Amplicon data of two different marker genes from the same samples

GPU functionality

- CUDA/NVDA GPU was loaded and functions
- Mac ARM/metal GPU device is functional

Bayesian distributions

- Hamiltonian/MCMC/NUTS functionality working
- SVI (Machine learning) functionality working

Next

- Current model works for 16s PCs (6) and only 1 18s PC. Predicting a matrix looks straight forward for SVI/Machine learning Bayesian.

Import all the dependencies

In [1]:
```
import os
import pandas as pd
import numpy as np
import subprocess
```

In [2]:
```
import logging
import os

import torch
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from torch.distributions import constraints

import pyro
import pyro.distributions as dist
import pyro.optim as optim
```

```
pyro.set_rng_seed(1)
assert pyro.__version__.startswith('1.8.4')
```

In [3]:
```
%matplotlib inline
plt.style.use('default')

logging.basicConfig(format='%(message)s', level=logging.INFO)
smoke_test = ('CI' in os.environ)
pyro.set_rng_seed(1)
```

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
#Test for cuda devise
torch.cuda.get_device_name(0)
```

Out[5]:
```
'Quadro T2000'
```

Move to working directory

In [6]:
```
os.chdir("C:/Users/Kimani.Kimbrough/MarineDNA/Data")
```

01. This section imports the amplicon data sets as raw counts and calls an R script to model the ASV occurrences as probability distributions.

Import amplicon data sheets as pandas dataframes and take a look

In [7]:
```
file1 = "Flyer2018_16S_table_counts.tsv"
file2 = "Flyer2018_18S_table_counts.tsv"
asvs1 = pd.read_csv(file1, index_col=0, sep="\t")
asvs2 = pd.read_csv(file2, index_col=0, sep="\t")
```

Run the function in a loop over both amplicon data sets and make a list of two data frames

02. Reduce dimensionality and visualize principal components

In [8]:
```
from sklearn.decomposition import PCA
import seaborn as sns
```

From untransformed matrices

In [9]:
```
df_16S = pd.read_csv('Flyer2018_16S_counts_modeled.tsv', index_col=0,
sep="\t")
df_18S = pd.read_csv('Flyer2018_18S_counts_modeled.tsv', index_col=0,
sep="\t")
```

In [10]:
```
df_16S_logodds = pd.read_csv('Flyer2018_16S_counts_modeled_logodds.tsv',
index_col=0, sep="\t")
df_18S_logodds = pd.read_csv('Flyer2018_18S_counts_modeled_logodds.tsv',
index_col=0, sep="\t")
```

In [11]:
```
df_18S_logodds.head(3)
```

Out[11]:

	ASV_1	ASV_2	ASV_3	ASV_4	ASV_5	ASV_6	ASV_7	A:
CN18Fc12_8_eDNA	-10.756859	-10.593197	-8.643773	-8.788970	-8.055674	-5.545171	-4.441849	-2.56
CN18Fc19_5_eDNA	-9.610935	-10.697727	-9.834517	-1.237623	-8.055740	-5.750287	-5.297517	-3.49
CN18Fc21_6_eDNA	-9.526299	-10.299450	-7.894128	-9.666646	-7.648044	-5.544435	-4.821070	-2.90

3 rows × 7385 columns

From untransformed matrices

In [12]:
```
# Untransformed 16S
pca = PCA(n_components=62)
pca.fit_transform(df_16S)
variance_array_16S = np.cumsum(pca.explained_variance_ratio_ * 100)
#variance_array_16S
```

In [13]:
```
# Untransformed 18S
pca = PCA(n_components=62)
pca.fit_transform(df_18S)
variance_array_18S = np.cumsum(pca.explained_variance_ratio_ * 100)
#variance_array_18S
```

From log_odds transformed matrices

In [14]:
```
# Log-odds transformed 16S
pca = PCA(n_components=62)
pca.fit_transform(df_16S_logodds)
variance_array_16S_logodds = np.cumsum(pca.explained_variance_ratio_ *
100)
#variance_array_16S_logodds
```

In [15]:
```
# Log-odds transformed 18S
pca = PCA(n_components=62)
pca.fit_transform(df_18S_logodds)
variance_array_18S_logodds = np.cumsum(pca.explained_variance_ratio_ *
100)
#variance_array_18S_logodds
```

Plot components vs variance explained

Based on the results above we will use the variances generated from the raw (untransformed) count probabilities

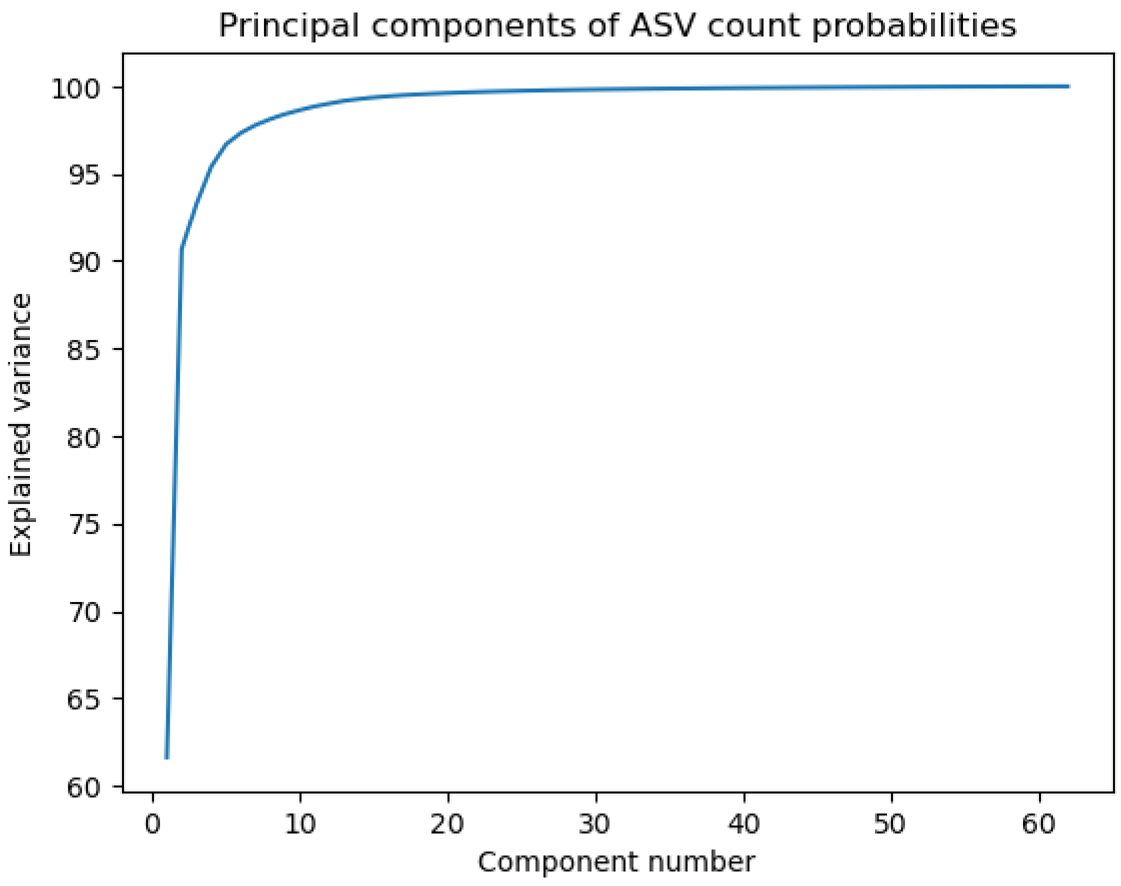
Function to format variance numpy array for seaborn plot

In [16]:
```
def format_variance_data_for_plotting(variance_array):
    df = pd.DataFrame(variance_array, columns = ["Explained variance"])
    df = df.reset_index(level=0)
    df['index'] = df['index'] + 1
    df = df.rename(columns = {"index" : "Component number"})
    return(df)
```

Apply function to raw count probabilities

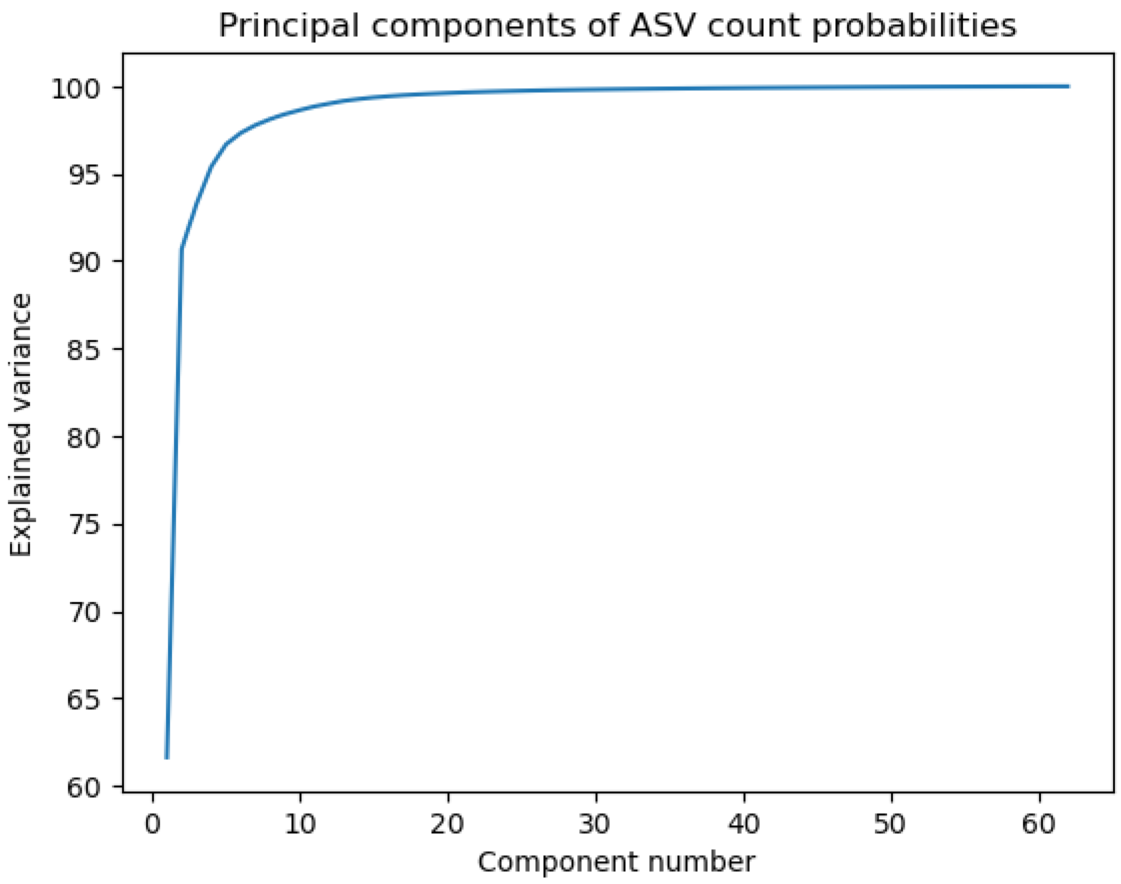
In [17]:
```
# 16S count data
df_variance = format_variance_data_for_plotting(variance_array_16S)
sns.lineplot(data=df_variance, x="Component number",
             y="Explained variance").set(title='Principal components of
ASV count probabilities')
```

Out[17]:
```
[Text(0.5, 1.0, 'Principal components of ASV count probabilities')]
```



In [18]:
```
# 18S count data
df_variance = format_variance_data_for_plotting(variance_array_16S)
sns.lineplot(data=df_variance, x="Component number",
             y="Explained variance").set(title='Principal components of
ASV count probabilities')
```

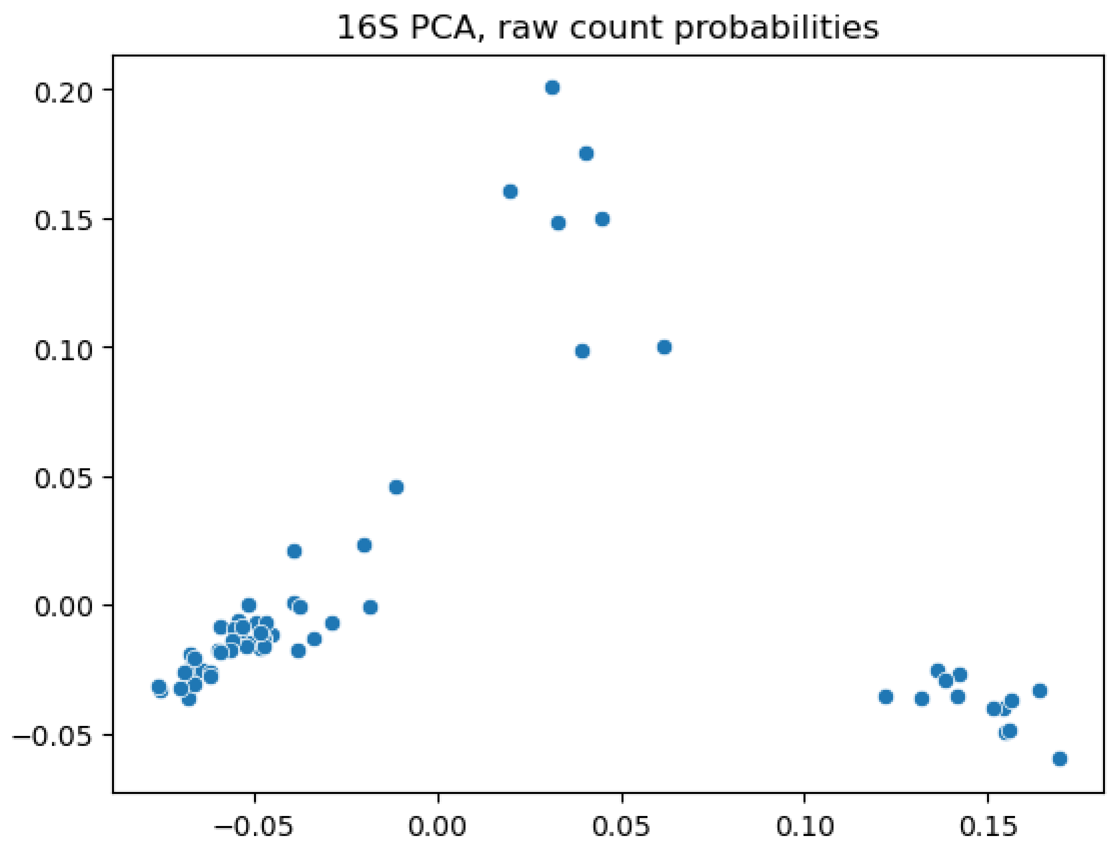
Out[18]:
```
[Text(0.5, 1.0, 'Principal components of ASV count probabilities')]
```



Plot principal components of raw ASV count probabilities

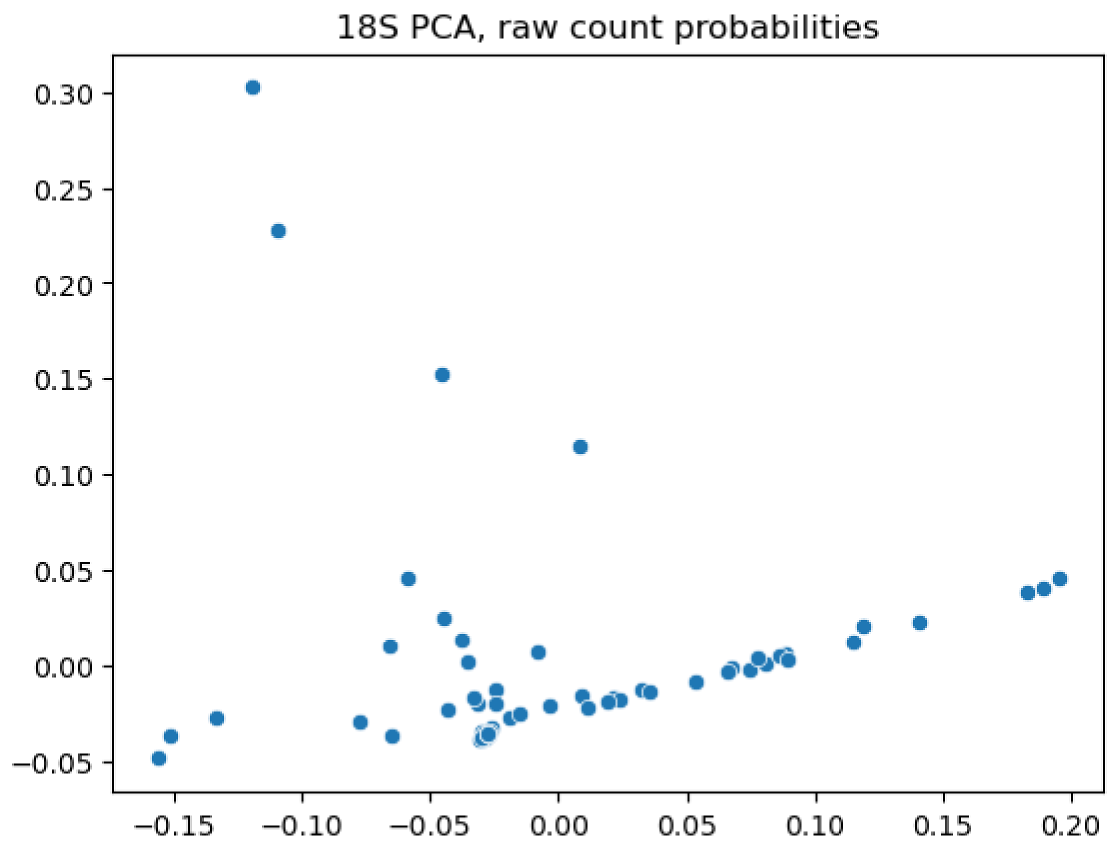
In [19]:
```
pca_16 = PCA(n_components=3)
pcs_16 = pca.fit_transform(df_16S)

pc1_values_16 = pcs_16[:,0]
pc2_values_16 = pcs_16[:,1]
sns.scatterplot(x=pc1_values_16, y=pc2_values_16).set(title="16S PCA, raw
count probabilities")
```

Out[19]:
```
[Text(0.5, 1.0, '16S PCA, raw count probabilities')]
```



In [20]:
```
pca_118 = PCA(n_components=3)
pcs_18 = pca.fit_transform(df_18S)

pc1_values_18 = pcs_18[:,0]
pc2_values_18 = pcs_18[:,1]
sns.scatterplot(x=pc1_values_18, y=pc2_values_18).set(title="18S PCA, raw
count probabilities")
```

Out[20]:
```
[Text(0.5, 1.0, '18S PCA, raw count probabilities')]
```



03. Test the power of 16S data as a predictor for 18S data

Function to extract defined number of PCs with sample labels

In [21]:
```
def extract_PCs_labeled(df_asvs_modeled, num_pcs):
    pca = PCA(n_components=num_pcs)
    pcs = pca.fit_transform(df_asvs_modeled)
    array = pcs[:, :num_pcs]
    cols = list()
    for i in range(1, num_pcs+1):
        n="PC%s" % i
        cols.append(n)
    df = pd.DataFrame(array, index=df_16S.index, columns = cols)
    return(df)
```

Export the first six PCs of the 16S data which explain 97% of the variance

In [22]:
```
pcs_16S = extract_PCs_labeled(df_16S, 6)
pcs_16S.to_csv("Flyer2018_16S_PCs.tsv", sep="\t")
```

```
pcs_16S
```

Out[22]:

	PC1	PC2	PC3	PC4	PC5	PC6
CN18Fc12_8_eDNA	-0.020128	0.023832	-0.024139	0.038050	-0.016783	0.001938
CN18Fc19_5_eDNA	-0.075552	-0.033142	0.010200	-0.011909	0.003156	-0.002386
CN18Fc21_6_eDNA	-0.048505	-0.016760	-0.003077	0.003165	-0.006532	0.004679
CN18Fc22_6_eDNA	-0.066767	-0.022830	0.005425	-0.001503	-0.004624	0.003922
CN18Fc24_6_eDNA	-0.059178	-0.008103	-0.005169	0.006752	-0.001524	0.001449
...	...	...	...	...	...	...
CN18SESPkoa_SC42	0.131701	-0.035945	0.003022	-0.002341	-0.002827	0.002591
CN18SESPkoa_SC44	0.032810	0.148524	-0.005046	0.012712	-0.028669	-0.015435
CN18SESPkoa_SC45	0.141820	-0.035358	0.004186	0.004716	-0.003955	0.000039
CN18SESPkoa_SC47	0.136522	-0.025320	-0.000650	0.013615	-0.006447	-0.002134
CN18SESPkoa_SC49	0.138331	-0.028862	0.002282	0.010463	-0.006027	-0.001998

62 rows × 6 columns

Export the first two PCs of the 18S data for which we will test the 16S predictive power

In [23]:

```
pcs_18S = extract_PCs_labeled(df_18S, 2)
pcs_18S.to_csv("Flyer2018_18S_PCs.tsv", sep="\t")
pcs_18S
```

Out[23]:

	PC1	PC2
CN18Fc12_8_eDNA	-0.018686	-0.027522
CN18Fc19_5_eDNA	-0.133394	-0.027902
CN18Fc21_6_eDNA	-0.031460	-0.019696
CN18Fc22_6_eDNA	-0.042921	-0.023416
CN18Fc24_6_eDNA	-0.151135	-0.036783
...	...	...
CN18SESPkoa_SC42	-0.026949	-0.034856
CN18SESPkoa_SC44	-0.029892	-0.037498
CN18SESPkoa_SC45	-0.027413	-0.035936
CN18SESPkoa_SC47	-0.027500	-0.036121
CN18SESPkoa_SC49	-0.027284	-0.035349

62 rows × 2 columns

Pass results of 16S and 18S PCAs to Bayesian modeling R Script

Pyro Pytorch linear model

Define variables based on PCA results from above; can only pass strings to R script

In [24]:
```
path_to_rscript =
"/Users/nastassia.patin/GitHub/MarineDNA/PC_bayesian_runner_test.R"
num_18S_PCs = "2" # Number of PCs to predict in 18S data
num_16S_preds = "6" # Number of predictor PCs to use from 16S data; as
many as account for expected variance of predictor
num_ind = "62" # Number of observances (samples) in 18S data
```

Run the function and display summary output

In [25]:
```
#bayes_summary_16S_18S =
bayesian_modeling_of_two_markergenes(path_to_rscript,
#                                                              num_18S_PCs,
#
num_16S_preds,
#                                                              num_ind)
#bayes_summary_16S_18S
```

In [26]:
```
print(pcs_16S.columns.values)
```

```
['PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6']
```

In [27]:
```
df1 = pcs_16S[['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6']]
df1.columns = ['pc1_values_16', 'pc2_values_16','pc3_values_16',
'pc4_values_16', 'pc5_values_16', 'pc6_values_16']
df2 = pcs_18S[['PC1']]
df2.columns = ['pc1_values_18']
df = pd.merge(df1, df2, left_index=True, right_index=True)
```

In [29]:
```
def model(pc1_values_16, pc2_values_16, pc3_values_16, pc4_values_16,
pc5_values_16, pc6_values_16, pc1_values_18):
    a = pyro.sample("a", dist.Normal(0., 100000.))
    b_PC1 = pyro.sample("b1", dist.Normal(0., 10000.))
    b_PC2 = pyro.sample("b2", dist.Normal(0., 10000.))
    b_PC3 = pyro.sample("b3", dist.Normal(0., 10000.))
    b_PC4 = pyro.sample("b4", dist.Normal(0., 10000.))
    b_PC5 = pyro.sample("b5", dist.Normal(0., 10000.))
    b_PC6 = pyro.sample("b6", dist.Normal(0., 10000.))
    sigma = pyro.sample("sigma", dist.Uniform(0., 10000.))

    mean = a + b_PC1 * pc1_values_16 + b_PC2 * pc2_values_16 + b_PC3 *
pc3_values_16 + b_PC4 * pc4_values_16 + b_PC5 * pc5_values_16 + b_PC6 *
pc6_values_16

    with pyro.plate("data", len(pc6_values_16)):
        return pyro.sample("obs", dist.Normal(mean, sigma),
obs=pc1_values_18)
```

In [40]:

```
from pyro.infer.autoguide import AutoMultivariateNormal, init_to_mean



guide = AutoMultivariateNormal(model, init_loc_fn=init_to_mean)

svi = SVI(model,
          guide,
          optim.Adam({"lr": .01}),
          loss=Trace_ELBO())

pc1_values_16, pc2_values_16, pc3_values_16, pc4_values_16, pc5_values_16,
pc6_values_16, pc1_values_18 = train[:, 0], train[:, 1], train[:, 2],
train[:, 3], train[:, 4], train[:, 5], train[:, 6]
pyro.clear_param_store()
for i in range(num_iters):
    elbo = svi.step(pc1_values_16, pc2_values_16, pc3_values_16,
pc4_values_16, pc5_values_16, pc6_values_16, pc1_values_18)
    if i % 500 == 0:
        logging.info("Elbo loss: {}".format(elbo))
```

```
Elbo loss: 663.874231338501
Elbo loss: 337.2063698768616
Elbo loss: 71.16333341598511
Elbo loss: 9.3086519241333
Elbo loss: 9.785514831542969
Elbo loss: 10.661048889160156
Elbo loss: 8.096829414367676
Elbo loss: 11.31319808959961
Elbo loss: 12.004347801208496
Elbo loss: 10.443873405456543
```

In [31]:

```
# Utility function to print latent sites' quantile information.
def summary(samples):
    site_stats = {}
    for site_name, values in samples.items():
        marginal_site = pd.DataFrame(values)
        describe = marginal_site.describe(percentiles=[.05, 0.25, 0.5,
0.75, 0.95]).transpose()
        site_stats[site_name] = describe[["mean", "std", "5%", "25%",
"50%", "75%", "95%"]]
    return site_stats
```

In [32]:

```
# Prepare training data
df4 = df[['pc1_values_16', 'pc2_values_16','pc3_values_16',
'pc4_values_16', 'pc5_values_16', 'pc6_values_16', 'pc1_values_18']]
df5 = df4[np.isfinite(df.pc1_values_18)]
train = torch.tensor(df5.values, dtype=torch.float)
```

In [33]:

```
from pyro.infer import SVI, Trace_ELBO



svi = SVI(model,
          guide,
          optim.Adam({"lr": .05}),
          loss=Trace_ELBO())

pc1_values_16, pc2_values_16, pc3_values_16, pc4_values_16, pc5_values_16,
pc6_values_16, pc1_values_18 = train[:, 0], train[:, 1], train[:, 2],
train[:, 3], train[:, 4], train[:, 5], train[:, 6]
pyro.clear_param_store()
num_iters = 5000 if not smoke_test else 2
```

```
for i in range(num_iters):
    elbo = svi.step(pc1_values_16, pc2_values_16, pc3_values_16,
pc4_values_16, pc5_values_16, pc6_values_16, pc1_values_18)
    if i % 500 == 0:
        logging.info("Elbo loss: {}".format(elbo))
```

```
Elbo loss: 670.0780696868896
Elbo loss: 13.34172248840332
Elbo loss: 17.20850944519043
Elbo loss: 9.504776954650879
Elbo loss: 9.566930770874023
Elbo loss: 10.968674659729004
Elbo loss: 11.626540184020996
Elbo loss: 9.353282928466797
Elbo loss: 18.191561698913574
Elbo loss: 13.249004364013672
```

In [34]:

```
# Prepare training data
train = torch.tensor(df.values, dtype=torch.float)
#train
```

In [35]:

```
from pyro.infer import Predictive



num_samples = 1000
predictive = Predictive(model, guide=guide, num_samples=num_samples)
svi_samples = {k: v.reshape(num_samples).detach().cpu().numpy()
               for k, v in predictive(pc1_values_16, pc2_values_16,
pc3_values_16, pc4_values_16, pc5_values_16, pc6_values_16,
pc1_values_18).items()
               if k != "obs"}
```

In [36]:

```
from pyro.infer import MCMC, NUTS



nuts_kernel = NUTS(model)

mcmc = MCMC(nuts_kernel, num_samples=1000, warmup_steps=200)
mcmc.run(pc1_values_16, pc2_values_16, pc3_values_16, pc4_values_16,
pc5_values_16, pc6_values_16, pc1_values_18)
```

```
hmc_samples = {k: v.detach().cpu().numpy() for k, v in
mcmc.get_samples().items()}
```

```
Sample: 100%|██████████| 1200/1200 [01:21, 14.81it/s, step size=5.97e-01, acc. prob=
0.903]
```

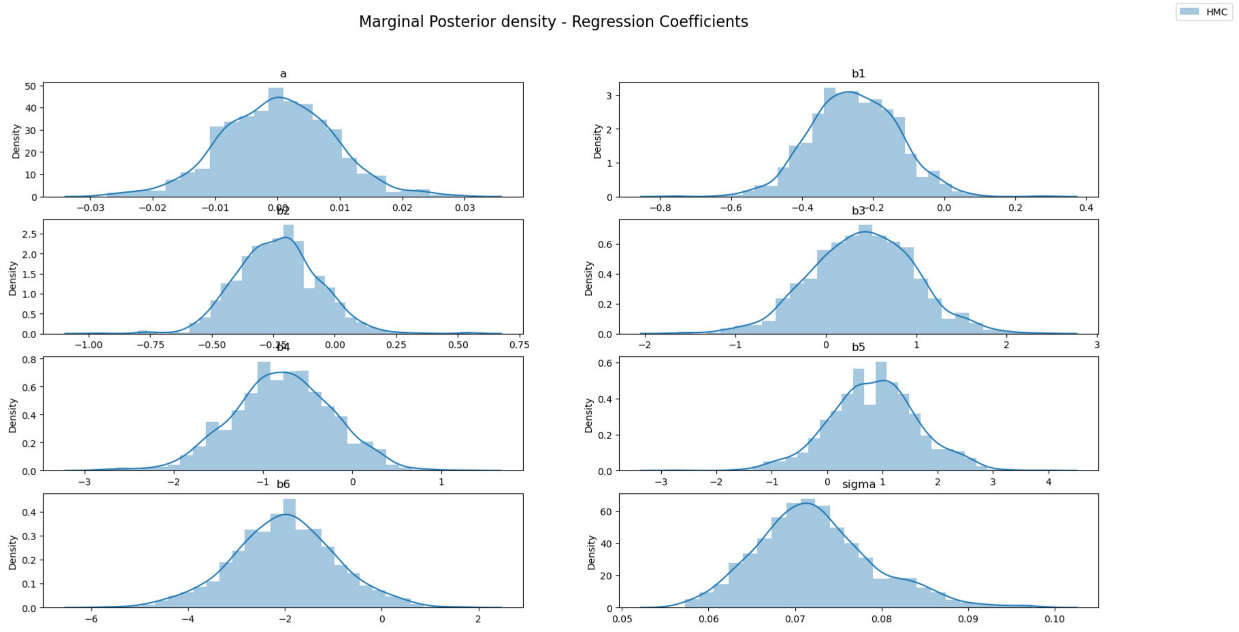
In [37]:

```
for site, values in summary(hmc_samples).items():
    print("Site: {}".format(site))
    print(values, "\n")
```

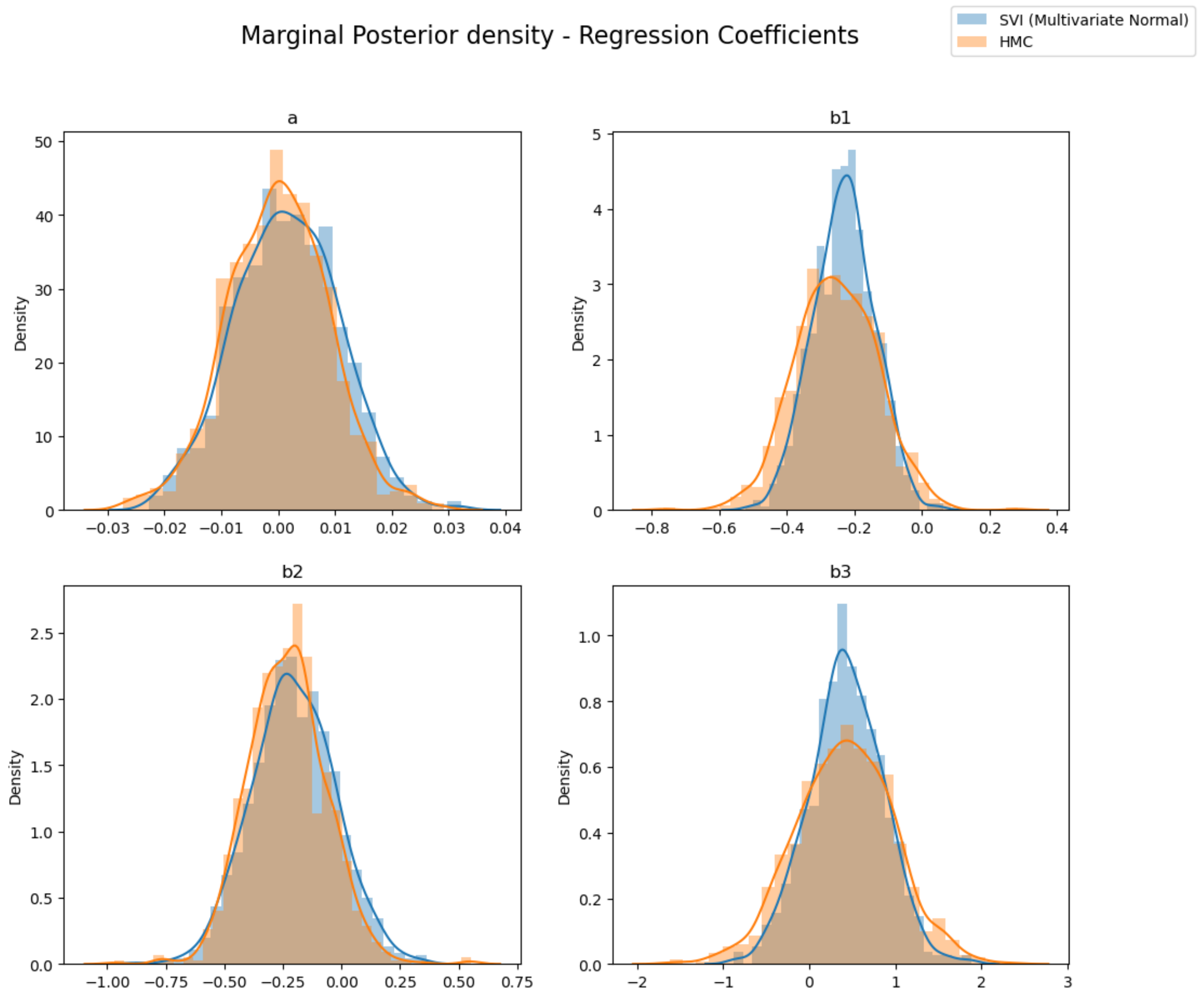
```
Site: a
       mean       std        5%       25%       50%       75%       95%
0 -0.000008  0.008881 -0.014398 -0.006117  0.000015  0.005905  0.014473

Site: b1
       mean       std        5%       25%       50%       75%       95%
0 -0.253389  0.120913 -0.442184 -0.335472 -0.257342 -0.168988 -0.050668

Site: b2
       mean      std        5%       25%       50%       75%       95%
0 -0.231491  0.16589 -0.492472 -0.343375 -0.230323 -0.128819  0.027552

Site: b3
       mean       std       5%       25%       50%       75%       95%
0  0.410288  0.565431 -0.48369  0.025277  0.411468  0.798516  1.316391

Site: b4
       mean      std        5%      25%       50%       75%       95%
0 -0.780196  0.56178 -1.684114 -1.13819 -0.772413 -0.397522  0.145287

Site: b5
       mean       std        5%       25%       50%       75%       95%
0  0.846589  0.802281 -0.473396  0.333958  0.884097  1.346998  2.224882

Site: b6
       mean       std        5%       25%       50%       75%       95%
0 -2.010562  1.069957 -3.802254 -2.692759 -1.999521 -1.322854 -0.195606

Site: sigma
       mean       std        5%       25%       50%       75%       95%
0  0.072411  0.006784  0.062412  0.067775  0.071762  0.076364  0.084708
```

In [38]:

```
sites = ["a", "b1", "b2", "b3", "b4", "b5", "b6","sigma"]

fig, axs = plt.subplots(nrows=4, ncols=2, figsize=(20, 10))
fig.suptitle("Marginal Posterior density - Regression Coefficients",
fontsize=16)
for i, ax in enumerate(axs.reshape(-1)):
    site = sites[i]
    #sns.distplot(svi_samples[site], ax=ax, label="SVI (DiagNormal)")
    sns.distplot(hmc_samples[site], ax=ax, label="HMC")
    ax.set_title(site)
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='upper right');
```



In [39]:
```
predictive = Predictive(model, guide=guide, num_samples=num_samples)
svi_mvn_samples = {k: v.reshape(num_samples).detach().cpu().numpy()
                   for k, v in predictive(pc1_values_18, pc1_values_16,
pc2_values_16, pc3_values_16, pc4_values_16, pc5_values_16,
pc6_values_16).items()
                   if k != "obs"}
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
fig.suptitle("Marginal Posterior density - Regression Coefficients",
fontsize=16)
for i, ax in enumerate(axs.reshape(-1)):
    site = sites[i]
    sns.distplot(svi_mvn_samples[site], ax=ax, label="SVI (Multivariate
Normal)")
    sns.distplot(hmc_samples[site], ax=ax, label="HMC")
    ax.set_title(site)
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='upper right');
```


Marginal Posterior density - Regression Coefficients
SVI (Multivariate Normal)
HMC
a
b1
b2
b3
Density

```
In [ ]:
```

```
In [ ]:
```