Course reader: *Image signals*

- Two-dimensional data are more than just a dimensional extension of 1D data; there is an explosion of information that you get from a 2D plane compared to a 1D line. Just think about how much information is packed into this 2D text, as compared with the following 1D line (and I'm cheating a bit by making it pseudo-2D):

- Image processing, filtering, feature extraction, and recognition are some of the biggest topics in modern computer science and AI. Google, for example, is putting a huge amount of resources into developing image recognition algorithms (because it's important that our future robot overlords can recognize cats and cucumbers when watching youtube videos?).

- This section focuses on more fundamental aspects of images and other 2D plane data, including edges, lines, 2D sine gratings, and basic geometric shapes.

- But "fundamental" doesn't mean unimportant — image processing and feature extraction methods must be checked against simple ground-truth metrics. Plus, your brain has dedicated brain cells that respond to fundamental image features such as edges, spatial frequencies, and geometric shapes!

- Along with the increased information possibilities for 2D planes, there is also increased diversity in how to simulate features in 2D space. There are many solutions to the same problem, and I encourage you to explore other methods or try to reproduce the images using different methods.

- The same signal-vs-noise discussion applies here: There are no absolute differences between "signal" and "noise"; that depends on the context and goals of the analysis. In this course, I consider "noise" to be data generated by manipulating random numbers, while "signal" is anything else.

- I simulated image lines by solving the equation $y = mx + b$, where m and b are parameters you can specify, corresponding to slope and intercept. There are a few additional programming tricks to get the line to look nicer (e.g., thicker, not throwing an error at the boundary of the image), but solving the equation is the main part.

- The image itself is made up of 0's and 1's where the line is drawn. This is generally how images are produced. Note that *creating an image with an embedded line* is different from *drawing a line on an axis*.

- Making an edge is the same as the line but filling in one side with 1's.

- 2D sine gradients (also called sine patches or gratings) are widely used in vision research, because brain cells in your visual cortex are "tuned" to respond to sine gradients in the visual input.

- The formula for sine patches is very similar as for sine waves: $\sin(2\pi fX + \phi)$ where f is the spatial frequency and X is a matrix of linearly increasing numbers (amplitude is implicitly set to 1). The main difference is that there are **two** phase values:
  1. The additive ϕ in the formula, which *shifts* the sine gradient.
  2. The multiplicative θ, which is used to *rotate* the values in the X matrix and therefore rotates the sine gradient, like the face of a clock.

- The Gabor patch is created by point-wise multiplying the sine patch by a 2D Gaussian. The result is a sine patch in the middle that fades with distance.

- For the purpose of this course, a "ring" can be defined as the intersection between two shapes such that one shape fits inside the other shape to create a 2D object with a hole completely enclosed by the outer shape. That just means putting one basic geometric shape on top of another.

Exercises

Note about the exercises: I give my answers in MATLAB code but you should feel free to use whatever program (Python, C++, Julia, html5) you feel most comfortable with.

1. Compute a histogram of a 2D sine patch (you'll need to vectorize the image to avoid making the histogram of each column of the image matrix). How does it compare qualitatively to the histogram of a 1D sine wave?

2. Same as above but for the 2D Gaussian.

3. The code that generates a circle based on radius specifies the radius in pixels, but perhaps fraction of the image size would be better. Adapt the code to implement this. What would be a solution if the image had a different number of pixels wide vs. high?

4. Adjust the triangle code to produce a triangle-shaped sine patch. The troughs (dips) of the triangle-sine patch are black and blend into the black background. How could you make the entire image stand out?

5. Imagine someone gave you a stack of images like the one below. The images contain circles and rings of different sizes, and you have the coordinates of the red outer tracings. Describe an algorithm that would distinguish circles from rings.

imagesignal_reader_qcirc.png

Answers

1. The histogram shows two peaks above -1 and below +1. It's overall very similar to the 1D sine wave. Code: `hist(img(:),100)`

2. Noticeably dissimilar to the 1D Gaussian: There are only positive values and the drop-off is steeper than the exponential shape of the 1D Gaussian.

3. Code:
   ```
   radius = .15;
   %later...
   img(dist<n*radius) = 1;
   ```
   There is no right answer, but my first choice would be to set the radius to be the average of width and height.

4. If you copy the code from earlier in the script to create the `sine2d` matrix, then:
   ```
   imagesc((1-img).*sine2d)
   ```
   The `1-img` part is because I wrote the code be 1=background and 0=1 image. If you leave the 1- part out, then the result also looks neat!

   To make the sine patch stick out more, you could, for example, change the color axis limits to [-1 1].

5. Based on the image shown below the question, one idea is to compute the ratio of pixel values inside the red ring to the diameter of the ring. That ratio would be either high (circle) or low).