MDA Part 2, Assignment 1

Marinos Marios

January 5, 2021

Exercise 1

Because all the y_i are **independent** we can factorise the probabilities of each one to find the likelihood $P(y|\mu)$.

$$P(y|\mu) = P(y_1|\mu_1) \cdot P(y_2|\mu_2) \cdot \cdots \cdot P(y_n|\mu_n) = \prod_{i=1}^{n} P(y_i|\mu_i),\ \ y_i \sim Pois(\mu_i)$$

$$Thus,\ P(y|\mu) = \prod_{i=1}^{n} \frac{-e^{x^T\theta}(e^{x^T\theta})^{y_i}}{y_i!},\ as\ \mu_i = e^{x_i^T\theta}$$

$$LL = logP(y|\mu) = \sum_{i=1}^{n}(-e^{x^T\theta} + y_i x^T\theta - log(y_i!)) \tag{1.1}$$

Exercise 2

To find out the the gradient we need to take the derivative of formula 1.1 with respect to θ as this is what we want to maximize.

$$\frac{\partial LL}{\partial \theta} = \sum_{i=1}^{n}(-e^{x^T\theta} + y_i x^T\theta - log(y_i!)) = -X^T e^{X\theta^T} + X^T Y \tag{2.1}$$

With θ a 1 × p vector, X is a n × p matrix and Y a n × 1 vector. Now, to get the hessian of log likelihood we need again to get the derivative of formula 2.1. but now with respect to θ^T.

$$\frac{\partial^2 LL}{\partial \theta \theta^T} = -X^T e^{X\theta^T} X \tag{2.2}$$

where $e^{X\theta^T}$ being an identity matrix with $e^{X\theta^T}$ on the diagonal entries.

Exercise 3

The Laplace approximation is approximated as normal distribution so it is evident that :

$$N(m, P)$$

$$m = \hat{\theta}$$

$$P = -\left(\frac{\partial^2 LL}{\partial \theta \partial \theta^T}\right)^{-1}\Bigg|_{\hat{\theta}} \tag{3.1}$$

Using formulas 2.1 and 2.2 to calculate the gradient and the hessian respectively we end up having the means and Covariance matrix of the Laplace approximation after 1000 iterations (it is already converging from 50-100 iterations). Finally, the new thetas for each iteration were calculated based on formula 3.2.

$$\theta^{j+1} = \theta^{j} - (H(\theta^{j}))^{-1}\nabla F(\theta^{j}) \quad [3.2]$$

$$means = \begin{bmatrix} 1.12777336 \\ 0.42858431 \\ 0.01512131 \\ -0.05416601 \end{bmatrix}, P = \begin{bmatrix} 0.03140956 & -0.00820031 & 0.00116572 & -0.00139328 \\ -0.00820031 & 0.00305825 & -0.00031134 & 0.00066138 \\ 0.00116572 & -0.00031134 & 0.0148073 & -0.0014676 \\ -0.00139328 & 0.00066138 & -0.0014676 & 0.01173136 \end{bmatrix}$$

Exercise 4

Random-walk Metropolis Hastings algorithm have 3 steps. First we propose a $\hat{\theta}$. Then we compute the acceptance rate and if $acceptance\ rate > 1$ we accept new $\theta's$ instantly. If $0 < acceptance\ rate < 1$ we accept the proposed $\theta's$ with probability $acceptance\ rate(r)$. shown in formula 4.1 and finally we have to whether accept or not the proposed $\hat{\theta's}$.

$$\alpha(\theta, \hat{\theta}) = min\left(1, \frac{\pi(\hat{\theta})}{\pi(\theta)}\frac{q(\hat{\theta}, \theta)}{q(\theta, \hat{\theta})}\right) \quad [4.1]$$

$$\theta_{n+1} = \begin{cases} \hat{\theta} & with\ probability\ \alpha(\theta, \hat{\theta}) \\ \theta & with\ probability\ 1 - \alpha(\theta, \hat{\theta}) \end{cases} \quad [4.2]$$

But, as we consider random walk proposals drawn by normal distribution $q(\theta, \hat{\theta}) = q(\hat{\theta}, \theta)$, thus we need only to compute the posterior $\pi(\theta) = p(\theta|x)$.

$$\alpha = \begin{cases} 1 & if\ log(\pi(\hat{\theta})) > log(\pi(\theta)) \\ \dfrac{log\pi(\hat{\theta})}{log\pi(\theta)} & else\ log(\pi(\hat{\theta})) < log(\pi(\theta)) \end{cases} \quad [4.3]$$

If the second if holds, then we draw a random number from uniform distribution $u \sim U(0, 1)$ and if the ratio $\dfrac{log\pi(\hat{\theta})}{log\pi(\theta)}$ is bigger than the random number u then we accept the proposed $\hat{\theta}$, otherwise we stay in current θ.

The $\sigma_{proposed}$ was set to 0.1 after running multiple experiments and with 5000 iterations achieves approximately 25% of acceptance rate. The plot for $\theta_1\ vs\ \theta 2$ is shown in Figure 1.

Figure 2: MCMC sampling for thetas with Metropolis-Hastings. All samples are shown.

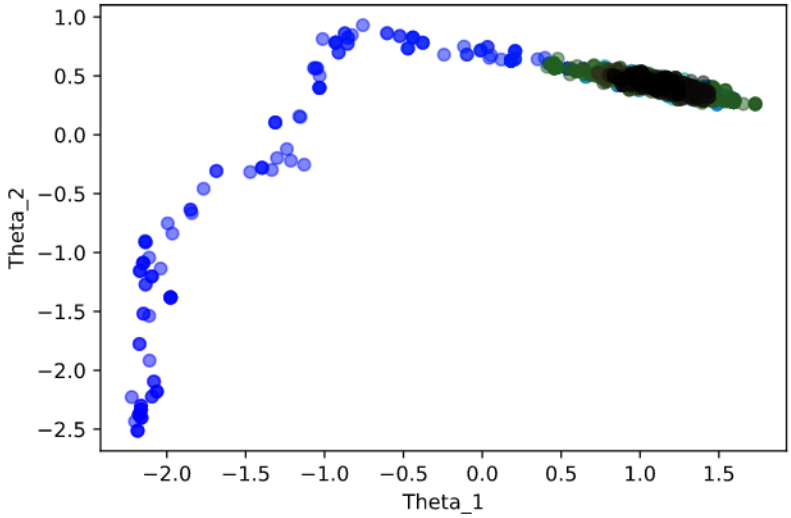

Figure 1: θ_2 vs θ_1, iterations is shown in color of data points starting from first blue iterations and reaching black (latest iterations).

The monte-carlo posterior mean estimation is shown in matrix 4.4 after removing first 1000 iterations as it is considered as burn-in:

$$MC\ means = \begin{bmatrix} 1.10261855 \\ 0.43174725 \\ 0.01600701 \\ -0.04735936 \end{bmatrix} \quad [4.4]$$

Exercise 5

To use the Gibbs Sampler we need to do 2 steps. Iteratively update θ and σ^2.

$$\theta|\sigma^2, \underline{y} \quad [5.1]$$
$$\sigma^2|\theta, \underline{y} \quad [5.2]$$

The first step (updating θ) is implemented on exercise 4. So the $sigma^2$ update rule is missing. This is what we will compute.

$$p(\sigma^2|\theta, y) \propto p(\sigma^2)p(\theta|\sigma^2),\ \theta \sim N(0, \sigma^2 I_p)$$
$$p(\sigma^2)\frac{1}{2\pi}(\sigma^2)^{-2}e^{-\frac{1}{2\sigma^2}\|\theta\|^2},\ but\ \sigma^2 \sim IG(\alpha, \beta)$$
$$(\sigma^2)^{-(A+1)}e^{-\frac{B}{\sigma^2}}\frac{1}{2\pi}(\sigma^2)^{-2}e^{-\frac{1}{2\sigma^2}\|\theta\|^2}$$

So, after keeping all things that only depends on σ^2 we have :

$$p(\sigma^2|\theta) \propto (\sigma^2)^{-(A+2)-1} e^{-\frac{\frac{1}{2}\|\theta\|^2 + B}{\sigma^2}},$$
$$And\ thus\ \sigma^2|\theta \sim IG(A+2, B+\frac{1}{2}\|\theta\|^2)$$

The trace plot for the update-step σ^2 is shown in Figure 2.

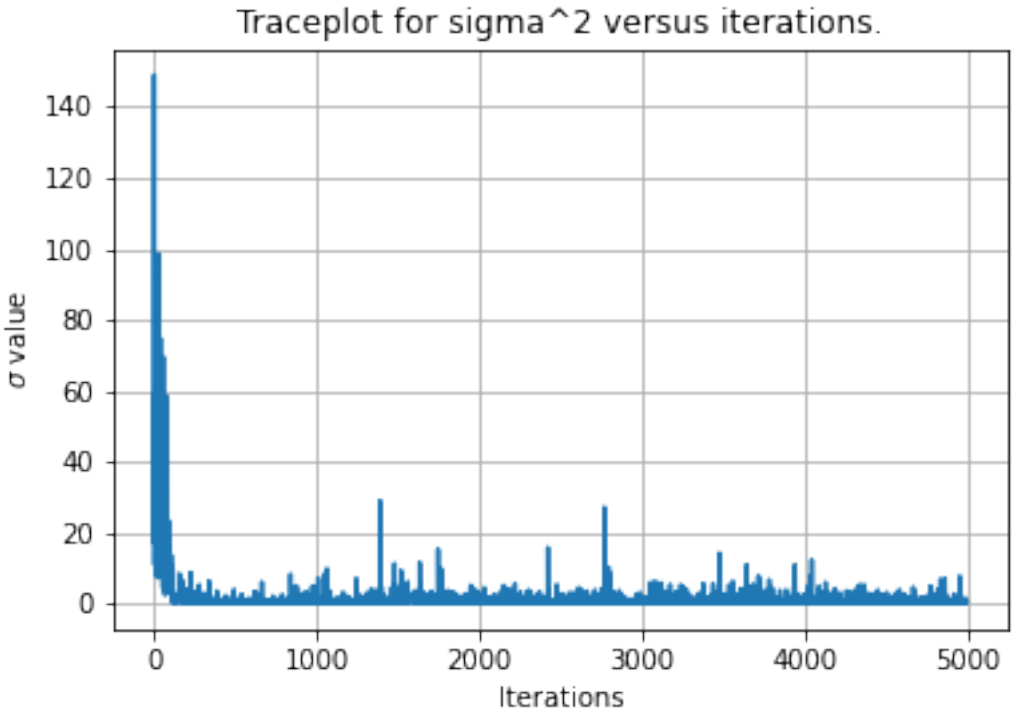

Figure 2: Traceplot showing σ^2 versus iterations.

A Code Exercise 3

```
import pandas as pd
import numpy as np

from numpy import genfromtxt
my_data = genfromtxt('dataexercise2.csv', delimiter=',')
# drop first line of nan's (x1,x2..,y)
my_data = np.delete(my_data, (0), axis=0)
np.set_printoptions(suppress=True) #prevent numpy exponential
  #notation on print, default False

  # split the data
Data_X = my_data[:,0:4]
Data_Y = my_data[:,4]
std_dev = 5
# iterations to reach the minimum.
iterations = 1000
# draw random thetas from multivariate normal distribution.
thetas = np.random.multivariate_normal(np.zeros(4,), np.eye(4) * std_dev)

# iterate to calculate the Laplace approx
for iteration in range(iterations):
    # calculate the gradiant of the points.
    grad = -Data_X.T @ np.exp(Data_X @ thetas.T) + Data_X.T @ Data_Y
    # calculate the hessian matrix.
    hessian = -Data_X.T @ np.diag(np.exp(Data_X @ thetas.T)) @ Data_X
    # calculate new thetas
    thetas = thetas - (np.linalg.inv(hessian) @ grad)

#Calculate final means and covariance matrices of the posterior Laplace approxima
means = thetas
#print("means {}".format(means))
P = -np.linalg.inv(hessian)
#print(P)
```

B Code Exercise 4

```
import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from colour import Color

from numpy import genfromtxt

def prior(thetas):
    # thetas : p x 1 vector.
```

```
    # Calculate the prior of thetas given the initilization distribution of theta
    prior_probability = multivariate_normal.pdf \
        (thetas.T, mean = [0,0,0,0], cov = 5**2 * np.eye(4))
    # return the prior probability.
    return prior_probability

# Computes the posterior probability of the given parameters theta and data.
def manual_log_like_normal(thetas, data):
    # Data_X is p x 1 vector
    # thetas is p x 1 vector
    # my data (X) is the design matrix n x p
    # Data_Y is n x 1 vector with the observed output.
    Data_X = my_data[:,0:4]
    Data_Y = my_data[:,4]

    # Computes the likelihood of the data given thetas (new or current) according
    likelihood = 0
    for i in range(len(Data_X)):
        likelihood += -np.exp(Data_X[i].T @ thetas) + \
            (Data_Y[i] * Data_X[i].T @ thetas) - np.log(np.math.factorial(Data_Y

    # calculate the posterior by adding the log prior's
    # probability as we have log in both likelihood and prior.
    posterior = likelihood + np.log(prior(thetas))
    return posterior

#Defines whether to accept or reject the new sample thetas.
def acceptance_rule(thetas_curr, thetas_new):
    # check if the posterior with theta new is bigger than
    # current posterior (thetas_new > thetas_curr) we update theta.
    if thetas_new > thetas_curr:
        return True
    else:
        accept=np.random.uniform(0,1)
        # Since we did a log likelihood, we need to
        # exponentiate in order to compare to the random number
        return (accept < (np.exp(thetas_new-thetas_curr)))

def metropolis_hastings(likelihood_computer, param_init, iterations, data, accept
    # likelihood_computer(x,data): returns the likelihood that these parameters g
    # transition_model(x): a function that draws a sample from a symmetric distri
    # param_init: a starting sample
    # iterations: number of accepted to generated
    # data: the data that we wish to model
    # acceptance_rule(x, x_new): decides whether to accept or reject the new samp
    theta_curr = param_init
    accepted = []
    rejected = []
```

```
    # initialize an empty max numpy array for each iteration
    # to save thetas to calculate posterior mean.
    theta_accepted = np.zeros(shape = (iterations, 4))
    for i in range(iterations):
        # adjust the current theta by gaussian noise.
        theta_new =  theta_curr + np.random.normal(0, 0.1, (4,1))
        posterior_current_theta = likelihood_computer(theta_curr, data)
        posterior_new_theta = likelihood_computer(theta_new, data)
        if (acceptance_rule(posterior_current_theta, posterior_new_theta)):
            theta_curr = theta_new
            # update the accepted means to calculate the monte carlo posterior me
            theta_accepted[i] = np.reshape(theta_curr, (4,))
            accepted.append(theta_new)
        else:
            theta_accepted[i] = np.reshape(theta_curr, (4,))
            rejected.append(theta_new)

    return np.array(accepted), np.array(rejected), theta_accepted


if __name__ == "__main__":

    my_data = genfromtxt('dataexercise2.csv', delimiter=',')
    # drop first line of nan's (x1, x2..,y)
    my_data = np.delete(my_data, (0), axis=0)
    np.set_printoptions(suppress=True) #prevent numpy exponential
    #notation on print, default False.
    std_dev = 5
    # iterations to reach the minimum.
    iterations = 5000
    # draw random thetas from normal distribution to initialize thetas.
    thetas = np.random.normal(0, 5, (4, 1))
    # thetas using multivariate insstead.
    # thetas = np.random.multivariate_normal([0,0,0,0], 5 * np.eye(4))
    accepted, rejected, means_accepted = metropolis_hastings \
    (manual_log_like_normal, thetas, iterations, my_data, acceptance_rule)

    # plot the results of the simulation.
    # pick a color for the iterations
    blue = Color("blue")
    colors = list(blue.range_to(Color("black"), len(means_accepted)))
    fig, ax = plt.subplots()
    # plot theta_1 vs theta 2
    for i in range(len(means_accepted)):
        ax.scatter( means_accepted[i,0], means_accepted[i,1],alpha=0.5, c = str(c
    plt.xlabel("Theta_1")
    plt.ylabel("Theta_2")
    plt.title("Figure 2: MCMC sampling for thetas with Metropolis-Hastings. All s
```

```
    plt.show()

        # calcualte acceptance rate
    acceptance_rate = accepted.shape[0] / iterations
    acceptance_rate

    # burn in is 20% of the iterations (1000 iterations out of 5000) so we drop t
    means_accepted = means_accepted[1000:, :]
    # calculate the means of each theta.
    column_mean = means_accepted.mean(axis=0)
    print(column_mean)
```

C Code Exercise 5

```
import pandas as pd
import numpy as np
from numpy import genfromtxt
from scipy.stats import multivariate_normal, invgamma
import matplotlib.pyplot as plt
from colour import Color


def prior(thetas):
    # thetas : p x 1 vector.
    # Calculate the prior of thetas given the initilization
    #distribution of theta (probability those theta are drawn
    # from this distribution with means = 0 and variance 5.).
    prior_probability = multivariate_normal.pdf\
        (thetas.T, mean = [0,0,0,0], cov = 5 ** 2 * np.eye(4))
    # return the prior probability.
    return prior_probability

# Computes the posterior probability of the given parameters theta and data.
def manual_log_like_normal(thetas, data):
    # Data_X is p x 1 vector
    # thetas is p x 1 vector
    # my data (X) is the design matrix n x p
    # Data_Y is n x 1 vector with the observed output.
    Data_X = my_data[:,0:4]
    Data_Y = my_data[:,4]

    # Computes the likelihood of the data given thetas (new or current) accordin
    likelihood = 0
    for i in range(len(Data_X)):
        likelihood += -np.exp(Data_X[i].T @ thetas) + \
        (Data_Y[i] * Data_X[i].T @ thetas) - np.log(np.math.factorial(Data_Y[i]))
```

```
    # calculate the posterior by adding the log prior's probability as we have lo
    posterior = likelihood + np.log(prior(thetas))
    return posterior

#Defines whether to accept or reject the new sample thetas.
def acceptance_rule(thetas_curr, thetas_new):
    # check if the posterior with theta new is bigger than current posterior (the
    if thetas_new > thetas_curr:
        return True
    else:
        accept=np.random.uniform(0,1)
        # Since we did a log likelihood, we need to exponentiate in order to comp
        # less likely x_new are less likely to be accepted
        return (accept < (np.exp(thetas_new-thetas_curr)))

def Gibbs_sampler(likelihood_computer, param_init, iterations, data, acceptance_r
    # likelihood_computer(x,data): returns the likelihood that these parameters g
    # transition_model(x): a function that draws a sample from a symmetric distri
    # param_init: a starting sample
    # iterations: number of accepted to generated
    # data: the data that we wish to model
    # acceptance_rule(x, x_new): decides whether to accept or reject the new samp
    theta_curr = param_init
    accepted = []
    rejected = []
    # alpha, beta for the gibbs sampler for sigma^{2} to draw
    # samples from sigma^{2} ~ IG(alpha + 2, B + /thetas.T @ thetas)
    alpha = 0.1
    beta = 0.1
    sigmas = []
    # initialize an empty max numpy array for each iteration to save thetas to ca
    theta_accepted = np.zeros(shape = (iterations, 4))
    for i in range(iterations):
        # 1st step : draw sigma from inverse distribution,
        # sample sigma from ~ IG(A+2, B+ (np.linalg.norm(thetas) ** 2) /2)
        inverse_gamma_sigma = invgamma.rvs \
            (a=alpha + 2, scale = beta + (theta_curr.T @ theta_curr) / 2)
        # 1/gamma_sigma to get the inverse gamma value.
        sigmas.append(inverse_gamma_sigma)
        # 2nd step : draw theta from normal distribution.
        # Then adjust the current theta by gaussian noise,
        # using sigma drawn from Inverse gamma distribution
        theta_new =  theta_curr + np.random.normal(0, inverse_gamma_sigma, (4,1))
        # 3rd step : calculate the posterior : prior + likelihood using theta and
        posterior_current_theta = likelihood_computer(theta_curr, data)
        posterior_new_theta = likelihood_computer(theta_new, data)
        # 4th step : check whether you accept or not the proposed thetas
```

```
        # based on the acceptance rule.
        if (acceptance_rule(posterior_current_theta, posterior_new_theta)):
            theta_curr = theta_new
            # update the accepted means to calculate the monte carlo posterior me
            theta_accepted[i] = np.reshape(theta_curr, (4,))
            accepted.append(theta_new)
        else:
            theta_accepted[i] = np.reshape(theta_curr, (4,))
            rejected.append(theta_new)

    return np.array(accepted), np.array(rejected), theta_accepted, sigmas


if __name__ == "__main__":
    my_data = genfromtxt('dataexercise2.csv', delimiter=',')
    # drop first line of nan's (x1, x2..,y)
    my_data = np.delete(my_data, (0), axis=0)
    np.set_printoptions(suppress=True) #prevent numpy exponential
    #notation on print, default False.
    std_dev = 5
    # iterations to reach the minimum.
    iterations = 5000
    # draw random thetas from normal distribution to initialize thetas.
    thetas = np.random.normal(0, std_dev, (4, 1))
    # thetas using multivariate insstead.
    # thetas = np.random.multivariate_normal([0,0,0,0], 5 * np.eye(4))
    accepted, rejected, means_accepted, sigmas = Gibbs_sampler\
    (manual_log_like_normal, thetas, iterations, my_data, acceptance_rule)
    # create traceplot

    fig, ax = plt.subplots()

    ax.plot(range(len(sigmas)), sigmas)
    ax.set(xlabel='Iterations', ylabel='$\sigma$ value',
        title='Traceplot for sigma^2 versus iterations.')
    ax.grid()

    fig.savefig("gibbs_sigma.png")
    plt.show()
```