Some of the question below have instruction in **bold** that require you to make plots or answer questions. Create a short pdf document called "hmwk9.pdf" with your results for each question. You don't need to make an elaborate writeup, just put your plots into the pdf with labels indicating what they are, and answer questions when prompted. **Turn in both your pdf and your source code on the submit server.**

This homework covers topics in forward backward splitting. For a general reference, see the review article "A field guide to forward backward splitting with a FASTA implementation," or the lecture slides.

1. (a) (matlab users only)
   Download the FASTA solver (https://www.cs.umd.edu/ tomg/projects/fasta/). Skim the first 2 pages of the users manual. This solver performs general forward backward splitting with a user-supplied gradient and proximal operator.

   (b) (python users only) Write a method that uses FBS to solve a general problem of the form

   $$\text{minimize} \quad f(x) + g(x).$$

   You can do this by modifying your `grad_descent` code from homework 2. Your new method should have signature

   ```
   def  fbs(f, gradf, proxg, x0):
             ...
   ```

   The argument `f` computes the scalar-valued function $f(x)$. The argument `gradf` is a function handle that computes the gradient of f. This means that

   $$\texttt{gradf}(x) = \nabla f(x).$$

   The argument `proxg` is a function handle that computes the proximal operator of g with stepsize τ. This means that

   $$\texttt{proxg}(z, \tau) = \arg\min_x g(x) + \frac{1}{2\tau}\|x - z\|^2$$

   Your method should start by estimating the initial stepsize τ using the Lipschitz constant for the gradient of f. You already did this in homework 2 in your `grad_descent` method. The method should then perform an iteration of FBS, and use a backtracking line search until the following condition is met:

   $$f(x^{k+1}) \le f(x^k) + \langle x^{k+1} - x^k, \nabla f(x^k)\rangle + \frac{1}{2\tau}\|x^k - x^{k+1}\|^2.$$

   Your method should terminate when the relative residual is "small." Formulas for the line search and residuals can be found in the paper "A field guide to forward backward splitting with a FASTA implementation," (the line search condition is discussed in section 4.4, and formulas for the residuals are in section 4.6) and also in the lecture slides.

2. Use the FASTA code (or your python solver) to solve the sparse logistic regression problem

   $$\text{minimize} \quad \mu|x| + \text{logreg\_objective}(x, \hat{D}, c)$$

where `logreg_objective` is the method you wrote in homework 1. Build the classification problem by choosing $[D, c] = \text{create\_classification\_problem}(500,\ 5,\ 1)$, and then append a bunch of random columns to D to form $\hat{D} = [D|G]$ where G is a matrix of 5 random Gaussian columns. Then, rescale the columns of G so they all have unit norm.

Pick some reasonable value of μ that generates a sparse solution, and solve the problem using FBS. To do this, you will need to call your FBS solver with $g(x) = \mu|x|$.

Plot your (sparse) solution, and plot the residual as a function of iteration number.

3. Build two random binary 50×3 matrices, A and B. Half the entries should be 1 and others are 0. Compute $D = AB^T$. Now, write a method that tries to recover the original matrices by solving the non-convex "non-negative matrix factorization" problem

$$\underset{X,Y}{\text{minimize}} \quad \frac{1}{2}\|XY - D\|^2 \ \text{ subject to } \ 0 \le X_{i,j} \le 1, 0 \le Y_{i,j} \le 1$$

using FBS. **Plot the residuals as a function of iteration number. Does the recovery method work? Justify your answers by describing exactly what you did to determine these things.**. Note: This is a non-convex problem and so the initialization matters! If you start with all zeros, then the gradients will also be zero, and the optimizer will be stuck.

4. Build a 128×128 test image. Then build a 128×128 Gaussian blur stencil with $\sigma = 3$. You can do this in matlab with something like this

```
blur = fspecial('gaussian',128,3);
blur = fftshift(blur);
```

or in Python with a function like this:

```
def matlab_style_gauss2D(shape=(128,128),sigma=3):
        m,n = [(ss-1.)/2. for ss in shape]
        y,x = np.ogrid[-m:m+1,-n:n+1]
        h = np.exp( -(x*x + y*y) / (2.*sigma*sigma) )
        h = h/h.sum()
        h = np.fft.fftshift(h)
        return h
```

Blur your image by convolving with the stencil to get a blurry image b. Now, recover a restored image by solving

$$\min_x \mu|\nabla x| + \frac{1}{2}\|Kx - b\|^2$$

where μ is an appropriately chosen scalar, and K is a linear operator that convolves with your Gaussian stencil. Your solver must use an ADMM approach with Lagrangian

$$L(x, y, \lambda) = \mu|y| + \frac{1}{2}\|Kx - b\|^2 + \langle y - \nabla x, \lambda\rangle + \frac{\tau}{2}\|y - \nabla x\|^2$$

Choose τ so that the average eigenvalue of $\tau\nabla^T\nabla$ is the same as the average eigenvalue of $K^T K$. **Plot the residual $\|y - \nabla x\|$ on a logarithmic y-axis. Show your original, blurred, and deblurred images.**. Note that this is a convex problem, and so it doesn't matter how you initialize.