PMR - Coursework Two

S1424164, S1897273

March 2019

3 Part (a)

```
def mh(p_star, param_init, num_samples=5000, stepsize=1.0):
    points = [np.array(param_init)]
    variance = [stepsize for i in range(0, len(param_init))]

    for i in range(1, num_samples):
        x_l_minus_1 = points[i-1]
        x_cand = np.random.normal(x_l_minus_1, variance)
        a = p_star(x_cand)/p_star(x_l_minus_1)
        if a >= 1.0: # If the candidiate is a better guess then just add it
            points.append(x_cand)
        else: # Randomly sample to see if it at least has a decent prob dist.
            u = random.uniform(0.0,1.0)
            if u < a:
                points.append(x_cand)
            else: # Just probabilistically take the same point again.
                points.append(x_l_minus_1)

    return points
```

Part (b)

```
def mh(p_star, param_init, warmup, num_samples=5000, stepsize=1.0):
    points = [np.array(param_init)]
    variance = [stepsize for i in range(0, len(param_init))]

    for i in range(1, num_samples):
        x_l_minus_1 = points[i-1]
        x_cand = np.random.normal(x_l_minus_1, variance)
        a = p_star(x_cand)/p_star(x_l_minus_1)
        if a >= 1.0: # If the candidiate is a better guess then just add it
            points.append(x_cand)
        else: # Randomly sample to see if it at least has a decent prob dist.
            u = random.uniform(0.0,1.0)
            if u < a:
                points.append(x_cand)
            else: # Just probabilistically take the same point again.
                points.append(x_l_minus_1)

    return points[warmup:]
```

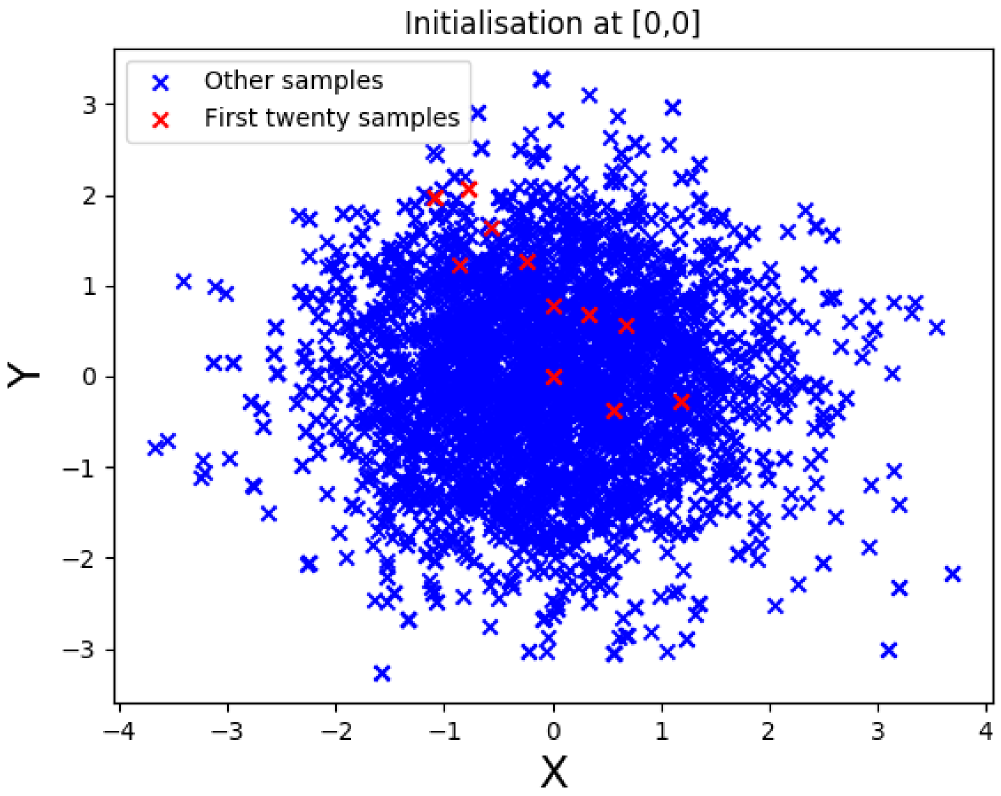

Figure 1: This shows part b) with an initialisation at (0,0). This shows correctly that the peak of the posterior is a Gaussian at (0,0), because the multiplication of two Gaussians is also a Gaussian, with a variance of $\frac{1}{2}$ and a mean also of zero.

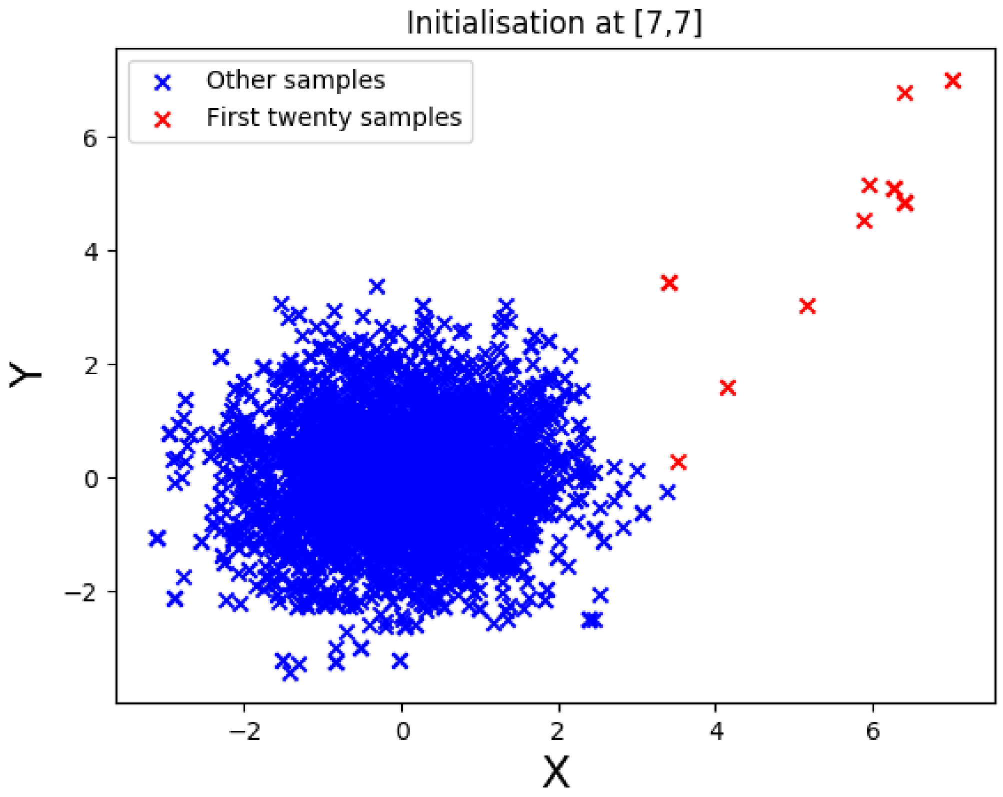

Figure 2: This shows part b) for an initialisation at (7,7). This also shows the same posterior as the last example, yet this time, the burn in samples gradually drift from (7,7) as expected, towards the areas of highest density of the posterior.

Part (c)

Derivation of the result:

$$p(\alpha, \beta|D) \propto p(D|\alpha, \beta)p(\alpha, \beta)$$

From the fact that alpha and beta are sampled independently.

$$p(\alpha, \beta|D) \propto p(D|\alpha, \beta)p(\alpha)p(\beta)$$

From the fact that the likelihood is Poisson.

$$p(\alpha, \beta|D) \propto poisson(D|\alpha, \beta)p(\alpha)p(\beta)$$

From the definition of poisson sampling. In addition, because the posterior over α and β is the quantity of interest, dependencies over the variables x can be ignored, because they are treated as a constant when dealing with proportionality.

$$p(\alpha, \beta|D) \propto \prod_{i}^{N} poisson(y_i|exp(\alpha x_i + \beta))p(\alpha)p(\beta)$$

Alpha mean	0.899
Beta mean	-0.220
Pearson Correlation	-0.672

```
# ellipsis indicates more data excluded for the sake of the report.
def posterior_alpha_beta(alpha_beta):
    xs = [-5.051905265552104618e-01 ...]
    ys = [1.000000000000000000e+00 ...]
```

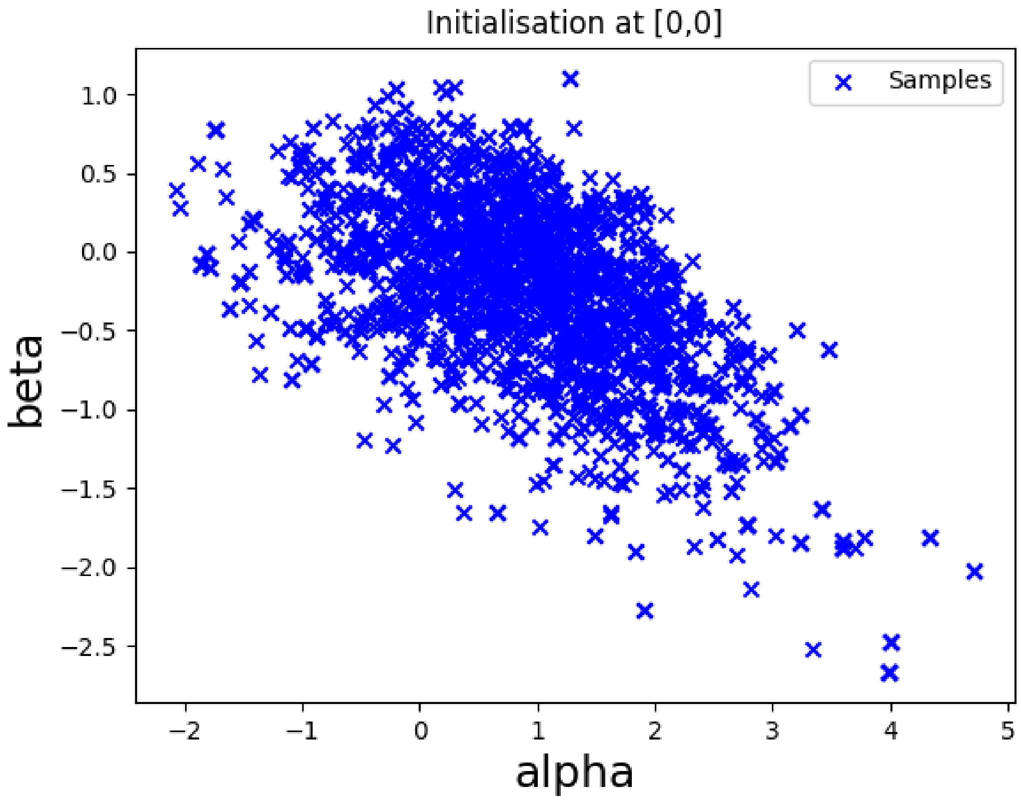

Figure 3: This shows part c), alpha versus beta

```
alpha = alpha_beta[0]
beta = alpha_beta[1]
prior_term = scipy.stats.norm(0, 100).pdf(alpha) * scipy.stats.norm(0, 100).pdf(beta)
mus = [np.exp(alpha*x+beta) for x in xs]
likelihoods = scipy.stats.poisson.pmf(ys, mus)
return product(likelihoods)
```

Part (d)

This analysis assumes a Gaussian proposal distribution and a Poisson target distribution.

Figure 2b) has a smaller stepsize than Figure 2a).
Explanation: One would expect that a Gaussian proposal distribution with a much smaller variance than the target distribution implies a small, but non zero, update at each timestep, given a reasonably smooth target distribution. The variance quantifies how exploratory the MH algorithm is with respect to the proposal distribution. There is movement in the samples in both directions. This indicates that during the case where the proposal distribution chooses a less likely sample, the sample still has a moderately high probability of being accepted, which would be true if the variance of the proposal distribution is very small and the proposal distribution proposes points around a locally smooth window over the target distribution. In addition, the y-axis shows that the updates in Figure 2b) are smaller than those in 2a). Figure 2a) has a higher variance which is indicated by the rapid fluctuation in sample values at each time step.

Figure 2c) has a larger stepsize than Figure 2a).
Explanation: In 2c), the most important piece of evidence to support this notion are the presence of flat lines in the space. Given a proposal distribution with very high variance, proposed points will often be chosen very far away from the target distributions maximum. These points will have a probability that will tend to zero and thus these candidate points will rarely be chosen over the previous point. There are likely to be repeated instances of poor point choices, that oscillate very rapidly around the maximum of the target distribution, but rarely reach it. It is likely that there will be repeated occurrences of points with these low probabilities relative to the previous chosen point, given enough samples chosen. This means that the

candidate point will be repeatedly disregarded, resulting in flat lines. However, when points are chosen, the movement is often very large.

Part (e)

Let $\hat{\theta_1} = \theta_1 - \mathbb{E}[\theta_1]$
Let $\hat{\theta_2} = \theta_2 - \mathbb{E}[\theta_2]$
We also know that $\mathbb{E}[\hat{\theta_1}], \mathbb{E}[\hat{\theta_2}] = 0$ and that $\mathbb{V}ar[\hat{\theta_1}] = \mathbb{V}ar[\theta_1]$ and $\mathbb{V}ar[\hat{\theta_2}] = \mathbb{V}ar[\theta_2]$.

From the equivalence of the variances, and the fact that removing a term from inside a variance formula means we square it, we can see that:

$$\mathbb{V}ar[\frac{1}{2}(\theta_1 + \theta_2)] = \frac{1}{4}(\mathbb{V}ar[\hat{\theta_1} + \hat{\theta_2}])$$

$$= \frac{1}{4}(\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})^2] - \mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})]^2)$$

$$= \frac{1}{4}(\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})^2] - \mathbb{E}[\hat{\theta_1} + \hat{\theta_2}]\mathbb{E}[\hat{\theta_1} + \hat{\theta_2}])$$

$$= \frac{1}{4}(\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})^2] - (\mathbb{E}[\hat{\theta_1}] + \mathbb{E}[\hat{\theta_2}])\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})])$$

$$= \frac{1}{4}(\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})^2] - (0 + 0)\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})])$$

$$= \frac{1}{4}(\mathbb{E}[(\hat{\theta_1} + \hat{\theta_2})^2])$$

$$= \frac{1}{4}(\mathbb{E}[\hat{\theta_1}^2 + 2\hat{\theta_1}\hat{\theta_2} + \hat{\theta_2}^2])$$

$$= \frac{1}{4}(\mathbb{E}[\hat{\theta_1}^2] + \mathbb{E}[2\hat{\theta_1}\hat{\theta_2}] + \mathbb{E}[\hat{\theta_2}^2])$$

$$= \frac{1}{4}(\mathbb{V}ar[\hat{\theta_1}] + 2\mathbb{E}[(\theta_1 - \mathbb{E}[\theta_1])(\theta_2 - \mathbb{E}[\theta_2])] + \mathbb{V}ar[\hat{\theta_2}])$$

$$= \frac{1}{4}(\mathbb{V}ar[\hat{\theta_1}] + 2Cov(\theta_1, \theta_2) + \mathbb{V}ar[\hat{\theta_2}])$$

$$= \frac{1}{4}(2\sigma^2 + 2\sigma^2\rho)$$

$$= \frac{\sigma^2}{2}(1 + \rho)$$

So therefore,

$$\alpha = (1 + \rho)$$

Where the effective sample size reduces to one for correlated variables (i.e. $\rho \rightarrow 1$) and two in the case of uncorrelated variables (i.e. $\rho \rightarrow 0$).

4 Part (a)

```
data {
    int N;
    int y[N];
    real x[N];
}

parameters {
    real alpha;
    real beta;
}

model {
    alpha ~ normal(0, 100);
    beta ~ normal(0, 100);
    for (n in 1:N){
      y[n] ~ poisson(exp(alpha * x[n] + beta));
    }
}
```

As well as the python wrapper:

```
import matplotlib.pyplot as plt
import pystan
from pystan.external.pymc.plots import traceplot
import numpy as np
import scipy

# ellipsis indicates more data excluded for the sake of the report.
xs = [-5.051905265552104618e-01 ...]
ys = list(map(int, [1.000000000000000000e+00, 0.000000000000000000e+00 ...]))

sm = pystan.StanModel(file='q4.stan')

results = sm.sampling(data={"N": len(xs), "y": ys, "x": xs}, iter=10000, chains = 1)
alphas = results.extract()["alpha"]
betas = results.extract()["beta"]

fig = results.plot(['beta', 'alpha'])
plt.show()
plt.title("Initialisation at [0,0]")
plt.xlabel('alpha', fontsize=18)
plt.ylabel('beta', fontsize=18)
plt.scatter(alphas, betas, c='b', marker='x', label='Samples')
plt.legend()
plt.show()
print("Alpha mean: " + str(np.mean(alphas)))
print("Beta mean: " + str(np.mean(betas)))
print("Correlation: " + str(scipy.stats.pearsonr(alphas, betas)))
```

Alpha mean	0.889
Beta mean	-0.201
Pearson Correlation	-0.592

The results are almost identical to the results from 3c), which shows the utility of automatic inference engines. The trace plots are well mixed, unlike Figures 2b and 2c in Question 3. Figure 2a) is the only figure which demonstrates a well mixed trace plot. This is indicated by the spiky caterpillar like appearance of the trace plots, i.e. they have no discernible structure, similar to white noise.

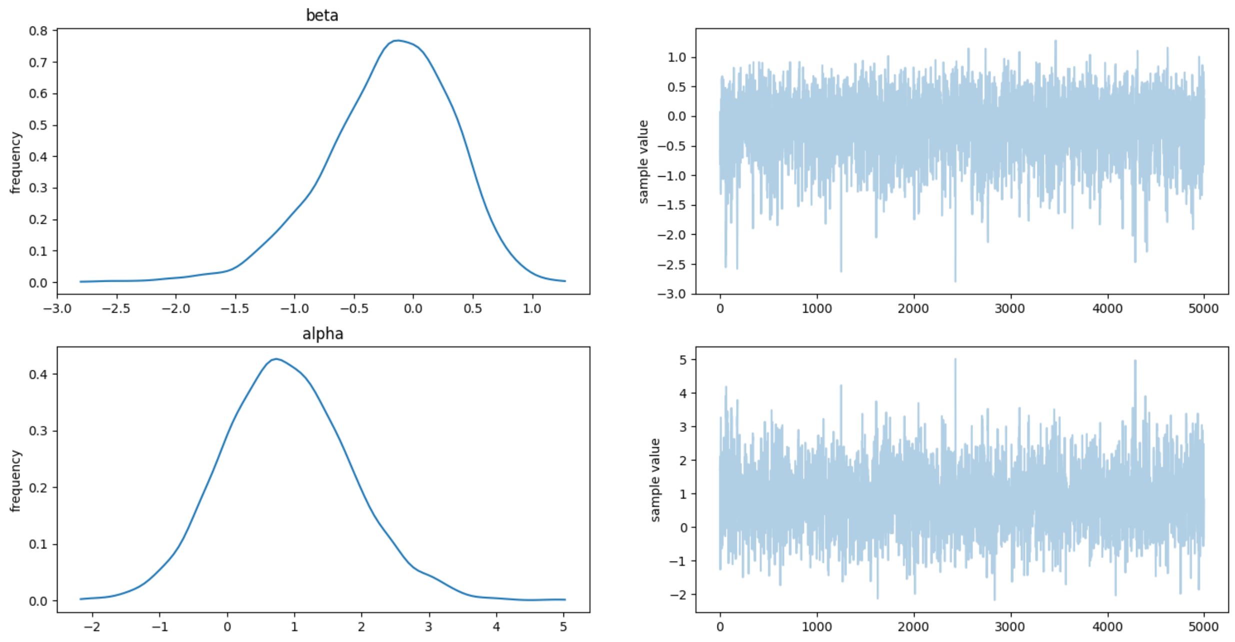

Figure 4: This shows the traceplots for both alpha and beta for 4 part a.

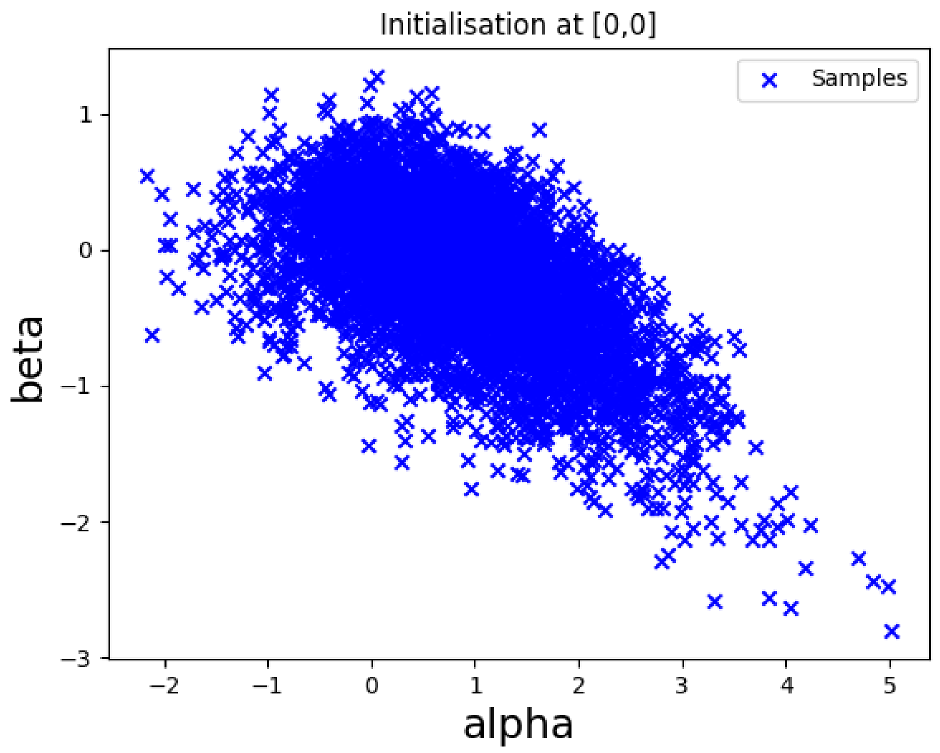

Figure 5: This shows the correlation between alpha and beta for 5,000 samples for 4 part a.

Part (b)

$$p(x_1,...,x_N,z_1,...,z_N,W,\tau,\alpha) = \Gamma(\tau;1,1)(\prod_{i=1}^{K}\Gamma(\alpha_i;\alpha_0,\beta_0))\delta\lambda(\prod_{r=1}^{N}(\prod_{c=1}^{D}(\mathcal{N}(X_{rc};\delta_r^T,T_{\alpha_c}))))$$

where

$$\lambda = \prod_{r=1}^{N}\prod_{c=1}^{K}\mathcal{N}(Z_{rc};0,1)$$

$$\delta = \prod_{c=1}^{K}\prod_{r=1}^{D}\mathcal{N}(W_{rc};\mu_w,T_\tau)$$

$$T_\alpha = \begin{bmatrix} 1/\sqrt{\alpha_1} \\ 1/\sqrt{\alpha_2} \\ \vdots \\ 1/\sqrt{\alpha_n} \end{bmatrix}$$

$$T_\tau = 1/\sqrt{\tau}$$

The hyper-parameters used in the equation are: $[\alpha_0, \beta_0, \mu_w]$

Part (c)

Figure 6 shows that standard PCA (with SVD as a subroutine) on the data matrix X, yields similarities with the principal eigenvectors obtained by Bayesian PCA. However, the results diverge on the second principal eigenvector. Here, standard PCA yields a vector in cyan, whereas Bayesian PCA yields the magenta vectors, though on examination, we found that the third principal eigenvector for standard PCA does agree with the second and third principal eigenvectors for Bayesian PCA. The first, second and third principal eigenvectors are denoted by #1,#2,#3 respectively.

25 Samples of rescaled eigenvectors

$u_1, u_2 = [1.2, 0.78, -0.11], [0.15, -0.22, 0.06]$
$u_1, u_2 = [1.21, 0.79, 0.25], [-0.02, -0.07, 0.29]$
$u_1, u_2 = [0.89, 0.85, -0.04], [0.31, -0.32, 0.12]$
$u_1, u_2 = [1.26, 0.44, -0.09], [0.17, -0.45, 0.2]$
$u_1, u_2 = [-1.36, -0.57, 0.39], [-0.19, 0.37, -0.12]$
$u_1, u_2 = [-1.33, -0.67, -0.62], [-0.18, 0.18, 0.18]$
$u_1, u_2 = [1.12, 0.7, -0.0], [0.18, -0.29, -0.04]$
$u_1, u_2 = [-1.03, -0.49, 0.36], [-0.01, 0.21, 0.26]$
$u_1, u_2 = [0.98, 0.56, 0.04], [0.14, -0.24, 0.02]$
$u_1, u_2 = [-1.3, -0.68, -0.67], [-0.22, 0.18, 0.25]$
$u_1, u_2 = [-1.26, -0.38, -0.21], [-0.17, 0.45, 0.2]$
$u_1, u_2 = [1.28, 0.62, -0.0], [0.02, -0.03, 0.34]$
$u_1, u_2 = [-1.11, -0.53, -0.26], [0.0, -0.19, 0.37]$
$u_1, u_2 = [-1.18, -0.73, 0.37], [-0.19, 0.23, -0.13]$
$u_1, u_2 = [-1.2, -0.64, -0.63], [-0.23, 0.22, 0.21]$
$u_1, u_2 = [1.16, 0.68, -0.01], [0.19, -0.32, 0.13]$
$u_1, u_2 = [-1.36, -0.62, -0.5], [0.01, -0.23, 0.27]$
$u_1, u_2 = [-1.1, -0.53, 0.49], [0.0, 0.21, 0.24]$
$u_1, u_2 = [-0.7, -0.63, 0.48], [-0.01, 0.25, 0.31]$
$u_1, u_2 = [-1.12, -0.68, -0.52], [0.02, -0.21, 0.24]$
$u_1, u_2 = [1.52, 0.68, 0.15], [0.04, -0.2, 0.46]$
$u_1, u_2 = [-1.19, -0.38, 0.23], [-0.02, 0.2, 0.24]$
$u_1, u_2 = [0.97, 0.64, 0.1], [-0.0, -0.05, 0.36]$
$u_1, u_2 = [-1.01, -0.52, 0.36], [-0.02, 0.19, 0.2]$
$u_1, u_2 = [-1.11, -0.6, 0.4], [-0.19, 0.28, -0.11]$

```
import matplotlib.pyplot as plt
import pystan
from pystan.external.pymc.plots import traceplot
import numpy as np
import scipy
import random
import math
import heapq

X = np.array([[6.950646740646656552e-01, -4.859325904038511168e-01, 1.199665310159269360e+00,
-1.500564639503813247e+00, 1.046589595792177585e+00, -3.919275793143054409e-01,
6.791836476729603556e-01, 5.237820161229895799e-01, 1.708613359915442276e+00,
-2.304572304318611931e+00, -2.901853234173955576e-01, 1.574806991991873906e+00,
-3.686506949654931864e-01, -5.659022855302459076e-01, -2.713224715456747371e-01],
[-3.666197184342024490e-02, -6.581173297440267023e-01, 9.860377041753285443e-01,
-1.326949218520296458e+00, 7.239557942782447464e-01, -3.407979385835242514e-01,
5.171609251807368635e-01, 2.781469506536183856e-01, 6.921820455152720708e-01,
-1.415174943118219675e+00, -2.909416696315803574e-01, 6.928982264963062798e-01,
-2.159990523316022704e-01, -2.053933938917264224e-01, -2.961628695712526307e-02],
[-1.417846795007553784e-01, -2.872523793602910908e-02, -1.926139784730311055e-01,
2.453954429832126138e-01, 1.195810213348604129e-01, -1.788218466349852237e-03,
8.902599387877900561e-03, -3.149081757871563872e-03, -6.390198831448234973e-02,
-8.155385586014166077e-02, -2.901346149215791939e-02, 9.021375536552728169e-02,
-2.662070296081403553e-01, 1.643091293337956005e-02, -8.117585143928934821e-03]])


sm_pca = pystan.StanModel(file='pca.stan')
data = {"N": 15, "D": 3, "K": 2, "X":X.T, "mu_W": 0, "alpha0": 1, "beta0": 1}
results = sm_pca.sampling(data=data, iter=10000, chains=2, n_jobs=2)
Ws = results.extract()["W"]
Taus = results.extract()["tau"]

Ws = [np.array(W) for W in Ws]
Taus = [Tau for Tau in Taus]
rand_indices = [random.randint(0, len(Ws)-1) for i in range(0, 25)]

U1s = []
U2s = []

for rand_index in rand_indices:
    WtW = np.dot(Ws[rand_index],(Ws[rand_index].T))
    cov_matrix = WtW + (1.0/Taus[rand_index]) * np.identity(len(WtW))
    (eigen_values, eigen_vectors) = np.linalg.eig(cov_matrix)
    paired_results = list(zip(eigen_values, eigen_vectors))
    # Filter the eigenvalues to keep the first two largest.
    best_two = (heapq.nlargest(2, paired_results, key = lambda item:item[0]))

    us = [np.multiply(np.array(eigen_vec), math.sqrt(eigen_val)) for (eigen_val, eigen_vec) in b
    U1s.append(us[0])
    U2s.append(us[1])

projected_u1s = [[x,y,0] for [x,y,z] in U1s]
projected_u2s = [[x,y,0] for [x,y,z] in U2s]
projected_data = [[x,y,0] for [x,y,z] in X.T]
(u,s,v) = np.linalg.svd(X.T)
projected_eigen_vectors = [[x,y,0] for [x,y,z] in v]

projected_u1s_xs = [x[0] for x in projected_u1s]
projected_u1s_ys = [y[1] for y in projected_u1s]
```

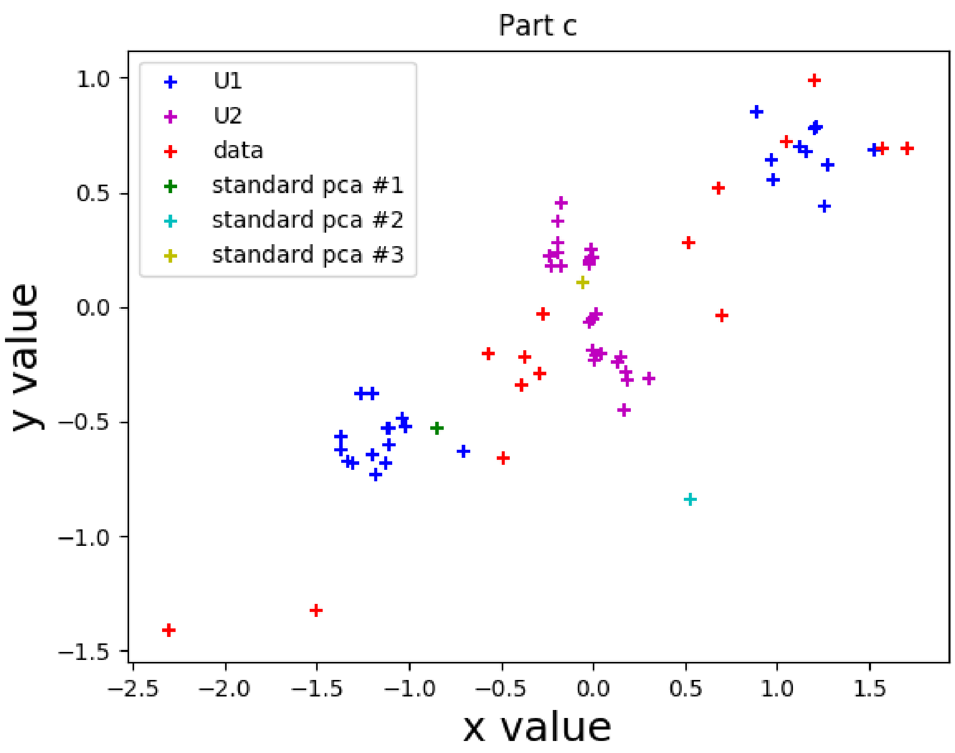

Figure 6:

```
projected_u2s_xs = [x[0] for x in projected_u2s]
projected_u2s_ys = [y[1] for y in projected_u2s]

projected_data_xs = [x[0] for x in projected_data]
projected_data_ys = [y[1] for y in projected_data]

projected_eigen_vectors_xs = [x[0] for x in projected_eigen_vectors]
projected_eigen_vectors_ys = [y[1] for y in projected_eigen_vectors]

print(list(zip(U1s, U2s)))

plt.scatter(projected_u1s_xs, projected_u1s_ys, c='b', label='U1', marker = '+')
plt.scatter(projected_u2s_xs, projected_u2s_ys, c='m', label='U2', marker = '+')
plt.scatter(projected_data_xs, projected_data_ys, c='r', marker = '+', label='data')
plt.scatter(projected_eigen_vectors_xs[0], projected_eigen_vectors_ys[0], c='g', marker = '+', la
plt.scatter(projected_eigen_vectors_xs[1], projected_eigen_vectors_ys[1], c='c', marker = '+', la
plt.scatter(projected_eigen_vectors_xs[2], projected_eigen_vectors_ys[2], c='y', marker = '+', la
plt.legend()
plt.xlabel('x value', fontsize=18)
plt.ylabel('y value', fontsize=18)
plt.title("Part c")
plt.show()
```

Part (d)

The plots are Figures 10 and 11. Our analysis was not necessarily conclusive and there are many levels of explanation which could occur. However, we include a basic analysis to demonstrate our efforts. Here, we analyse this question by first looking at the observed data. The observed data will explain how the model has parameters inferred. We then dissected the Stan code revealing that in Bayesian PCA, the latent variables, and all of the weights have Gaussian values with mean and variances that are free to independently vary in order to explain the data. We investigated potential combinations of Gaussian variables that

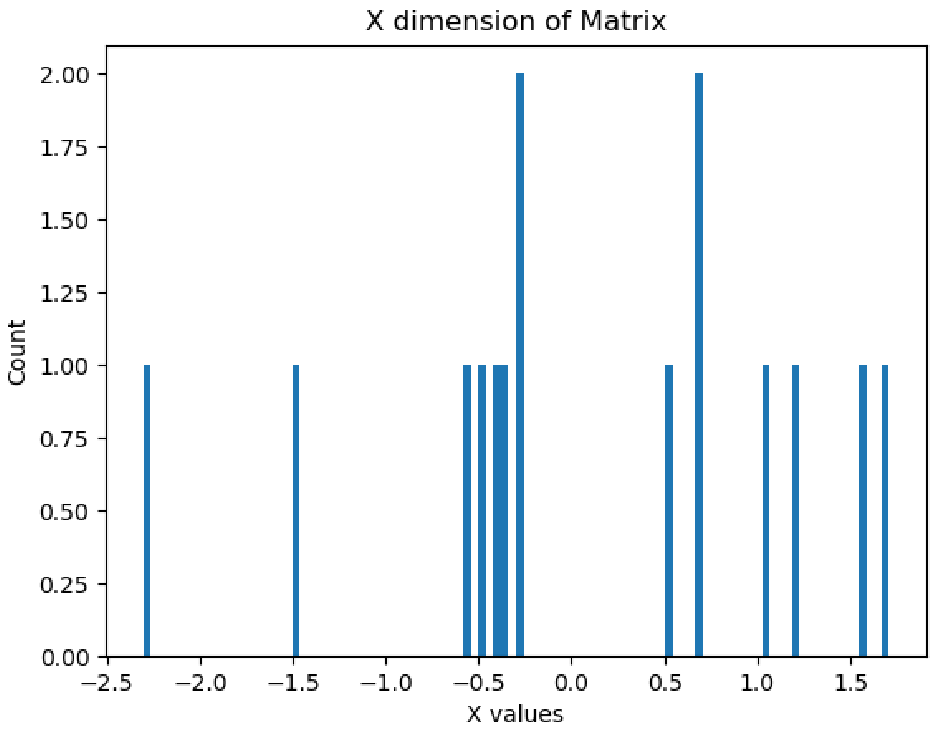

Figure 7:

would combine in ways that produced this doughnut plot, which looks like a Gaussian subtracted from its middle. By considering the PCA directed graph, there are three observed, x variables, corresponding to each dimension. They are plotted in a histogram here. Each latent variable z, is Gaussian, and a linear sum of these latents with the weights yield the observed data plus some added Gaussian noise. The data is plotted in Figures 7, 8 and 9. In the case of the first dimension of our data, the data is bimodal with non-zero means. This means that if the weights are Gaussians centred at zero, then observed variables will tend to be zero, which does not fit the data. This explains the fact that not both of the weights can be zero in Figure 10. However, one of the weights can be zero and the other non-zero as this has the effect of explaining one of the peaks in the data.

Using this as a theory, we then predicted that the third dimension, which resembles a singular Gaussian centred at zero, would have both weights centred at zero, which is what we found. As the second dimension has also a bimodal distribution but both peaks centred closer to zero, we expected a doughnut shaped plot but with a smaller ring in the model, which again is exactly what we found. These are plotted in Figures 12 and 13.

When the mean is changed to a value of 1, we believe that the latents change, to accommodate the negative values. This is because the weights now have a prior in the positive regions, so to accommodate the values below zero for each dimension, the latents, which only have a prior mean of zero will move instead to sub zero values. We plotted the values of z and found this relationship exactly. When the mean is zero, the z values were equally likely to assume values from the unit Gaussian, but when the prior of W moved to one, as one of the two z values increases, the other z value decreases. These two relationships are shown in 14 and 15. In some sense the latent variables internalised this shift of prior.

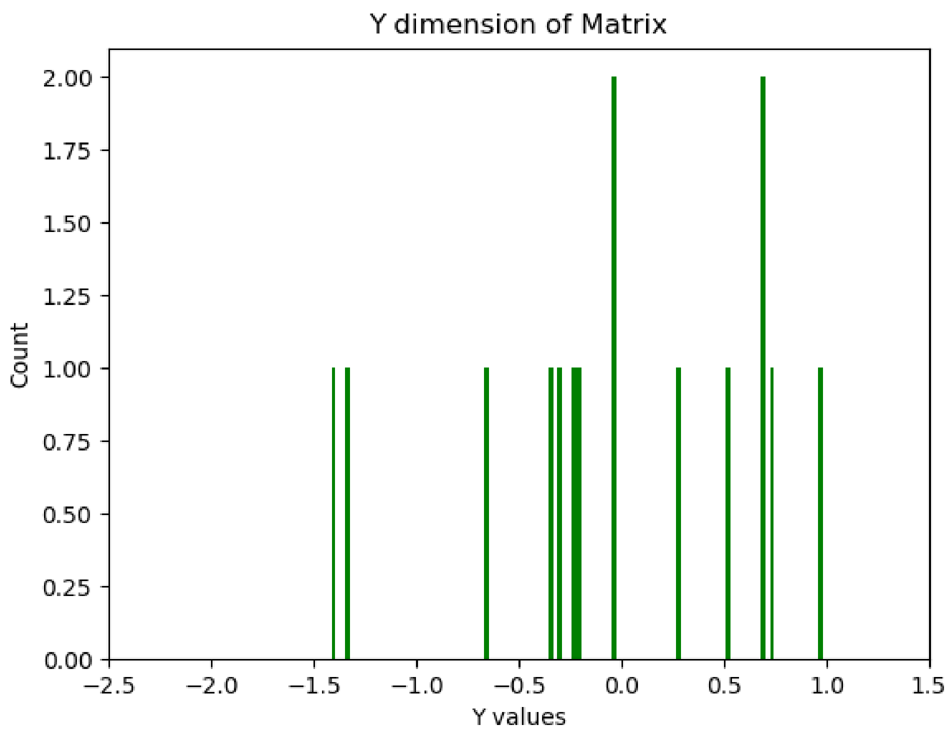

Figure 8:

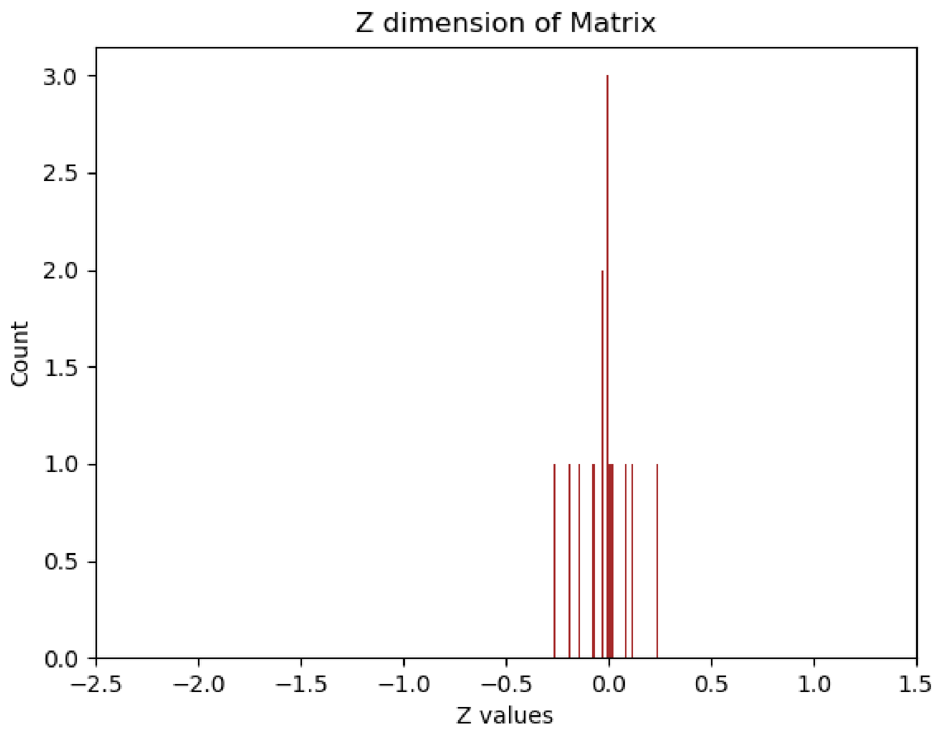

Figure 9:

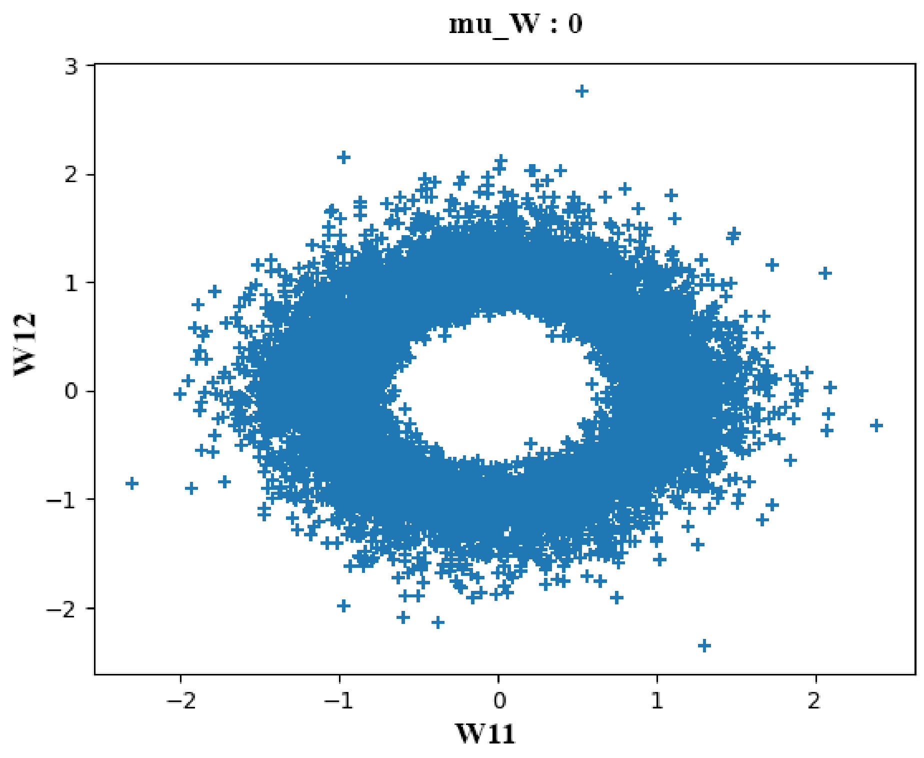

Figure 10:

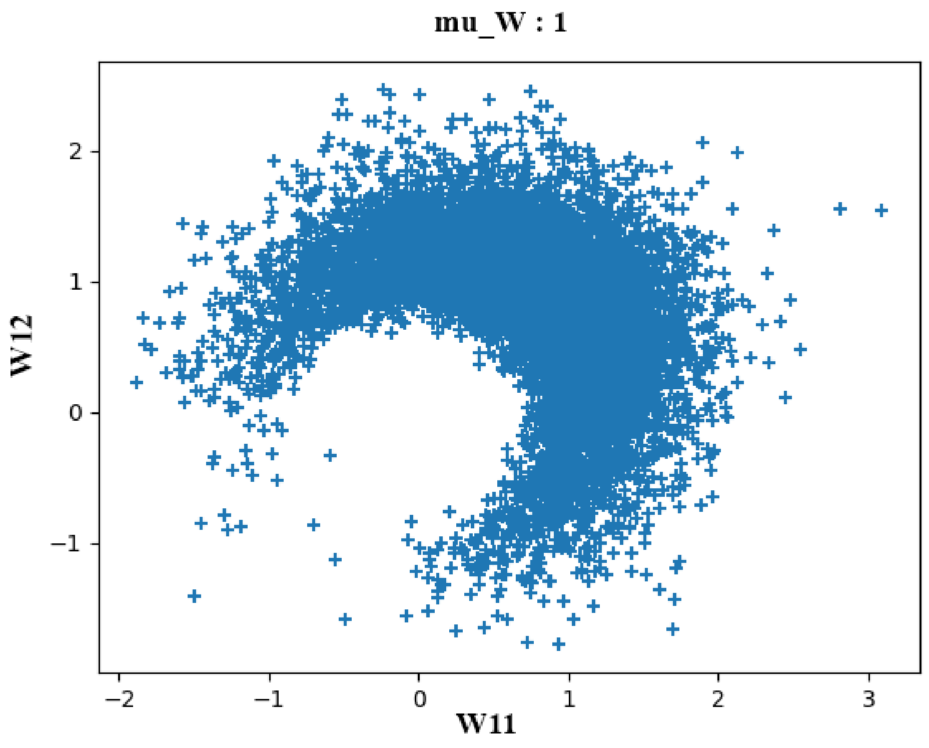

Figure 11:

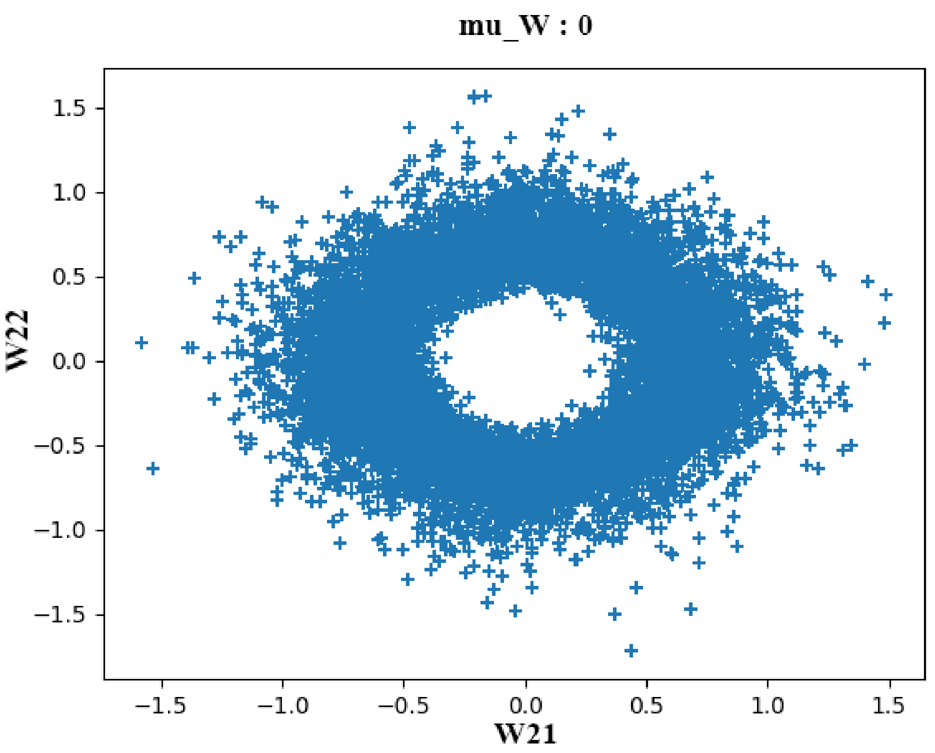

Figure 12:

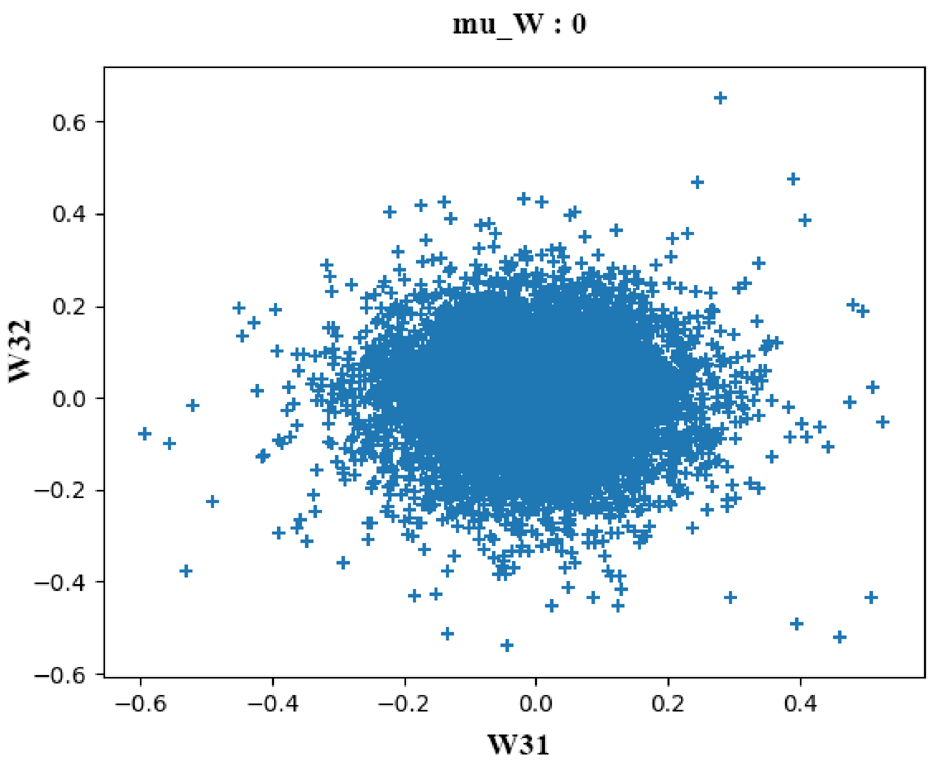

Figure 13:

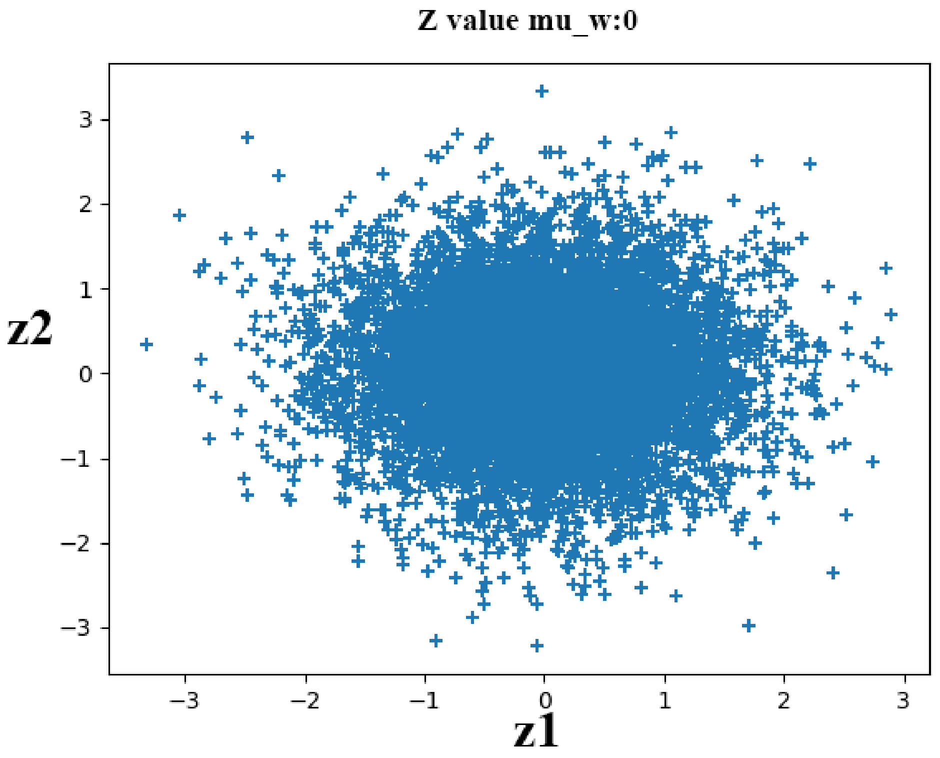

Figure 14:

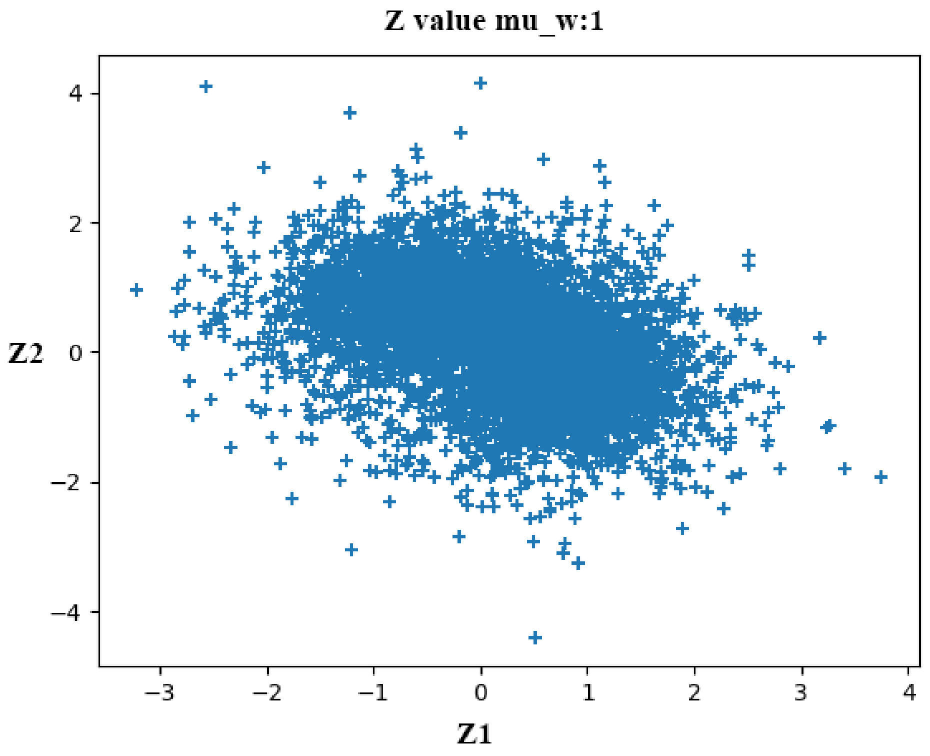

Figure 15: