CSE 559A: Fall 2020

Problem Set 2

Due: Oct 27, 2020. 11:59 PM

Instructions

Please read the late submission and collaboration policy on the course website:
http://www.cse.wustl.edu/~ayan/courses/cse559a/

Install Anaconda for Python 3.6+ from https://www.anaconda.com/download. We will test all code on this distribution. You can install it locally in a separate directory without interfering with your system install of Python.

1. Complete the code files in `code/` by filling out the required functions.
2. Run each program to generated output images in `code/outputs` directory.
3. Create a PDF report in `solution.pdf` with LaTeXby editing `solution.tex`. In particular, make sure you fill out your name/wustl key (top of the .tex file) to populate the page headers. Also fill out the final information section describing how long the problem set took you, who you collaborated / discussed the problem set with, as well as any online resources you used.
4. The main body of the report should contain responses to any math questions, and also include results and figures for the programming questions as asked for. These figures will often correspond to images generated by your python code in the `code/outputs/` directory.
5. Once you are done, "git add" the completed `solution.pdf` and your updated code files in `code/*.py`. Please do not add the generated output images, as these are already in your report (the git repo is setup to ignore those files). Then do a "git commit", and a "git push". Then, do a "git pull", and a "git log" to verify the timestamp of your submission and the files included. These instructions are also explained in the "problem-sets" section of the course website.

As a general guideline for all problem sets: Write efficient code. While most of the points are for writing code that is correct, some points are allocated to efficiency. Above all, try to minimize the total number of multiplies / adds. For the same number of underlying operations, try to keep the use of `for` loops to a minimum (i.e., over a minimum number of indices). Instead, use convolution, element-wise operations over large arrays, calls to matrix multiply, etc.

PROBLEM 1 **(Total: 15 points)**

A traditional approach to image denoising is to apply what is called $L1$ regularization to wavelet coefficients. Since the wavelet transform is unitary, additive un-correlated Gaussian noise to image pixel intensities shows up as un-correlated Gaussian noise, with the same variance, in wavelet coefficients. If our regularization is applied independently to wavelet coefficients, then we can carry out independent minimizations for each coefficient.

(a) What value of the scalar variable x minimizes the following cost function:

$$x = \arg\min_x (x - y)^2 + \lambda |x|$$

Here, y is also a scalar corresponding to the noisy observation, λ is a parameter, and $|x|$ refers to the absolute value of x. Find the expression for x in terms of y and λ. Remember that the gradient of $|\cdot|$ is *discontinuous* at 0, and changes sign. **(10 points)**

(b) Implement this expression in the function `denoise_coeff` in `prob1.py`. The function will be called with an array of "noisy" coefficients corresponding to y above, and a value of λ. Return an array of the corresponding clean coefficients, based on the above expression (applied to each coefficient independently).

You will also need to fill out the `im2wv` and `wv2im` functions from problem 6 in PSET1. (If you couldn't get that to work, we'll provide you with a working version of those routines). The support code handles the task of loading a noisy image, doing the pyramid decomposition, calling the `denoise_coeff` function with the 'right' value of λ for each level, and saving the denoised image in `outputs/prob1.png`. Include the denoised image in your report. Feel free to play around with modifying the values of λ being used by the support code. **(5 points)**

PROBLEM 2 **(Total: 15 points)**

In this problem, we will implement two strategies to doing color constancy, or white balance in the file `prob2.py`. In each case, you will correct an image by multiplying the red, green, and blue channels with scalars α_r, α_g, and α_b, which will be normalized so that $\alpha_r + \alpha_g + \alpha_b = 3$.

(a) *Gray World*: Fill out the `balance2a` function to do white balance with factors α_r, α_g, and α_b computed to be inversely proportional to the mean red, blue, and green intensity in the image. Running `prob2.py` will generate white-balanced versions `outputs/prob2a_?.png` of the images in `inputs/CC/`. Include these images in your report. **(5 points)**

(b) Implement the `balance2b` function to now compute the factors α_r, α_g, and α_b to be inversely proportional to averages computed over the 10% brightest intensities in each channel (i.e., α_r is inversely proportional to the average of the 10% highest red intensity values, and so on). The function `np.sort` may be useful here. Run `prob2.py` to get the corresponding versions `outputs/prob2b_?.png`. Include these in your report, and comment on how they differ from the results from part (a). **(10 points)**

PROBLEM 3 **(Total: 20 points)**

Implement photometric stereo as discussed in class for Lambertian surfaces with known light from point sources at infinity.

(a) Fill out the `pstereo_n` function in `prob3.py` to compute surface normals given a set of color images, and a matrix

of lighting directions. As described in class, do this by first converting the images to grayscale (you can simply take mean or sum of R,G, and B values) and then solving for the set of $\ell_i^T n = I_i = (R_i + G_i + B_i)$ equations at every pixel. For this part, you are allowed to use `np.linalg.solve`, but DO NOT use `linalg.lstsq` or any pre-programmed least-squares solver. Implement it yourself. (Note also that the function is provided with a mask of where pixels are valid. Outside the valid region, you may get division by zero errors when you try to normalize the normal vectors, etc. Be sure to avoid this.)

Include the saved image of your estimated normals `outputs/prob3_nrm.png` in your report. **(10 points)**

(b) Fill out the `pstereo_alb` to now compute an image of the surface color *albedos*, given the computed normals from the previous step. (This is your solution for $\rho_R \ell_i^T n = R_i$, and so on.) The results of this will be saved as `outputs/prob3_alb.png`. Include it in your report. **(10 points)**

PROBLEM 4 (Total: 25 points)

Implement the `ntod` function in `prob4.py` to compute a depth map, i.e. an array of surface height values Z, from estimated surface normal vectors, in the Fourier domain using the Frankot-Chellappa method as described in class. Your function will be called with the array of normal unit vectors, a mask where these values are valid, and regularization weight. Set the $x-$ and $y-$ derivatives to $-\hat{n}_x/\hat{n}_z$ and $-\hat{n}_y/\hat{n}_z$ where the normals are valid, and to 0 where they are not (i.e., where the mask is 0). Use the same regularizer as described in class (in terms of the f_r filter) weighted by the regularization parameter λ passed to the function.

Running the program will generate a python plot window with a 3D surface plot of the surface. Rotate it interactively to what you think is a good view, export the figure as a png (click on the Save button on your plot window), and include it in your report.

PROBLEM 5 (Total: 25 points)

Implement the `ntod` function in `prob5.py`, now using conjugate gradient. This function takes the same parameters as in `prob5.py` and minimizes a similar cost, except that the squared penalties for the derivatives are now weighted differently at each pixel as discussed in class. The weight is set to 0 at invalid pixels (where the mask is 0), and to $(\hat{n}_z)^2$ everywhere else. Note that there is no per-pixel weighting on the regularizer—just λ, the third parameter passed to the function, at all pixels.

Once your function is working, setup your code to run conjugate gradient for at least a 100 iterations (you can use fewer iterations while debugging). Like for the previous problem, running the support code produces a 3D surface plot. Export this figure as well, and include it in your report.