0.0
0.2
0.4
0.6
0.8
1.0
x

In [ ]:

```

```

In [ ]:

```

```