# CIS581, Computer Vision, Fall 2016
# Project 1, Image Edge Detection
# Due on Sept. 27, 3:00pm

**Instructions.** This is an individual assignment. All matlab functions should follow the names and arguments stated in the problems in order for them to run properly with the grading script. A test script will be provided shortly that will call your functions to ensure they will run inside the grading script. To submit the assignment, upload a zip file containing all your code via Canvas.

## Overview

This project focuses on understanding image convolution and edge detection. The final goal of this project is to compute the Canny Edges for any RGB image.

A basic code structure is provided in `cannyEdge.m`:

`E = cannyEdge(I)`

> (INPUT) `I`: $H \times W \times 3$ matrix representing the RGB image, where H, W are the height and width of the image, respectively.
> (OUTPUT) `F`: $H \times W$ binary matrix representing the canny edge map, where a 1 is an edge pixel and a 0 is a non-edge pixel.

Your task is to implement three functions `findDerivatives.m`, `nonMaxSup.m` and `edgeLink.m` which correspond to the steps described in the lecture notes:

1. compute local edge normal orientation,
2. seek local maximum in the edge normal orientation,

3. continue search in the edge orientation of detected edge point.

## 1 Local Edge Orientation

**Goal** Compute magnitude and orientation of derivatives for the given image $I$.

`[J, theta, Jx, Jy] = findDerivatives(I, Gx, Gy)`

> (INPUT) `I`: $H \times W$ matrix representing grayscale image.
> (INPUT) `Gx, Gy`: Gaussian filters.
> (OUTPUT) `J`: $H \times W$ matrix represents the magnitude of derivatives.
> (OUTPUT) `theta`: $H \times W$ matrix representing the orientation of derivatives.
>
> (OUTPUT) `Jx`: $H \times W$ matrix representing the magnitude of derivatives along x-axis.
> (OUTPUT) `Jy`: $H \times W$ matrix representing the magnitude of derivatives along y-axis.

## 2 Detect Local Maximum

**Goal** Find local maximum edge pixel using non-maximum suppression along the line of the gradient.

`M = nonMaxSup(J, theta)`

> (INPUT) `J`: $H \times W$ matrix representing the magnitude of derivatives.
> (INPUT) `theta`: $H \times W$ matrix representing the orientation of derivatives.
> (OUTPUT) `M`: $H \times W$ binary matrix representing the edge map after non-maximum suppression.

## 3 Edge linking

**Goal** Use hysteresis to link edges based on high and low magnitude thresholds.

`E = edgeLink(M, J, theta)`

(INPUT) `M`: $H \times W$ binary matrix representing the edge map after non-maximum suppression.
(INPUT) `J`: $H \times W$ matrix representing the magnitude of derivatives.
(INPUT) `theta`: $H \times W$ matrix representing the orientation of derivatives.
(OUTPUT) `E`: $H \times W$ binary matrix representing the final edge map.

## 4 Testing and Submission

- Good test images can be found in the Berkeley Segmentation Dataset at http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/. Additionally, benchmark results are included for each image. We also provide a few images with the test script in the `train_images_P1` folder.
- MATLAB has its own Canny edge detector implementation using `edge(I,'Canny')`. Use your own images and compare your results with MATLAB's version.
- We have provided a test script for this project on the course wiki `https://alliance.seas.upenn.edu/~cis581/wiki/index.php?title=Projects`. Extract the contents of the `TestScript_P1` to the same directory as your functions and run `TestScript_P1.m` in MATLAB. This does not test the correctness of your code itself, only whether the output is in the expected format. Additionally, when grading, we'll be calling your functions in the same manner, so make sure they work as you'd expect on the sample in the test script.
- You should collect all your source code files and test images into a folder named as your student number. Then package this folder to zip file and submit to Canvas. Any break to this rule will lead a failure in the test script.

## FAQ

**Q**: How to set thresholds of hysteresis in Canny edge detection?

**A**: Vision is all about picking the 'right thresholds'. For this project, you should pick thresholds yourselves to get good performance. Since the thresholds aren't passed into the function, you either choose the thresholds that work well across a wide range of images or use some method to automatically determine what thresholds would be good based on the input image. In class, we mentioned two types of thresholds that are commonly used: a threshold based on the range of all magnitudes in the image, or a threshold based on a local block of the image that is currently being evaluated.

**Q**: Are we allowed to use functions like bwselect and bwmorph?

**A**: There's no restriction on what functions you can use (except `edge(I)` of course). However, the edge linking step will be more involved than just calling those functions. Rather than checking all adjacent pixels to see if they're above a threshold, the edge linking should check in the direction perpendicular to the image gradient. In general we're not expecting everyone to implement the edge linking in the exact same way, so if you have a method without recursion that performs well that's fine.