# BDA_Assignment_6

October 23, 2022

## Bioassay with Stan

[ ]:
```
# imports and data reading

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import random
import arviz as az
from bioassaylp import bioassaylp
import stan

import nest_asyncio
nest_asyncio.apply()


# read data into dataframes
#bioas_post = pd.read_csv('./bioassay_posterior.txt', header=None,␣
↪names=['alpha','beta'], sep='\t')

bioassay = pd.read_csv('./bioassay.csv')

# in case needed to suppress warnings for nicer prints:
#import warnings
#warnings.filterwarnings('ignore')
```

We start by writing the model in Stan and defining the input data. As suggested, we assume the following Gaussian prior for $\alpha$ and $\beta$:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \sim \mathrm{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \quad \text{where} \quad \boldsymbol{\mu}_0 = \begin{bmatrix}0\\ 10\end{bmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_0 = \begin{bmatrix}2^2 & 12\\ 12 & 10^2\end{bmatrix}.$$

[ ]:
```
bioassay_code = """
data {
int<lower=0> N;          // num on experiments
vector[N] x;             // dose level on log scale
int<lower=0> y[N];       // num of success
```

```
int<lower=0> n[N];      // n from data
vector[2] mu;           // prior means
cov_matrix[2] Sigma;    // prior cov matrix
}
parameters {
vector[2] theta;     // theta[1] = alpha, theta[2] = beta
}
model {
    theta ~ multi_normal(mu, Sigma);
    y ~ binomial_logit(n, (theta[1] + theta[2]*x));
}
"""

bioassay_data = {
    'N': 4,
    'n': bioassay['n'].values,
    'x': bioassay['x'].values,
    'y': bioassay['y'].values,
    'mu': np.array([0,10]),
    'Sigma': np.array([[4,12],[12,100]])
}
```

Next we build the model using the model code and data, and draw the initial points for the chains.

[ ]:
```
post = stan.build(bioassay_code, data=bioassay_data)
```

[ ]:
```
# draw initial points for the chains from the prior distribution
init_points = stats.multivariate_normal.rvs(mean=[0,10],␣
 ↪cov=[[4,12],[12,100]],size=4)
```

Next we draw the samples for 4 different chains. We use warmup length of 2000 and after that draw another 2000 samples. The warmup samples are not saved. The initial points for each chain were picked above from the joint prior distribution.

[ ]:
```
# sampling
fit = post.sample(num_chains=4, num_samples=2000, init=[{'theta':
 ↪init_points[0]},
       {'theta':init_points[1]},{'theta':init_points[2]},{'theta':
 ↪init_points[3]}],
       num_warmup=2000)
# save the results to dataframe
df = fit.to_frame()
```

Next we print the summary of the draws. From there we can see that the $\hat{R}$ values are

$$\hat{R}_\alpha = 1.0 \quad \text{and} \quad \hat{R}_\beta = 1.0$$

suggesting that the chains have converged well. The method for computing the $\hat{R}$, is the rank normalized R-hat diagnostic from the article Vehtari et al. that is the default in ArviZ.

As discussed also on the previous assignment, the $\hat{R}$ value measures the potential scale reduction in the chains by comparing the within- and between-sequence vsriances. It is decreasing to 1 as the number of simulations $n \to \infty$. Therefore the closer to 1 the better, and we see that there is not much potential scale reduction left after the used warmup.

Note that we when writing the model, we used vector $\theta = [\theta[0], \theta[1]] = [\alpha, \beta]$, so below we have theta[0] $= \alpha$ and theta[1] $= \beta$.

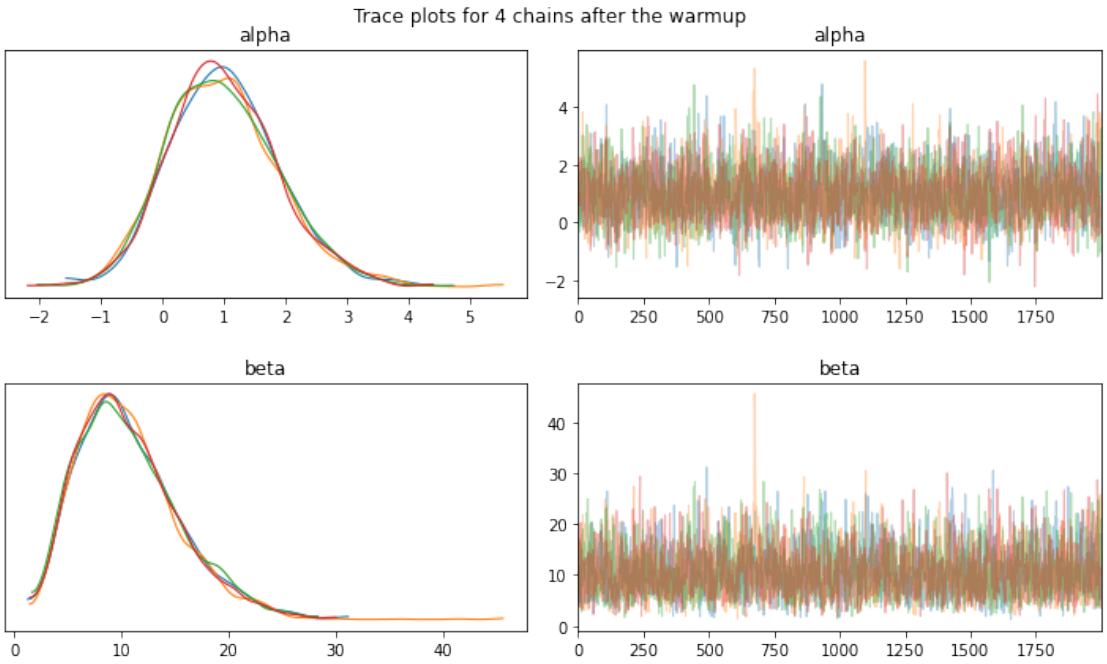
[ ]:
```
az.summary(fit)
```

[ ]:
```
            mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
theta[0]   0.996  0.904  -0.612    2.768      0.018    0.014    2528.0
theta[1]  10.657  4.751   2.564   19.385      0.102    0.076    2404.0

          ess_tail  r_hat
theta[0]    3155.0    1.0
theta[1]    2527.0    1.0
```

Then we plot the traces for the chains. The ArviZ plot includes also the distributions as default so we show them also.

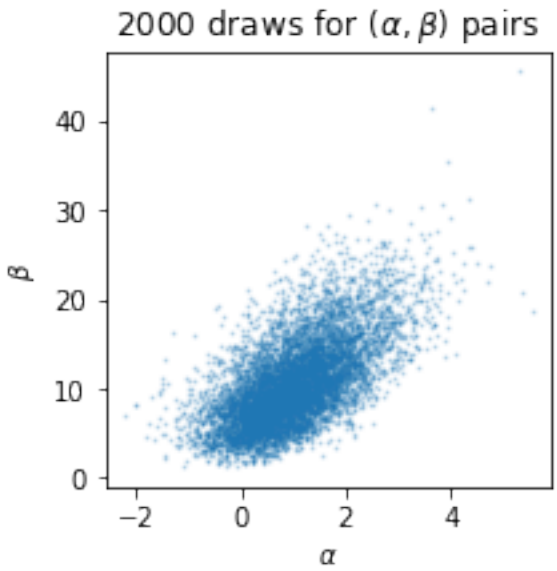
From the trace plots we can see that for both $\alpha$ and $\beta$ the chains are sampling from the same distributions, suggesting that the chains have converged nicely.

[ ]:
```
fig = az.plot_trace(fit, compact=False, figsize=(10,6))
plt.tight_layout()
plt.suptitle('Trace plots for 4 chains after the warmup')
fig[0][0].set_title('alpha')
fig[0][1].set_title('alpha')
fig[1][0].set_title('beta')
fig[1][1].set_title('beta')

plt.show()
```




Finally we plot the scatter plot for 2000 draws of $\alpha$ and $\beta$.

[ ]:
```
fig,ax = plt.subplots(figsize=(3,3))

ax.scatter(df['theta.1'].values.flatten(), df['theta.2'].values.flatten(), s=1,␣
 ↪alpha=0.2)
ax.set_ylabel(r'$\beta$')
ax.set_xlabel(r'$\alpha$')
ax.set_title(r'2000 draws for ($\alpha,\beta$) pairs')

plt.show()
```




## System configuration

- I used M1 powered MacBook Pro with macOS 12.6
- Programming environment was Python 3.9 with Visual Studio Code
- For Stan interface I used PyStan 3.5.0
- The installation of PyStan was a bit frustrating process as I had to install httpstan from source (because of M1). I hadn't done source installation before so it took a bit of time to figure out what that actually meant. After that things seemed to work rather straightforwardly (with some googling). If it's of interest, I also report how to actually install PyStan for M1 mac:
  - httpstan from source: clone the git repo to a local folder and then follow the instructions in https://httpstan.readthedocs.io/en/latest/installation.html
  - pip install PyStan as usual (wihtout the step above, an old version of PyStan 2 is installed)
  - in order to get PyStan work in VS Code, need to pip install nest-asyncio, import it and run nest_asyncio.apply() in some cell
- It took some time to figure out how to write the Stan code and how arrays, vectors, ints, reals etc. work. Some more examples would have been nice to get things rolling faster.