Statistical Models and Computing Methods **2020 Fall**

Problem Set 1. Solution

2020-11

1 Problem 1

(1) Show that $X \sim \mathcal{N}(0,1)$ is the maximum entropy distribution such that $\mathbb{E}X = 0$ and $\mathbb{E}X^2 = 1$.

Solution.

$$\begin{aligned} \min_p \quad & \textstyle\int_{\mathcal{X}} p(x)\log p(x)\,\mathrm{d}x \\ \text{s.t.} \quad & \textstyle\int_{\mathcal{X}} p(x)\,\mathrm{d}x = 1 \\ & \textstyle\int_{\mathcal{X}} x\,p(x)\,\mathrm{d}x = 0 \\ & \textstyle\int_{\mathcal{X}} x^2 p(x)\,\mathrm{d}x = 1 \end{aligned}$$

The Lagrangian is

$$\mathcal{L}(p;\boldsymbol{\lambda},\mu) = \textstyle\int p(x)\log p(x)\,\mathrm{d}x + \lambda_0\left(\int p(x)\,\mathrm{d}x - 1\right) + \lambda_1\int x\,p(x)\,\mathrm{d}x + \lambda_2\left(\int x^2 p(x)\,\mathrm{d}x - 1\right)$$

which is convex in p. Then taking

$$\frac{\partial\mathcal{L}}{\partial p} = \log p(x) + 1 + \lambda_0 + \lambda_1 x + \lambda_2 x^2 = 0$$

we have

$$p(x) = \exp\left(-\left(\lambda_0 + \lambda_1 x + \lambda_2 x^2\right) - 1\right) \geqslant 0.$$

$p(x) = \exp\left(-\left(\lambda_0 + \lambda_1 x + \lambda_2 x^2\right) - 1\right)$ with $\int x\,p(x)\,\mathrm{d}x = 0$, we have $\lambda_1 = 0$.
$p(x) = \exp\left(-\left(\lambda_0 + \lambda_2 x^2\right) - 1\right)$ with $\int x^2 p(x)\,\mathrm{d}x = 1$ and $\int p(x)\,\mathrm{d}x = 1$, we have $\lambda_0 = \log\sqrt{2\pi} - 1$, $\lambda_2 = \frac{1}{2}$.
Therefore, $p(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}x^2\right)$, i.e., $X \sim \mathcal{N}(0,1)$. ■

(2) Generalize the result in (1) for the maximum entropy distribution given the first k moments, i.e., $\mathbb{E}X^i = m_i, i = 1,\ldots,k$.

Solution. Write the problem as

$$\begin{aligned} \min_p \quad & \textstyle\int_{\mathcal{X}} p(x)\log p(x)\,\mathrm{d}x \\ \text{s.t.} \quad & \textstyle\int_{\mathcal{X}} x^n p(x)\,\mathrm{d}x = m_n, \quad n = 0,\ldots,k, \quad m_0 := 1 \end{aligned}$$

The Lagrangian is

$$\mathcal{L}(p;\boldsymbol{\lambda}) = \textstyle\int p(x)\log p(x)\,\mathrm{d}x + \sum_{0\leqslant n\leqslant k}\lambda_n\left(\int x^n p(x)\,\mathrm{d}x - m_n\right)$$

which is convex in p. Then taking

$$\frac{\partial\mathcal{L}}{\partial p} = \log p(x) + 1 + \textstyle\sum_{0\leqslant n\leqslant k}\lambda_n x^n = 0$$

we have

$$p(x) = \exp\left(-\textstyle\sum_{0\leqslant n\leqslant k}\lambda_n x^n - 1\right),$$

an exponential family, then $\boldsymbol{\lambda}$ is determined by the constraints $\int x^n p(x)\,\mathrm{d}x = m_n,\ n = 0,\ldots,k$. ■

2 Problem 2

Let $Y_1, \ldots, Y_n$ be a set of independent random variables with the following pdfs

$$p\left(y_i \mid \theta_i\right) = \exp\big(y_i b\left(\theta_i\right) + c\left(\theta_i\right) + d\left(y_i\right)\big), \quad i = 1, \ldots, n$$

Let $\mathbb{E}(Y_i) = \mu_i(\theta_i), g(\mu_i) = x_i^{\top}\beta$, where g is the link function and $\beta \in \mathbb{R}^d$ is the vector of model parameters.

(1) Denote $g(\mu_i)$ as η_i, and let s be the score function of β. Show that

$$s_j = \sum_{i=1}^{n} \frac{\left(y_i - \mu_i\right) x_{ij}}{\mathbb{V}\mathrm{ar}\left(Y_i\right)} \frac{\partial \mu_i}{\partial \eta_i}, \quad j = 1, \ldots, d$$

Solution. Let $L_i \coloneqq \log p\left(y_i \mid \theta_i\right) = y_i b\left(\theta_i\right) + c\left(\theta_i\right) + d\left(y_i\right)$, then

$$s_j = \sum_{i=1}^{n} \frac{\partial L_i}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial L_i}{\partial \theta_i} \frac{\partial \theta_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \eta_i}{\partial \beta_j} = \sum_{i=1}^{n} \left(y_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} \right) \frac{\partial \theta_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} x_{ij}, \quad j = 1, \ldots, d.$$

By interchanging the differentiation $\frac{\partial}{\partial \theta_i}$ and integration $\mathbb{E}1, \mathbb{E}Y_i$, we have

$$\begin{aligned} 0 &= \frac{\partial}{\partial \theta_i} \mathbb{E}1 = \mu_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} \\ \frac{\partial \mu_i}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i} \mathbb{E}Y_i = \left(\mathbb{E}Y_i^2\right) \frac{\partial b(\theta_i)}{\partial \theta_i} + \mu_i \frac{\partial c(\theta_i)}{\partial \theta_i}, \end{aligned}$$

then

$$\begin{aligned} y_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} &= \left(y_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} \right) - \left(\mu_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} \right) = \left(y_i - \mu_i\right) \frac{\partial b(\theta_i)}{\partial \theta_i} \\ \frac{\partial \mu_i}{\partial \theta_i} &= \frac{\partial \mu_i}{\partial \theta_i} - \mu_i \left(\mu_i \frac{\partial b(\theta_i)}{\partial \theta_i} + \frac{\partial c(\theta_i)}{\partial \theta_i} \right) = \mathbb{V}\mathrm{ar}\left(Y_i\right) \frac{\partial b(\theta_i)}{\partial \theta_i}. \end{aligned}$$

therefore

$$s_j = \sum_{i=1}^{n} \frac{\left(y_i - \mu_i\right) \frac{\partial b(\theta_i)}{\partial \theta_i}}{\mathbb{V}\mathrm{ar}\left(Y_i\right) \frac{\partial b(\theta_i)}{\partial \theta_i}} \frac{\partial \mu_i}{\partial \eta_i} x_{ij} = \sum_{i=1}^{n} \frac{\left(y_i - \mu_i\right) x_{ij}}{\mathbb{V}\mathrm{ar}\left(Y_i\right)} \frac{\partial \mu_i}{\partial \eta_i}, \quad j = 1, \ldots, d.$$

■

(2) Let $\mathcal{I}$ be the Fisher information matrix. Show that

$$\mathcal{I}_{jk} = \mathbb{E}\left(s_j s_k\right) = \sum_{i=1}^{n} \frac{x_{ij} x_{ik}}{\mathbb{V}\mathrm{ar}\left(Y_i\right)} \left(\frac{\partial \mu_i}{\partial \eta_i}\right)^2, \quad \forall\, 1 \leq j, k \leq d$$

Solution.

$$\begin{aligned} \mathcal{I}_{jk} &= \mathbb{E}\left(s_j s_k\right) \\ &= \mathbb{E}\left(\sum_{i=1}^{n} \frac{\left(y_i - \mu_i\right) x_{ij}}{\mathbb{V}\mathrm{ar}\left(Y_i\right)} \frac{\partial \mu_i}{\partial \eta_i} \sum_{l=1}^{n} \frac{\left(y_l - \mu_l\right) x_{lk}}{\mathbb{V}\mathrm{ar}\left(Y_l\right)} \frac{\partial \mu_l}{\partial \eta_l} \right) \\ &= \sum_{1 \leqslant i, l \leqslant n} \frac{x_{ij} x_{lk} \mathbb{E}\left(y_i - \mu_i\right)\left(y_l - \mu_l\right)}{\mathbb{V}\mathrm{ar}\left(Y_i\right) \mathbb{V}\mathrm{ar}\left(Y_l\right)} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \mu_l}{\partial \eta_l} \\ &= \sum_{i=1}^{n} \frac{x_{ij} x_{ik}}{\mathbb{V}\mathrm{ar}\left(Y_i\right)} \left(\frac{\partial \mu_i}{\partial \eta_i}\right)^2, \quad \text{by independence.} \end{aligned}$$

■

3 Problem 3

Use the following code to generate co-variate matrices X.

```python
import numpy as np

np.random.seed(1234)
n = 100
X = np.random.normal(size=(n,2))
```

(1). Generate $n = 100$ observations Y following the logistic regression model with true parameter $\beta_0 = (-2, 1)$.

Solution.

```python
import numpy as np
from scipy.special import expit as sigmoid

def generate_data(beta0, n=100):
    X = np.random.normal(size=(n, beta0.shape[0]))
    logits = X @ beta0
    probs = sigmoid(logits)
    return {"X":X, "logits":logits, "probs":probs}

def generate_y(probs):
    y = np.random.binomial(1, probs)
    return y

seed = 1234
np.random.seed(seed)
beta0 = np.array([-2., 1.])
data = generate_data(beta0, n=100)
X, probs = data["X"], data["probs"]
y = generate_y(probs=probs)
print(y)
```

```
[0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1 1 0 0 1 1 0
 1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1
 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0]
```

(2). Find the MLE using the iteratively reweighted least square algorithm.

Solution.

```python
import numpy as np
from scipy import sparse
from scipy.special import expit as sigmoid
from numpy.linalg import inv as inverse
from numpy.linalg import norm

```

```python
def irls_lr(X, y, max_itr=200, epsilon=1e-12, quiet=True):
    ''' Readme:
        this function is defined to estimate
        the parameters of logistic regression model
        by IRLS, Iteratively Reweighted Least Square Algorithm.
    '''
    # initialization
    n, d = X.shape
    beta = inverse(X.T@X) @ (X.T@y)
    W = sparse.dia_matrix((n, n))
    err_path = []

    # main
    for i in range(1, max_itr+1):
        logits = X @ beta
        probs = sigmoid(logits)
        W = sparse.diags(probs*(1-probs))
        beta_ = beta + inverse(X.T@W@X) @ (X.T@(y-probs))
        err = norm(beta_-beta)/norm(beta)
        err_path.append(err)
        beta = beta_

        if i % 5 == 0:
            if not quiet:
                print(f"err: {err:.3e}, itr: {i}")
        if err < epsilon:
            if not quiet:
                print(f"err: {err:.3e}, itr: {i}")
                print(f"MLE: {list(beta)}")
            break
    # returns
    out = {"beta":beta, "itr":i, "errs":err_path}
    return out
```

```python
out = irls_lr(X, y, quiet=0)
```

```
err: 2.217e-06, itr: 5
err: 1.627e-16, itr: 7
MLE: [-1.3708659534317205, 0.6698777671314895]
```

(3). Repeat (1) and (2) for 100 instances. Compare the MLEs with the asymptotical distribution $\hat{\beta} \sim \mathcal{N}\left(\beta_0, \mathcal{I}^{-1}\left(\beta_0\right)\right)$. Present your result with a scatter plot for MLEs with contours for the PDF of the asymptotical distribution.

Solution.

```python
W = sparse.diags(probs*(1-probs))
Fisher = X.T @ W @ X
cov = inverse(Fisher)
print(cov)
```

```
[[ 0.19350715 -0.0440933 ]
 [-0.0440933   0.10205911]]
```

```python
def experiment(
        X, probs, num_tri=100,
        generate_y=generate_y, alg=irls_lr,
    ):
    instances = [generate_y(probs) for i in range(1, num_tri+1)]
    betas = [alg(X, y)["beta"] for y in instances]
    return np.array(betas)

np.random.seed(1234)
beta0 = np.array([-2., 1.])
betas = experiment(X, probs, num_tri=100)
print(np.cov(betas.T))
```

```
[[ 0.23124879 -0.06162206]
 [-0.06162206  0.10223682]]
```

```python
from scipy import stats
import matplotlib.pyplot as plt

radius = 0.8
x0 = np.linspace(beta0[0]-radius, beta0[0]+radius, 50)
x1 = np.linspace(beta0[1]-radius, beta0[1]+radius, 50)
X0, X1 = np.meshgrid(x0, x1)  # create x0-x1 meshgrid
pos = np.dstack((X0, X1))     # shape-(num1,num2,d)

binorm_rv1 = stats.multivariate_normal(beta0, cov)
Z1 = binorm_rv1.pdf(pos)

plt.figure(figsize=(6, 6))
plt.plot(betas[:, 0], betas[:, 1], 'o', mfc='none')    # scatter
CS = plt.contour(X0, X1, Z1, cmap='cool')              # contour
plt.clabel(CS, inline=1)
plt.title(r'Comparison ($n=100$)')
plt.savefig('./comparison_100.pdf', bbox_inches='tight')
plt.show()
```

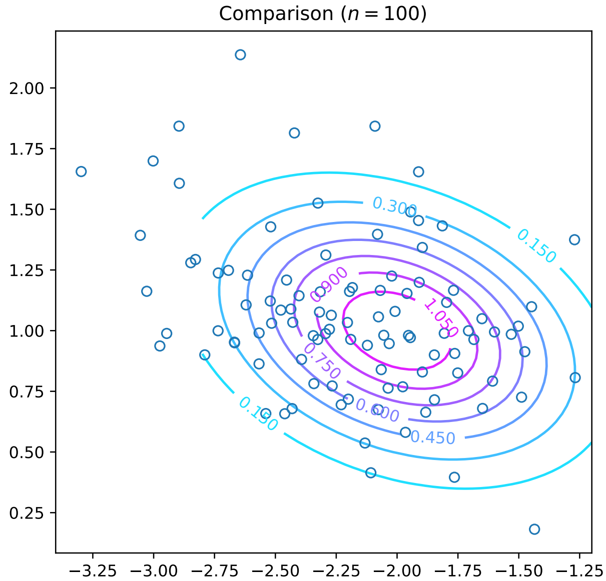

Figure 3.1: Comparison

(4). Try the same for $n = 10000$. Does the asymptotical distribution provide a better fit to the MLEs? You can use the empirical covariance matrix of the MLEs for comparison.

Solution.

```
seed = 1234
np.random.seed(seed)
beta0 = np.array([-2., 1.])

data = generate_data(beta0, n=100)
X, probs = data["X"], data["probs"]
betas_ = experiment(X, probs, num_tri=100)
betas_cov_ = np.cov(betas_.T)

W = sparse.diags(probs*(1-probs))
Fisher = X.T @ W @ X
cov_ = inverse(Fisher)
print(cov_)
print(betas_cov_)
err = np.linalg.norm(cov_-betas_cov_,"fro")
print(f"err:{err:.6f}, err_rel:{err/np.linalg.norm(cov_,'fro'):.6f}")

data = generate_data(beta0, n=10000)
X, probs = data["X"], data["probs"]
betas = experiment(X, probs, num_tri=100)
betas_mean = np.mean(betas, axis=0)
betas_cov = np.cov(betas.T)
W = sparse.diags(probs*(1-probs))
Fisher = X.T @ W @ X
```

```python
cov = inverse(Fisher)
print(betas_cov)
print(cov)
err = np.linalg.norm(cov-betas_cov,"fro")
print(f"err:{err:.6f}, err_rel:{err/np.linalg.norm(cov,'fro'):.6f}")
```

```
[[ 0.19350715 -0.0440933 ]
 [-0.0440933   0.10205911]]
[[ 0.25623927 -0.04276872]
 [-0.04276872  0.12578528]]
err:0.067095, err_rel:0.294943
[[ 0.00202756 -0.00069806]
 [-0.00069806  0.00105249]]
[[ 0.00174037 -0.00053367]
 [-0.00053367  0.00096422]]
err:0.000380, err_rel:0.178520
```

```python
plt.figure(figsize=(12, 6))

radius = 0.4
plt.subplot(121)
x0 = np.linspace(beta0[0]-radius, beta0[0]+radius, 50)
x1 = np.linspace(beta0[1]-radius, beta0[1]+radius, 50)
X0, X1 = np.meshgrid(x0, x1)  # create x0-x1 meshgrid
pos = np.dstack((X0, X1))     # shape-(num1,num2,d)

binorm_rv1 = stats.multivariate_normal(beta0, cov_)
Z1 = binorm_rv1.pdf(pos)
plt.plot(betas[:, 0], betas[:, 1], 'o', mfc='none')     # scatter
CS1 = plt.contour(X0, X1, Z1, cmap='cool')              # contour
plt.clabel(CS1, inline=1)
plt.title(r'Comparison ($n=10000$)')
plt.savefig('./comparison_10000.pdf', bbox_inches='tight')

plt.subplot(122)
x2 = np.linspace(betas_mean[0]-radius, betas_mean[0]+radius, 50)
x3 = np.linspace(betas_mean[1]-radius, betas_mean[1]+radius, 50)
X2, X3 = np.meshgrid(x2, x3)  # create x0-x1 meshgrid
pos2 = np.dstack((X2, X3))     # shape-(num1,num2,d)

binorm_rv2 = stats.multivariate_normal(betas_mean, betas_cov)
Z2 = binorm_rv2.pdf(pos2)
CS1 = plt.contour(X0, X1, Z1, cmap='cool')
plt.clabel(CS1, inline=1)
CS2 = plt.contour(X2, X3, Z2, cmap='spring')
```

```python
plt.clabel(CS2, inline=1)
plt.title(r'Comparison with asymptotical distribution')
plt.savefig('./Asymptotical distribution.pdf', bbox_inches='tight')

plt.show()
```

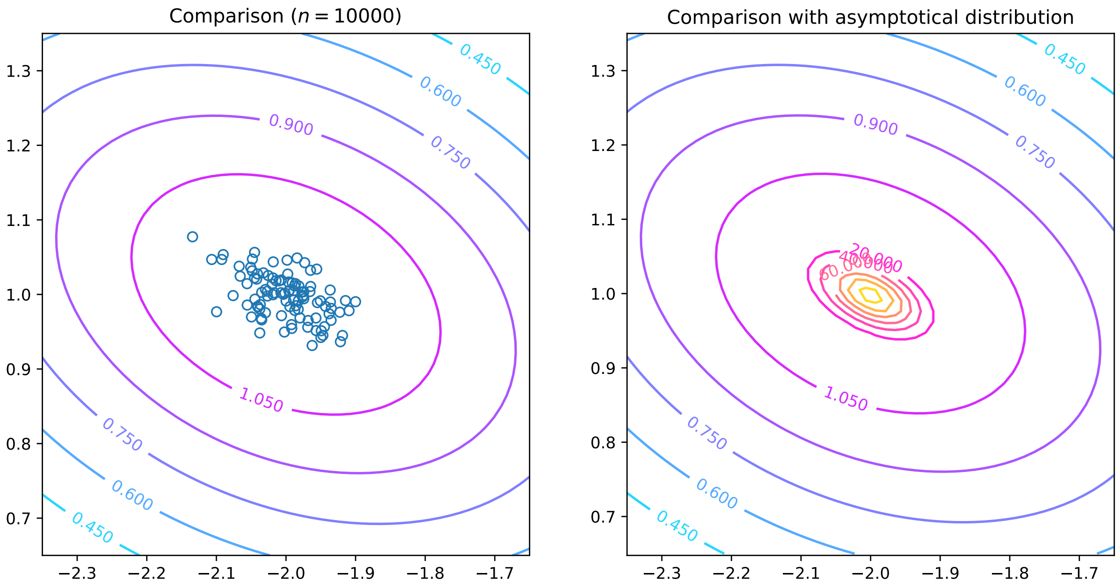

Figure 3.2: Comparison

The asymptotic distribution with the larger sample size, which has a more intense concentration and a lower error rate, suggests a better fit for the MLEs.

4 Problem 4

Consider the probit regression model

$$Y \mid X, \beta \sim \text{ Bernoulli } (p), \quad p = \Phi(X\beta)$$

where Φ is the cumulative distribution function of the standard normal distribution. Similarly as in Problem 3, generate a large covariate matrix X with 100000 instances and 100 features, and response Y with true parameter β_0

```python
import numpy as np
np.random.seed(1234)

n, d = 100000, 100
X = np.random.normal(size=(n, d))
beta_0 = np.random.normal(size=d)
```

(1). Compare gradient descent and Nesterov's accelerated gradient descent.

Solution.

```python
def GD(self,init,tol = 1e-6, step_size = 0.0001, maxit = 1000):
```

```
        beta_old = np.zeros(shape = (np.shape(init)[0],)) + 1
        beta_new = init
        likelihood = np.array([])
        l = 1000
        l_next = 1
        for i in range(1, maxit +1):
            l = l_next
            if i % 100 == 0 :
                print(l)
            grad = self.gradient(beta_new)
            beta_old = beta_new
            beta_new = beta_old + step_size * grad
            l_next = self.loglikelihood(beta_new)
            likelihood = np.append(likelihood,l_next)
            if abs(l - l_next) / abs(l) < tol:
                break
        return beta_new, likelihood

def NAG(self, init, tol = 1e-6, step_size = 0.0001, maxit = 1000):
    beta_previous = init
    beta_now = init
    beta_new = init
    likelihood = np.array([])
    l = 1000
    l_next = 1
    for i in range(1, maxit +1):
        l = l_next
        beta_previous = beta_now
        beta_now = beta_new
        if i % 100 == 0 :
            print(l)
        y = beta_now + (i - 2) / (i + 1) * (beta_now - beta_previous)
        grad = self.gradient(y)
        beta_new = y + step_size * grad
        l_next = self.loglikelihood(beta_new)
        likelihood = np.append(likelihood,l_next)
        if abs(l - l_next) / abs(l) < tol:
            break
    return beta_new, likelihood
```

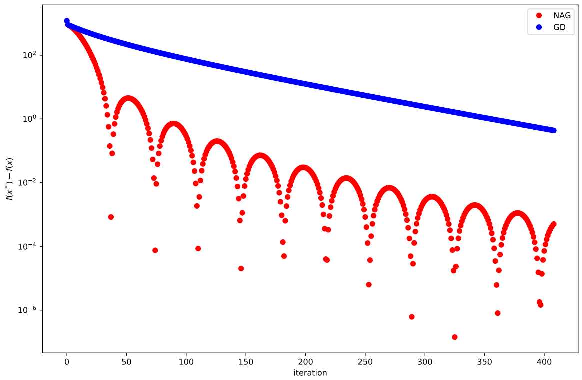

Figure 4.3: Comparison: SGD and NAG

Although the objective function is non-monotone decreasing when using nesterov's acceleration, it can reach a much optimal value in a fairly less amount of time.

(2). Compare vanilla stochastic gradient descent with different adaptive stochastic gradi- ent descent methods, including AdaGrad, RMSprop, and Adam. Using minibatch sizes 32,64,128.

Solution.

```
def SGD(self, init, tol = 1e-6, step_size = 0.1, batch_size = 32, maxit = 2500):
    beta_old = np.zeros(shape = (np.shape(init)[0],)) + 1
    beta_new = init
    likelihood = np.array([])
    l = 1000
    l_next = 1
    for i in range(1, maxit +1):
        l = l_next
        if i % 100 == 0 :
            print(l)
        grad = self.stoch_grad(beta_new, batch = batch_size)
        beta_old = beta_new
        beta_new = beta_old + step_size * grad
        l_next = self.loglikelihood(beta_new)
        likelihood = np.append(likelihood,l_next)
        if abs(l - l_next) / abs(l) < tol:
            break
    return beta_new, likelihood

def AdaGrad(self, init, tol = 1e-10, epsilon = 1e-8, step_size = 0.2, batch_size = 32,
            maxit = 2500):
    beta_new = init
    likelihood = np.array([])
    l = 1000
    l_next = 1
    grad_sum = 0
    for i in range(1, maxit +1):
        l = l_next
```

```
        if i % 100 == 0 :
            print(l)
        grad = self.stoch_grad(beta_new, batch = batch_size)
        grad_sum = grad_sum + np.square(grad)
        beta_new = beta_new + step_size * grad / np.sqrt(grad_sum + epsilon)
        l_next = self.loglikelihood(beta_new)
        likelihood = np.append(likelihood,l_next)
        if abs(l - l_next) / abs(l) < tol:
            break
    return beta_new, likelihood

def RMSprop(self, init, tol = 1e-10, epsilon = 1e-8, step_size = 0.01, batch_size = 32,
            maxit = 2500):
    beta_new = init
    likelihood = np.array([])
    l = 1000
    l_next = 1
    grad_sum = 0
    for i in range(1, maxit +1):
        l = l_next
        if i % 100 == 0 :
            print(l)
        grad = self.stoch_grad(beta_new, batch = batch_size)
        grad_sum = 0.9 * grad_sum + 0.1 * np.square(grad)
        beta_new = beta_new + step_size * grad / np.sqrt(grad_sum + epsilon)
        l_next = self.loglikelihood(beta_new)
        likelihood = np.append(likelihood,l_next)
        if abs(l - l_next) / abs(l) < tol:
            break
    return beta_new, likelihood

def Adam(self, init, tol = 1e-10, epsilon = 1e-8, step_size = 0.008, batch_size = 32,
         maxit = 2500):
    beta1 = 0.9
    beta2 = 0.999
    beta_new = init
    likelihood = np.array([])
    l = 1000
    l_next = 1
    grad_sum = 0
    grad_square = 0
    for i in range(1, maxit +1):
        l = l_next
        if i % 100 == 0 :
            print(l)
        grad = self.stoch_grad(beta_new, batch = batch_size)
        grad_sum = beta1 * grad_sum + (1 - beta1) * grad
```

```
        grad_square = beta2 * grad_square + (1 - beta2) * np.square(grad)
        beta1_power = beta1 ** (i + 1)
        beta2_power = beta2 ** (i + 1)
        mt = grad_sum / (1 - beta1_power)
        vt = grad_square / (1 - beta2_power)
        beta_new = beta_new + step_size * mt / np.sqrt(vt + epsilon)
        l_next = self.loglikelihood(beta_new)
        likelihood = np.append(likelihood,l_next)
        if abs(l - l_next) / abs(l) < tol:
            break
    return beta_new, likelihood
```

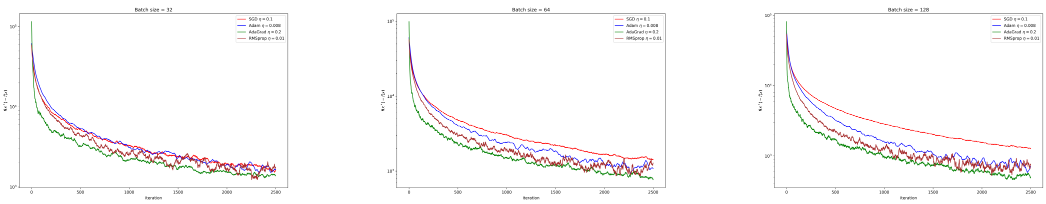

Figure 4.4: Comparison: batch size = 32, 64, 128

We can see that the adaptive learning rate methods can rapidly fall to a minimum value at the beginning of training, and this becomes more obvious when we enlarge batch size. Among the four, AdaGrad performs the best, while RMSProp fluctuates a lot. As the batch size enlarges, all of the adaptive learning rate methods can reach a lower value.

(3). Bonus question. Generate a random mask matrix M as follows and use it to sparsify the covariance matrix X

```
np.random.seed(1234)

sparse_rate = 0.3
M = np.random.uniform(size=(n,d)) < sparse_rate
X[M] = 0.
```

Repeat your experiments in (2), and compare with the results for the full covariance matrix

Solution.

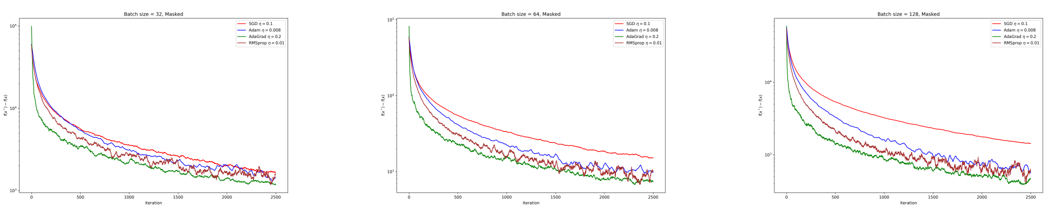

Figure 4.5: Comparison: batch size = 32, 64, 128

In this setting, the difference between SGD and its adaptive variants becomes more obvious, especially in the case of large batch size. Other observations are roughly the same as (2).