SOLUTION MANUAL
Math 189r
Homework 1
October 10, 2016

There are 6 problems in this set. You need to do 3 problems (due in class on Monday) every week for 2 weeks. Note that this means you must eventually complete all problems. Feel free to work with other students, but make sure you write up the homework and code on your own (no copying homework *or* code; no pair programming). Feel free to ask students or instructors for help debugging code or whatever else, though. When implementing algorithms you may not use any library (such as `sklearn`) that already implements the algorithms but you may use any other library for data cleaning and numeric purposes (`numpy` or `pandas`). Use common sense. Problems are in no specific order.

1. (Conditioning a Gaussian) Note that from Murphy page 113. "Equation 4.69 is of such importance in this book that we have put a box around it, so you can easily find it." That equation is important. Read through the proof of the result. Suppose we have a distribution over random variables $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ that is jointly Gaussian with parameters

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix},$$

where

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \mu_2 = 5, \ \boldsymbol{\Sigma}_{11} = \begin{bmatrix} 6 & 8 \\ 8 & 13 \end{bmatrix}, \ \boldsymbol{\Sigma}_{21}^{\top} = \boldsymbol{\Sigma}_{12} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}, \ \boldsymbol{\Sigma}_{22} = \begin{bmatrix} 14 \end{bmatrix}.$$

Compute

(a) The marginal distribution $p(\mathbf{x}_1)$. Plot the density in $\mathbb{R}^2$.

(b) The marginal distribution $p(\mathbf{x}_2)$. Plot the density in $\mathbb{R}^1$.

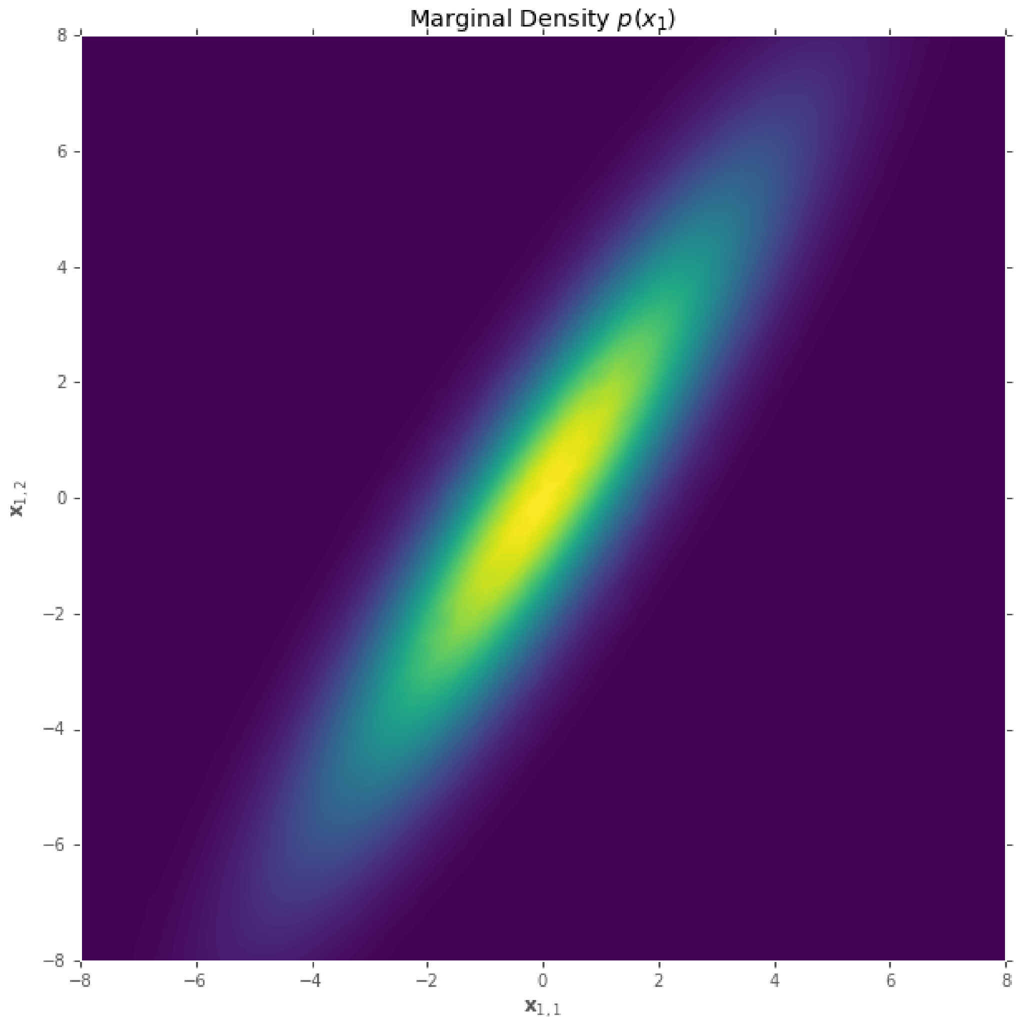
(c) The conditional distribution $p(\mathbf{x}_1|\mathbf{x}_2)$

(d) The conditional distribution $p(\mathbf{x}_2|\mathbf{x}_1)$

(a) We know

$$p(\mathbf{x}_1) = \mathcal{N}(\mu_1, \boldsymbol{\Sigma}_{11}) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 6 & 8 \\ 8 & 13 \end{bmatrix} \right). \tag{1}$$

We can vizualize the density as follows.



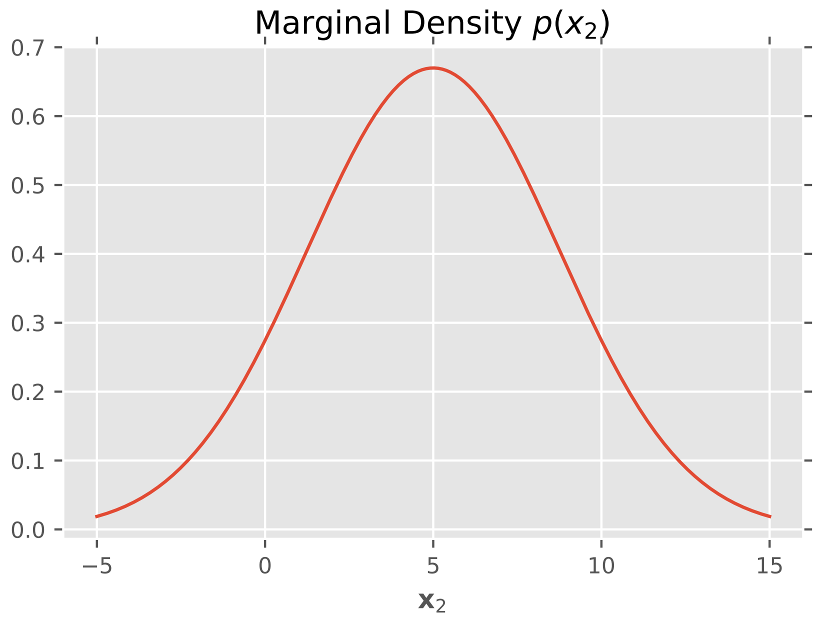
This used the following code (including the code for part (b) as well since they are so similar). Note that you can plot this with much less code, but making a nice looking 3D plot in Matplotlib is suprisingly difficult and requires a corresponding amount of elbow grease.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.tri as tri
import matplotlib.cm as cm

def normal_density(X, mu, cov):
    diff = X - mu
    return np.exp(-diff.T @ np.linalg.inv(cov) @ diff / 2) / (
        (2 * np.pi)**(-X.shape[0]/2) * np.sqrt(np.linalg.det(cov))
    )

a_mu = np.array([0,0])
a_cov = np.array([[6,8],[8,13]])
a_density = lambda x, y: normal_density(
    np.array([x,y]),
    a_mu, a_cov,
)

b_mu = 5
b_cov = np.array([[14]])
b_density = lambda x: normal_density(x.reshape(-1,1), b_mu, b_cov)

N = 15000
x = np.linspace(-12,12,N)
y = np.linspace(-12,12,N)
np.random.shuffle(x)
np.random.shuffle(y)

a_z = np.array([a_density(x_,y_) for x_,y_ in zip(x,y)])

b_x = np.linspace(-5,15,1000)
b_z = np.array([b_density(x) for x in b_x]).flatten()

n_levels = 100
levels = np.linspace(np.min(a_z), np.max(a_z), n_levels)

plt.figure(figsize=(10,10))
plt.gca().set_aspect(equal)
plt.tricontourf(x, y, a_z, levels=levels)
plt.xlim(-8,8)
plt.ylim(-8,8)
plt.title(rMarginal Density $p(x_1)$)
plt.xlabel(r$\mathbf{x}_{1,1}$)
plt.ylabel(r$\mathbf{x}_{1,2}$)
plt.savefig(/Users/cdipaolo/Courses/math189r/solutions/oct_10/1_a.pdf)
plt.savefig(/Users/cdipaolo/Courses/math189r/solutions/oct_10/1_a.png)

plt.plot(b_x, b_z)
plt.title(rMarginal Density $p(x_2)$)
plt.xlabel(r$\mathbf{x}_2$)
plt.savefig(/Users/cdipaolo/Courses/math189r/solutions/oct_10/1_b.pdf)
```

(b) We know

$$p(\mathbf{x}_2) = \mathcal{N}(\mu_2, \mathbf{\Sigma}_{22}) = \mathcal{N}(5, 14). \tag{2}$$

We vizualize this density below.



(c) From Equation 4.69 in Murphy we know

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}\left(\mu_{1|2}, \mathbf{\Sigma}_{1|2}\right) \tag{3}$$

where

$$\mu_{1|2} = \mu_1 + \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \mu_2) = \frac{1}{14}\begin{bmatrix}5\\11\end{bmatrix}(\mathbf{x}_2 - 5) \tag{4}$$

and

$$\mathbf{\Sigma}_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = \begin{bmatrix}6 & 8\\8 & 13\end{bmatrix} - \frac{1}{14}\begin{bmatrix}5\\11\end{bmatrix}\begin{bmatrix}5 & 11\end{bmatrix} = \begin{bmatrix}\frac{59}{14} & \frac{57}{14}\\\frac{57}{14} & \frac{61}{14}\end{bmatrix}. \tag{5}$$

(d) As above we have

$$p(\mathbf{x}_2|\mathbf{x}_1) = \mathcal{N}\left(\mu_{2|1}, \mathbf{\Sigma}_{2|1}\right) \tag{6}$$

where

$$\mu_{2|1} = \mu_2 + \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \mu_1) = 5 + \begin{bmatrix}5 & 11\end{bmatrix}\begin{bmatrix}6 & 8\\8 & 13\end{bmatrix}^{-1}(\mathbf{x}_1 - \mu_1) = 5 + \begin{bmatrix}-\frac{23}{14} & \frac{13}{7}\end{bmatrix}\mathbf{x}_1 \tag{7}$$

and

$$\mathbf{\Sigma}_{2|1} = \mathbf{\Sigma}_{22} - \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}\mathbf{\Sigma}_{12} = 14 - \begin{bmatrix}5 & 11\end{bmatrix}\begin{bmatrix}6 & 8\\8 & 13\end{bmatrix}^{-1}\begin{bmatrix}5\\11\end{bmatrix} = \frac{25}{14}. \tag{8}$$

2. (ℓ_1-Regularization) Consider the ℓ_1 norm of a vector $\mathbf{x} \in \mathbb{R}^n$:

$$\|\mathbf{x}\|_1 = \sum_i |\mathbf{x}_i|.$$

Plot the norm-ball $B_k = \{\mathbf{x} : \|\mathbf{x}\|_1 \leq k\}$ for $k = 1$. On the same plot, plot the Euclidean norm-ball $A_k = \{\mathbf{x} : \|\mathbf{x}\|_2 \leq k\}$ for $k = 1$ behind the first plot. Show that the optimization problem

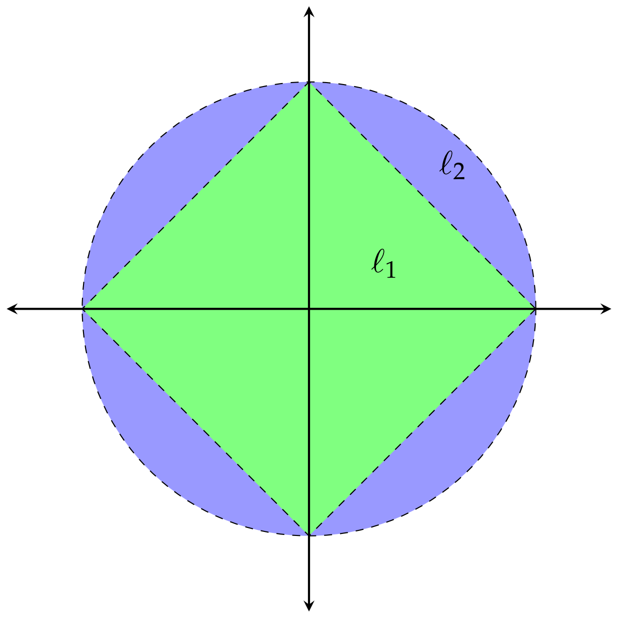
$$\begin{aligned} \text{minimize: } & f(\mathbf{x}) \\ \text{subj. to: } & \|\mathbf{x}\|_p \leq k \end{aligned}$$

is equivalent to

$$\text{minimize: } f(\mathbf{x}) + \lambda\|\mathbf{x}\|_p$$

(hint: create the Lagrangian). With this knowledge, and the plots given above, argue why using ℓ_1 regularization (adding a $\lambda\|\mathbf{x}\|_1$ term to the objective) will give sparser solutions than using ℓ_2 regularization for suitably large λ.

We see the norm balls below.



We know the optimization problem

$$\begin{aligned} \text{minimize: } & f(\mathbf{x}) && (9) \\ \text{subj. to: } & \|\mathbf{x}\|_p \leq k && (10) \end{aligned}$$

is equivalent to

$$\inf_{\mathbf{x}} \sup_{\lambda \geq 0} \mathcal{L}(\mathbf{x}, \lambda) = \inf_{\mathbf{x}} \sup_{\lambda \geq 0} f(\mathbf{x}) + \lambda(\|\mathbf{x}\|_p - k). \tag{11}$$

In its dual we can flip the inf and sup.

$$\sup_{\lambda \geq 0} \inf_{\mathbf{x}} f(\mathbf{x}) + \lambda(\|\mathbf{x}\|_p - k) = \sup_{\lambda \geq 0} g(\lambda) \tag{12}$$

Since the minimizing value of $f(\mathbf{x}) + \lambda(\|\mathbf{x}\|_p - k)$ over $\mathbf{x}$ is equivalent to the minimizing value of $f(\mathbf{x}) + \lambda\|\mathbf{x}\|_p$ ($-\lambda k$ doesn't depend on $\mathbf{x}$), we know the the optimizing $\mathbf{x}$ will solve

$$\text{minimize: } f(\mathbf{x}) + \lambda\|\mathbf{x}\|_p \tag{13}$$

for some suitable value of $\lambda \geq 0$. Looking at the plot and this result, we can consider ℓ_1 regularization as projecting the actual optimal solution of your problem onto some suitably sized ℓ_1 norm ball. Since the ℓ_1 ball has sharper edges, the probability of landing on an edge and not on the face (where both elements of the vector are nonzero) is infinitely larger than the ℓ_2 ball. This is due to the rotation invariance of the ℓ_2 that certainly doesn't hold for the ℓ_1 ball. Generalizing to higher dimensions, we can see that the ℓ_1 penalty will encourage more weights to be zero compared to the ℓ_2 ball, as desired.

---

3. (Lasso) Show that placing an equal zero-mean Laplace prior on each element of the weights $\boldsymbol{\theta}$ of a model is equivelent to ℓ_1 regularization in the Maximum-a-Posteriori estimate

$$\text{maximize: } \mathbb{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})}{\mathbb{P}(\mathcal{D})}.$$

Note the form of the Laplace distribution is

$$\text{Lap}(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right)$$

where μ is the location parameter and $b > 0$ controls the variance. Plot the density $\text{Lap}(x|0, 1)$ and the standard normal $\mathcal{N}(x|0, 1)$ and suggest why this would lead to sparser solutions than a Gaussian prior on each elements of the weights (which correspond to ℓ_2 regularization).

We know the Maximum-a-Posteriori problem

$$\text{maximize: } \mathbb{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})}{\mathbb{P}(\mathcal{D})}. \tag{14}$$

is equivalent to maximizing $\log \mathbb{P}(\boldsymbol{\theta}|\mathcal{D})$ given the monotonicity of $\log(x)$. This gives

$$\text{maximize: } \log \mathbb{P}(\boldsymbol{\theta}|\mathcal{D}) = \log \mathbb{P}(\mathcal{D}|\boldsymbol{\theta}) + \log \mathbb{P}(\boldsymbol{\theta}) - \log \mathbb{P}(\mathcal{D}). \tag{15}$$

Since $\mathbb{P}(\mathcal{D})$ is a constant not dependent on $\boldsymbol{\theta}$, we can drop that term from the problem and flip into a minimization problem, giving

$$\text{minimize: } -\log \mathbb{P}(\mathcal{D}|\boldsymbol{\theta}) - \log \mathbb{P}(\boldsymbol{\theta}). \tag{16}$$

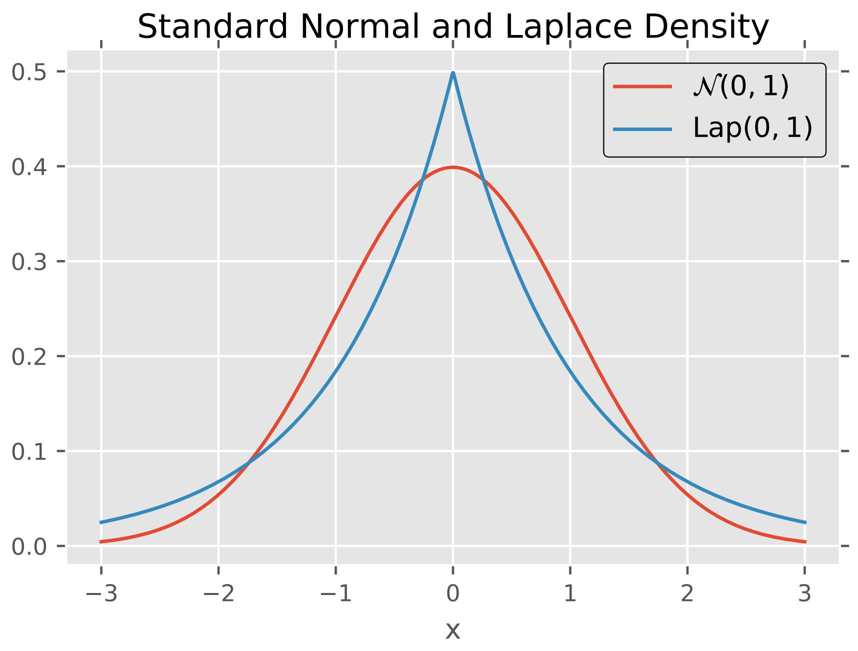
Given a prior $\boldsymbol{\theta}_i \sim Lap(0, b)$,

$$\begin{aligned} -\log \mathbb{P}(\boldsymbol{\theta}) &= -\log \prod_i \exp\left(-\frac{|\boldsymbol{\theta}_i|}{b}\right) + Z && \text{(where } Z \text{ is a constant)} \\ &= \frac{1}{b}\sum_i |\boldsymbol{\theta}_i| + Z && (17) \\ &= \lambda \|\boldsymbol{\theta}\|_1 + Z. && \text{(where } \lambda = 1/b\text{)} \end{aligned}$$

It follows that our original problem is equivalent to

$$\text{minimize: } -\log \mathbb{P}(\mathcal{D}|\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1, \tag{18}$$

or a ℓ_1 regularized maximum likelihood estimate, as desired. Note the plots of the Standard Normal and Laplace Densities.



We can see that $\text{Lap}(0, 1)$ will place much more mass at $x = 0$. It follows that when we use a Laplace prior instead of a Gaussian prior on our weights, our weights will be more encouraged to be exactly zero, forcing sparsity.

4. (Lasso Feature Selection) Ignoring undifferentiability at $x = 0$, take $\frac{\partial |x|}{\partial x} = \text{sign}(x)$. Using this, show that $\nabla \|\mathbf{x}\|_1 = \text{sign}(\mathbf{x})$ where sign is applied elementwise. Derive the gradient of the ℓ_1 regularized linear regression objective

$$\text{minimize: } \|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1$$

Now consider the shares dataset we used in problem 1 of homework 1 (`https://math189r.github.io/hw/data/online_news_popularity/online_news_popularity.txt`). Implement a gradient descent based solution of the above optimization problem for this data. Produce the convergence plot (objective vs. iterations) for a non-trivial value of λ. In the same figure (and different axes) produce a 'regularization path' plot. Detailed more in section 13.3.4 of Murphy, a regularization path is a plot of the optimal weight on the y axis at a given regularization strength λ on the x axis. Armed with this plot, provide an ordered list of the top five features in predicting the log-shares of a news article from this dataset (with justification). We can see a more detailed analysis of this at `https://en.wikipedia.org/wiki/Proximal_gradient_methods_for_learning` and `https://web.stanford.edu/~boyd/papers/pdf/prox_algs.pdf` but you will have to wrap the gradient descent step with a threshold function

$$\text{prox}_\gamma(\mathbf{x})_i = \begin{cases} \mathbf{x}_i - \gamma & \mathbf{x}_i > \gamma \\ 0 & |\mathbf{x}_i| \leq \gamma \\ \mathbf{x}_i + \gamma & x_i < -\gamma \end{cases}$$

so that each iterate

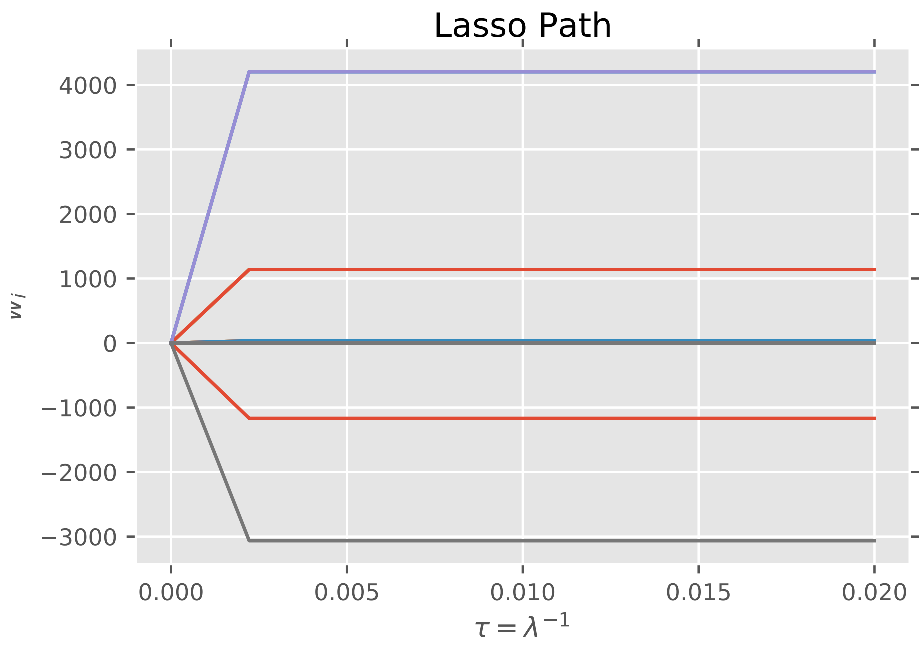
$$\mathbf{x}_{i+1} = \text{prox}_\gamma \left(\mathbf{x}_i - \gamma \nabla f(\mathbf{x}_i) \right)$$

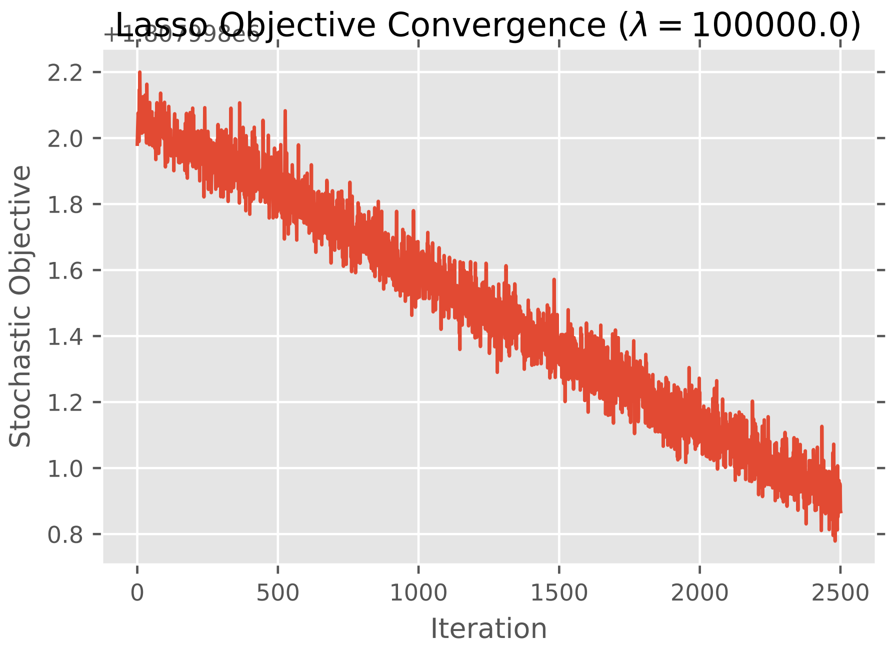
where γ is your learning rate. Tip: you can reuse most of your code from the first homework.

Clearly $\frac{\partial \|\mathbf{x}\|_1}{\partial \mathbf{x}_i} = \frac{\partial \sum |\mathbf{x}_i|}{\partial \mathbf{x}_i} = \text{sign}(\mathbf{x}_i)$. It follows that $\nabla \|\mathbf{x}\|_1 = \text{sign}(\mathbf{x}_i)$. We can then see

$$\begin{aligned} \nabla \|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1 &= \nabla \mathbf{x}^\top A^\top A\mathbf{x} - 2\mathbf{b}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{b} + \lambda \|\mathbf{x}\|_1 && (19) \\ &= 2A^\top A\mathbf{x} - 2\mathbf{b}^\top A + \lambda \text{sign}(\mathbf{x}). && (20) \end{aligned}$$

We can see plots and code below. Note that since we instantiated our weights with the least squares estimate, we can see our lasso objective not moving significantly, although if we look at sparsity over time it does in fact increase.

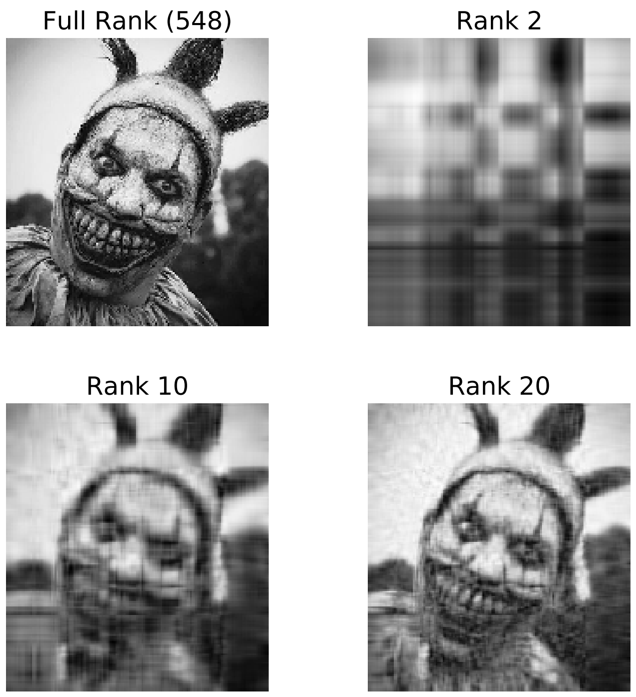


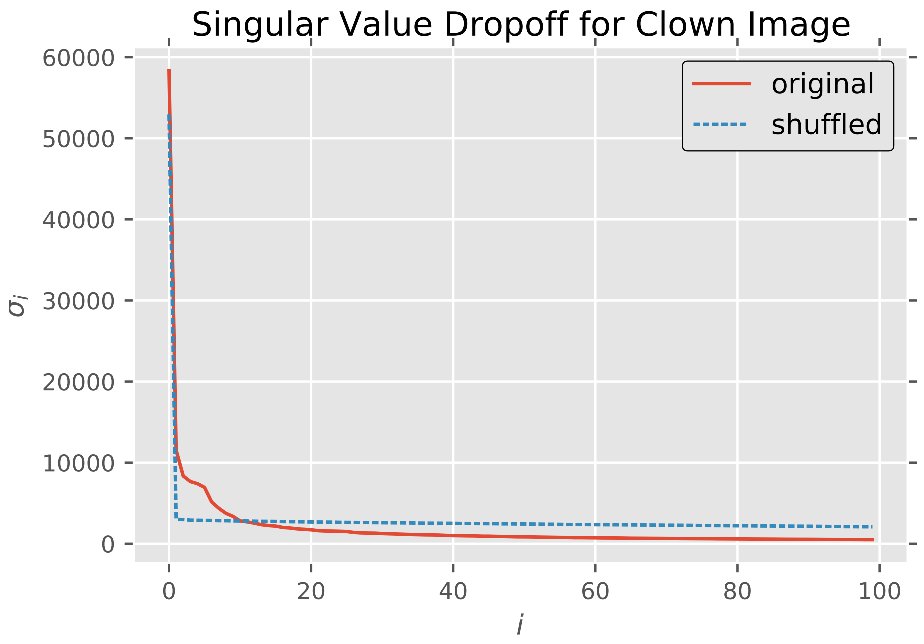


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv(https://math189r.github.io/hw/data/online_news_popularity/online_news_p
               sep=, , engine=python)
X, y = df[[col for col in df.columns if col not in [url, shares, cohort]]], \
           np.log(df.shares).reshape(-1,1)
X = np.hstack((np.ones_like(y), X))
```

```
def objective(X, y, w, reg=1e-6):
    err = X @ w - y
    err = float(err.T @ err)
    return (err + reg * np.abs(w).sum())/len(y)

def grad_objective(X, y, w):
    return X.T @ (X @ w - y) / len(y)

def prox(x, gamma):
     lasso proximal operator.
    Note: modifies x in-place

    x[np.abs(x) <= gamma] = 0.
    x[x > gamma] = x[x > gamma] - gamma
    x[x < -gamma] = x[x < -gamma] + gamma
    return x

def lasso_grad(
    X, y, reg=1e-6, lr=1e-3, tol=1e-6,
    max_iters=300, batch_size=256, eps=1e-5,
    verbose=False, print_freq=5,
):
    y = y.reshape(-1,1)
    w = np.linalg.solve(X.T @ X, X.T @ y)

    ind = np.random.randint(0, X.shape[0], size=batch_size)
    obj = [objective(X[ind], y[ind], w, reg=reg)]
    grad = grad_objective(X[ind], y[ind], w)

    while len(obj)-1 <= max_iters and \
          np.linalg.norm(grad) > tol:

        if verbose and (len(obj)-1) % print_freq == 0:
            print([i={}] objective: {}. sparsity = {:0.2f}.format(
                    len(obj)-1, obj[-1], (np.abs(w) < reg*lr).mean()
            ))

        ind = np.random.randint(0, X.shape[0], size=batch_size)
        grad = grad_objective(X[ind], y[ind], w)
        w = prox(w - lr * grad, reg*lr)
        obj.append(objective(X[ind], y[ind], w, reg=reg))

    if verbose:
        print([i={}] done. sparsity = {:0.2f}.format(
            len(obj)-1, (np.abs(w) < reg*lr).mean()
        ))
    return w, obj

def lasso_path(
    X, y, reg_min=1e-8, reg_max=10,
    regs=10, **grad_args
):
    W = np.zeros((X.shape[1], regs))
    tau = np.linspace(reg_min, reg_max, regs)
    for i in range(regs):
        W[:,i] = lasso_grad(
            X, y, reg=1/tau[i], max_iters=1000,
            batch_size=1024, **grad_args
        )[0].flatten()
    return tau, W

tau, W = lasso_path(X, y, reg_min=1e-15, reg_max=0.02, regs=10, lr=1e-12)
p = plt.plot(tau, W.T)
plt.title(Lasso Path)
plt.xlabel(r$\tau = \lambda^{-1}$)
plt.ylabel(r$w_i$)
plt.savefig(/Users/cdipaolo/Courses/math189r/solutions/oct_10/4_lasso_path.pdf)

# find most important features
np.array(df.columns)[np.argsort(-W[:,9])[:5]+1]
# [weekday_is_sunday, is_weekend, weekday_is_saturday,
# weekday_is_tuesday, weekday_is_friday]

lr = 1e-12
reg = 1e5
w, obj = lasso_grad(
    X, y, reg=reg, lr=lr, eps=1e-2,
    max_iters=2500, batch_size=1024,
    verbose=True, print_freq=250,
)
plt.title(rLasso Objective Convergence ($\lambda={}$).format(reg))
plt.ylabel(Stochastic Objective)
plt.xlabel(Iteration)
plt.plot(obj)
plt.savefig(/Users/cdipaolo/Courses/math189r/solutions/oct_10/4_convergence.pdf)
```

5. (SVD Image Compression) Load the image of a scary clown at `http://i.imgur.com/X017qGH.jpg` into a matrix/array. Plot the progression of the 100 largest singular values for the original image and a randomly shuffled version of the same image (all on the same plot). In a single figure plot a grid of four images: the original image, and a rank k truncated SVD approximation of the original image for $k \in \{2, 10, 20\}$.





```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import ndimage
import urllib
plt.style.use(ggplot)

img = ndimage.imread(
    urllib.request.urlopen(http://i.imgur.com/X017qGH.jpg),
    flatten=True,
)
shuffle_img = img.copy().flatten()
np.random.shuffle(shuffle_img)
shuffle_img = shuffle_img.reshape(img.shape)

_, s, _ = np.linalg.svd(img)
_, s_, _ = np.linalg.svd(shuffle_img)
k = 100
plt.plot(s[:k], label=original)
plt.plot(s_[:k], --, label=shuffled)
plt.legend()
plt.title(Singular Value Dropoff for Clown Image)
plt.ylabel(r$\sigma_i$)
plt.xlabel(r$i$)
plt.savefig(5_singular_values.pdf)

U_, s_, V_ = np.linalg.svd(img)
S_ = np.zeros((U_.shape[0],V_.shape[0]))
S_[:V_.shape[0], :V_.shape[0]] = np.diag(s_)

plt.figure(figsize=(6,6))
plt.subplot(2,2,1)
plt.imshow(U_ @ S_ @ V_, cmap=Greys_r)
plt.title(Full Rank ({}).format(max(img.shape[0], img.shape[1])))
plt.axis(off)

ranks = [2,10,20]
for i, k in enumerate(ranks):
    plt.subplot(2,2,i+2)
    plt.imshow(U_[:,:k] @ S_[:k,:] @ V_, cmap=Greys_r)
    plt.title(Rank {}.format(k))
    plt.axis(off)

plt.tight_layout()
plt.savefig(5_clowns.pdf)
```

6. (Murphy 12.5 - Deriving the Residual Error for PCA) It may be helpful to reference section 12.2.2 of Murphy.

(a) Prove that

$$\left\|\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right\|^2 = \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j.$$

Hint: first consider the case when $k = 2$. Use the fact that $\mathbf{v}_i^\top \mathbf{v}_j$ is 1 if $i = j$ and 0 otherwise. Recall that $z_{ij} = \mathbf{x}_i^\top \mathbf{v}_j$.

(b) Now show that

$$J_k = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \lambda_j.$$

Hint: recall that $\mathbf{v}_j^\top \mathbf{\Sigma} \mathbf{v}_j = \lambda_j \mathbf{v}_j^\top \mathbf{v}_j = \lambda_j$.

(c) If $k = d$ there is no truncation, so $J_d = 0$. Use this to show that the error from only using $k < d$ terms is given by

$$J_k = \sum_{j=k+1}^{d} \lambda_j.$$

Hint: partition the sum $\sum_{j=1}^{d} \lambda_j$ into $\sum_{j=1}^{k} \lambda_j$ and $\sum_{j=k+1}^{d} \lambda_j$.

(a) We know

$$\left\|\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right\|_2^2 = \left(\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \tag{21}$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i - \mathbf{x}_i^\top \sum_{j=1}^{k} z_{ij}\mathbf{v}_j + \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \tag{22}$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \qquad \text{(bringing } \mathbf{x}_i^\top \text{ into sum)}$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \sum_{j=1}^{k} \mathbf{v}_j^\top z_{ij}^\top z_{ij}\mathbf{x}_j \qquad \text{(since } \mathbf{v}_i^\top \mathbf{v}_j = 1 \text{ iff } i = j\text{)}$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i\mathbf{x}_i^\top \mathbf{x}_j \qquad (z_{ij}^\top z_{ij} = \mathbf{x}_i\mathbf{x}_i^\top)$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i\mathbf{x}_i\mathbf{v}_j^\top + \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i\mathbf{x}_i^\top \mathbf{x}_j \qquad \text{(since } z_{ij} \in \mathbb{R}\text{)}$$

$$= \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i\mathbf{x}_i^\top \mathbf{v}_j, \tag{23}$$

as desired.

(b) By definition

$$J_k = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i\mathbf{x}_i^\top \mathbf{v}_j\right) \tag{24}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \frac{1}{n}\left(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top\right)\mathbf{v}_j \tag{25}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \boldsymbol{\Sigma}\mathbf{v}_j \tag{26}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \lambda_j \tag{27}$$

as desired.

(c) Since $J_d = 0$ we know $\sum_{j=1}^{d}\lambda_j = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^\top \mathbf{x}_i$. Then

$$J_k = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{d}\lambda_j + \sum_{j=k+1}^{d}\lambda_j = \sum_{j=k+1}^{d}\lambda_j. \tag{28}$$

This is an exciting result. This states that the reconstruction error when using a PCA projection of your data is exactly equal to the sum of the eigenvalues you throw out.