

Homework 2

Due: 9 Oct 2026

(late homeworks penalized 10% per day)

See the course web site for submission details. Reminder: rather than using the functions `pinv()` and `norm()`, use the linear algebra tools we learned in class. Please: don't wait until the day before the due date... start *now*!

1. **Trichromacy.** Load the file `colMatch.mat` in your MATLAB environment (or use `scipy.io.loadmat` for Python). This file contains matrices and vectors related to the color matching experiment presented in class. The variable `P` is an $N \times 3$ matrix containing wavelength spectra for three “primary” lights, to be used in a color-matching experiment. For these problems $N = 31$, corresponding to samples of the visible wavelength spectrum from 400 nm to 700 nm in increments of 10 nm.

The function `humanColorMatcher.p` simulates a normal human observer in a color matching experiment. For Python, download the file and use `from trichromacy import human_color_matcher`. The input variable `light` should contain the wavelength spectrum of a test light (a 31-dimensional column vector). The input variable `primaries` should contain the wavelength spectra of a set of primary lights (typically, a 31×3 matrix, as for matrix `P` described above). The function returns a 3-vector containing the observer's “knob settings” - the intensities of each of the primaries that, when mixed together, appear identical to the test light. The function can also be called with more than one test light (by passing a matrix whose columns contain 31-dimensional test lights), in which case it returns a matrix whose columns are the knob settings corresponding to each test light.

- (a) Create a test light with an arbitrary wavelength spectrum, by generating a random column vector with 31 positive components (use `rand` in MATLAB or `np.random.rand` in Python). Use `humanColorMatcher` to “run an experiment”, asking the “human” to set the intensities of the three primaries in `P` to match the appearance of the test light. Compute the 31-dimensional wavelength spectrum of this combination of primaries, plot it together with the original light spectrum, and explain why the two spectra are so different, even though they appear the same to the human.
- (b) Now characterize the human observer as a linear system that maps 31-dimensional lights to 3-dimensional knob settings. Specifically, run a set of experiments to estimate the contents of a 3×31 color-matching matrix `M` that can predict the human responses. Verify on a few random test lights that this matrix exactly predicts the responses of the function `humanColorMatcher`.
- (c) The variable `Phosphors` contains the emission spectra of three standard color display phosphors (from an old-fashioned cathode ray tube!). Suppose you wanted to make the background color of this screen match the appearance of an arbitrary test light. Write a matlab expression to compute the three phosphor intensities that would achieve this. Verify that this particular mixture of phosphor spectra satisfies the “matching” criterion (i.e., that a human would see this spectral mixture as being identical to the test light).

- (d) The function `altHumanColorMatcher(light, primaries)` simulates a color-deficient human observer in a standard color matching experiment (for Python: `from trichromacy import alt_human_color_matcher`). (i) For a random test light, compare the knob settings for this observer with those of the normal human. Do this for several runs. How do they differ? (ii) The variable `Cones` contains (in the rows) approximate spectral sensitivities of the three color photoreceptors (cones) in the human eye: `Cones(1,:)` is for the L (long-wavelength, or *red*) cones, `Cones(2,:)` the M (green) cones, and `Cones(3,:)` the S (blue) cones (for Python users, the indexing starts from 0). Applying the matrix `Cones` to any light $\vec{l}$ yields a 3-vector containing the average number of photons absorbed by each cone (try `plot(Cones')` to visualize them!). Compute cone absorptions for the test light, and for the mixture of three matching primaries (by applying the `Cones` matrix). Do this for both the normal human observer, and for multiple runs of the abnormal observer. Try this for several different test lights. How do the cone responses of the normal and abnormal observers differ? Can you offer a diagnosis of the underlying cause of color deficiency in the abnormal observer? Explain your reasoning.
2. **2D polynomial regression.** Load the file `regress2.mat` into your MATLAB environment. The matrix D contains 3 columns of data, which we'll refer to as X , Y , and Z respectively. The corresponding elements of these vectors, (X_k, Y_k, Z_k) , represent 3D data points. X and Y are uniformly distributed on a square grid.
- plot Z as a function of X and Y (using `surf` in MatLab, or `plot_surface` in python). To use the mouse to rotate the 3D space and view the data from different angles, execute `rotate3d on` in MatLab, or placing `%matplotlib widget` at the top of your Juypiter notebook. Note: you'll need to reshape the three column vectors into square matrices.
 - Fit the Z values with polynomials in X and Y , up to order 3: $p_0(X, Y) = \beta_0$, $p_1(X, Y) = \beta_0 + \beta_1 X + \beta_2 Y$, $p_2(X, Y) = \beta_0 + \beta_1 X + \beta_2 Y + \beta_3 X^2 + \beta_4 XY + \beta_5 Y^2$, etc. Compute this using `svd` and basic linear algebra manipulations that you've learned in class!
 - For each of the polynomials, (a) plot the fitted surface (use `surf`) and data points (use `plot3`) in the same figure, and rotate it around to convince yourself that the fit is reasonable. (b) compute the error for each element of Z , plot a histogram of these values, and compute the mean of the squared errors. How does the error behave as you increase the order of the polynomial? Which polynomial do you think gives the "best" fit? Explain.
3. **Trimmed regression.** One of the limitations of least-squares regression is sensitivity to outliers. As described in class, a common solution is to iteratively discard the bad points. Load the file `regress3.mat`. First solve the standard regression problem, using a constant and a linear term. Then write a loop that locates the data point with the largest magnitude error (use function `max`), eliminates that row from the data vector and regressor matrix, and then re-solves the regression problem. Note that you can also do this by including a diagonal weight matrix, and zeroing the entry corresponding to the outlier. On each iteration, plot the points and the best-fitting regression line, plot a histogram (use `histogram` in MATLAB or `plt.hist` in Python) of the squared errors over all points, and record the average over these errors (i.e., on the n th iteration, save the average error in the n th element of the vector). You may want put the two plots in separate figure windows, and you may want to pause after each iteration (use `pause` in MATLAB or `time.sleep` in Python). Run the loop until half of the data points have been discarded.

After running, plot the vector of average errors, as a function of iteration, in a third figure. Based on this, and the histograms you observed, which iteration gave the “best” fit? To visualize this solution, plot the corresponding regression line, and *all* of the data points, labeling the discarded data points with a different plot symbol. Did you make a good choice?

4. **Principal components of neural population trajectories.** Load the file `PCA.mat` into your MATLAB or Jupyter notebook environment. You’ll find a matrix M whose columns contain mean spike counts of 40 neurons from motor cortex, measured over 50 temporal intervals (100msec each), recorded while a monkey makes a stereotypical reaching movement.
 - (a) Plot all of the cell responses, as a function of time, on a single graph. This will look pretty messy, but can you see any systematic behaviors amongst the cells?
 - (b) Compute the principal components of these temporal trajectories. First, center the data by subtracting the mean response $\text{mean}(M)$ from every row of the matrix (hint: you might find the function `repmat` or `np.tile` helpful). Call this re-centered data matrix $\tilde{M}$. Then compute the eigenvectors and eigenvalues of $\tilde{M}\tilde{M}^T$ (alternatively, you can compute the singular values of $\tilde{M}$). Plot the square root of the eigenvalues (or the singular values). What do you think is the “true” dimensionality of the responses?
 - (c) Plot the significant eigenvectors as functions of time (all on a single plot). These should look “well-behaved” - verify that the next eigenvector (the first “insignificant” one) is less well behaved.
 - (d) Plot the *multi-dimensional* temporal trajectory mapped out by the significant eigenvectors (i.e., plot a set of 50 connected points in a 2D or 3D space, using `plot` or `plot3`, respectively). Describe what you see.