Just like before, your homework submission must contain the following to receive full credit:

- A single script called "hmwk5.m" or "hmwk5.py" that requires no arguments and prints the things (and makes the plots) I ask for in bold below. Each output should be labeled.
- A pdf called hmwk5_results.pdf with your solutions to all theory problems and also all the PLOTS generated in the hmwk5 script. See the posted homework example.
- As many other code files as you wish - but only things you wrote or solutions to previous homeworks.

1. Consider the quadratic image denoising model

$$\text{L2 denoising:} \quad \min \quad \frac{\mu}{2}\|\nabla x\|^2 + \frac{1}{2}\|x-b\|^2$$

where μ is a scaling parameter and b is a noisy image. We use $\|\cdot\|$ to denote the ℓ_2/Frobenius norm. Note that ∇ denotes the discrete gradient (i.e., your `grad2d` function). For this problem, you should choose b to be a noisy, black-and-white test image of your choice, such as Barbara, Lena, the Shepp-Logan phantom, etc... Scale the image to have pixels between 0 and 1, and set $\mu = 2$.

(a) **Write the optimality condition for this problem by taking the gradient and setting it equal to zero. Now, put the optimality condition in the form of a large linear system $Ax = b$.**

(b) Create a function or function handle that computes the linear operator A from part (a). The only argument to this function should be a 2d pixel array representing x. This can be done with only 1 line of code when using your `grad2d` and `div2d` functions. Both the inputs and the outputs of your function MUST be 2d arrays.

(c) Write a function with signature

```
function x, resids = richardson(A,b,x,t)
```

that solves the system $Ax = b$ using Richardson iteration with stepsize `t`, where A is *any* arbitrary handle/function that computes a symmetric positive definite linear operator. The method must accept an initial starting point `x` of arbitrary dimensions (i.e., 1d, 2d, 3d, etc...). The method should terminate when the residual norm falls below 10^{-6}. The method returns the solution, `x`, and also a vector containing the residual norms on every iteration. Use your code to minimize the L2 denoising energy above. **Make a plot of residual norms vs iteration number. Display the noisy image beside the denoised image. Report how many iterations this took.**

(d) Write a function

```
function x, resids = conjgrad(A,b,x)
```

that implements the conjugate gradient method for solving $Ax = b$ where A is a symmetric positive definite linear operator. The method must accept an initial guess `x` of arbitrary dimensions, and return the solution to the system in addition to a vector of residuals. Use this routine to solve the L2 denoising problem. **Make a plot of residual norms vs iteration number. Display the noisy image beside the denoised image. Report how many iterations this took.**

(e) Now, set $\mu = 10$, and re-run your experiments. **Show your images and convergence plots. What happened to your iteration counts? Why did this happen? Make a mathematical argument with some equations.**

(f) Write a function

```
function x = l2denoise(b,mu)
```

that directly computes the exact solution (up to numerical rounding errors) of the denoising problem using Fourier transforms (i.e., without using an iterative method). Your implementation can call `fft2`/`ifft2` at most 4 times total, and you may not use any loops of any kind. **Call your method to solve the denoising problem, and evaluate and report the norm of the gradient of the objective function. This value should be very small.**

2. In this problem you'll do spectral clustering on a toy dataset. Download the "two moons" dataset generator from the website.

(a) Build a dataset with 100 points. Scatterplot the point and have a look at them. Pick some reasonable value of σ (the length scale for the problem), and build the 100×100 pairwise similarity matrix S with

$$S(i,j) = exp(-\|x_i - x_j\|/\sigma).$$

Your value of σ should be small enough that the two moons are "far apart" on this distance scale.

(b) Compute the diagonal normalization matrix with $D(i,i) = \sum_j S(i,j)$. This matrix just contains the row sums of the similarity matrix. Use this matrix to compute the normalized similarity matrix

$$\hat{S} = D^{-1/2} S D^{-1/2}.$$

Note that we split the normalization matrix up and apply it to both sides of S to preserve symmetry.

(c) Compute the eigenvalue decomposition of $\hat{S}$, and select the second eigenvector u_2 (i.e., the eigenvector with second largest eigenvalue). This is the "spectral embedding" of your data points. **Plot the vector** u_2. The two moons should now look linearly separable (expect for maybe a few points).

(d) Run k-means clustering on the entries in u_2 (I suggest you use the `kmeans` command in Matlab, and `sklearn.cluster.KMeans` in Python) to produce 2 clusters. **Plot the original datapoints with two different colors to indicate which class the datapoint was assigned by k-means. You should have separated the moons almost perfectly.**

3. Repeat Question 2, but this time with 100,000 datapoints. The process outlined below is described in detail in the paper "Spectral Grouping Using the Nystrom Method."

   (a) Build a Nystrom approximation of S of the form $S_{ny} = CW^{-1}C^T \approx S$ by sampling 200 randomly chosen columns of S.

   (b) We can't compute row sums of S because it's too big to form explicitly, so instead compute the row-sums of S_{ny}, and define the diagonal matrix $D(ii) = \sum_j S_{ny}(i,j)$. Normalize the rows/columns you sampled to find the matrix C_n containing 200 column samples from the normalized matrix $\hat{S}_{ny} = D^{-1/2}SD^{-1/2}$. Define W_n by sampling the corresponding rows from C_n. Define the matrix M_n comprising all the rows of C_n corresponding to un-sampled column indices.

   (c) Compute the approximate eigenvectors of $\hat{S}_{ny}$. Do this in stages:

$$\begin{aligned}
\text{Compute orthogonalization matrix: } & \hat{W} = W_n + W_n^{-1/2} M_n^T M_n W_n^{-1/2} \\
\text{Compute eigen-decomposition: } & U_W D_W U_W^T = \hat{W} \\
\text{Compute approximate eigenvectors: } & U = C_n W_n^{-1/2} U_W D_W^{-1/2} \\
\text{Rescale: } & U(:,k) \leftarrow U(:,k)./U(:,200), \text{ for } k = 1,2,\cdots,200.
\end{aligned}$$

   The factorization $\hat{S}_{ny} = UD_W U^T$ is now an eigenvalue decomposition. Finally, I recommend that you perform this rescaling step:

$$\text{Rescale: } U(:,k) \leftarrow U(:,k)./U(:,200), \text{ for } k = 1,2,\cdots,200.$$

   Note, the last step of the process ("Rescale") divides every non-leading eigenvector by the leading eigenvector, i.e., I assume your vectors are ordered so that $U(:,200)$ is the vector of largest eigenvalue.

   Let U_2 contain the 2nd most significant eigenvector. **Plot** U_2. The two moons should now be linearly separable.

   (d) Do k-means on U_2. **Plot the original datapoints with two different colors to indicate which class the datapoint was assigned by k-means. You should have separated the moons almost perfectly.**