HOMEWORK 8

Due: Tuesday, May 29, 2018, 11:55 pm

1. **Spatial Hashing.** In class, we discussed how to speed up the identification of particle-particle interactions like collision by discretizing the entire domain into a series of rectangular bins. In this problem, you'll construct your own approach to mapping particle data onto a 2D grid. The linear-indexing and spatial conventions we'll use are illustrated in Figure 1.

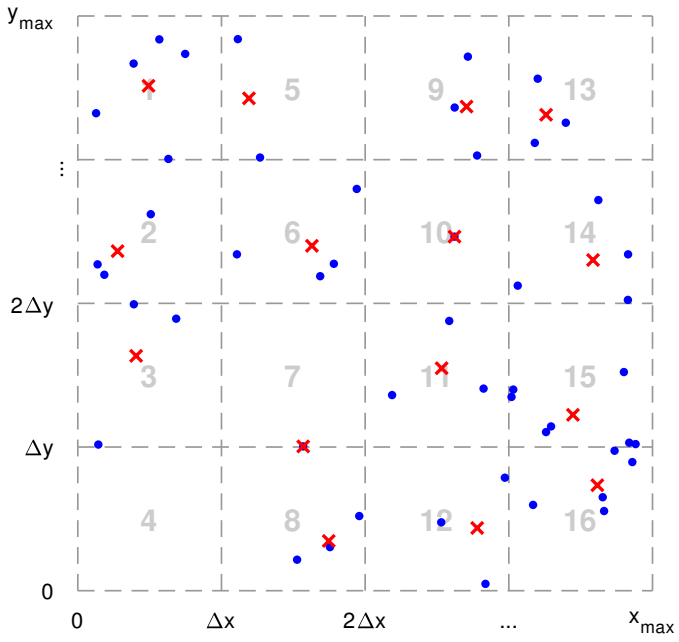

Figure 1: The location of each particle (blue dot) and bin-average location (red x) as calculated on a 4×4 grid, with $x_{max} = y_{max} = 1$ and $h = 0.25$. The following calculations assume bin numbering starts in the upper-left corner, proceeds first by column, and that the lower-left corner of the grid corresponds to (0, 0)

   (a) Begin by defining two `[1xN]` vectors with uniformly distributed random values to represent the x and y position of each particle. Set the size of your grid with variables x_{max} and y_{max}. You're free to scale and/or shift the particles to fall in any region of your domain, but you must choose your limit variables such that $0 < x_k < x_{max}$ and $0 < y_k < y_{max}$ for $k = 1, 2, \ldots, N$.

   (b) Next, begin partitioning the grid. Specify the minimum cell dimension, h, that will be used along with x_{max} and y_{max} to generate the integer number of bins in both the x- and y-directions, N_X and N_Y, respectively. Depending on your choice of h, the *actual* bin dimensions, Δx and Δy, may be larger than requested (remember, the value h controls the *minimum* width and height).

   (c) Construct a vector with the same size as x and y to hold the bin number of each particle. For every particle k with x-position x_k and y-position y_k, the linearly-indexed bin number can be calculated as

$$\text{binNum}_k = \left(\left\lceil \frac{x_k}{\Delta x} \right\rceil - 1 \right) N_Y + \left\lceil \frac{y_{max} - y_k}{\Delta y} \right\rceil \qquad \text{for } k = 1, 2, ..., N \tag{1}$$

where $\lceil ... \rceil$ indicates rounding the bracketed quantity up to the nearest integer. Using the bin numbers to classify points, calculate the average (x, y) position of the particles in each bin for bins $1, 2, ..., N_X \times N_Y$.

(d) Construct a single plot that displays both the location of each particle and the location of each bin average as unconnected markers. You can plot the bin dividing lines as a visual check of your algorithm by first calling `grid on` and then changing the interval spacing (e.g., `xticks(0:dx:xMax)`). Finally, display the particle ID's contained within each bin to the command window using *exactly* the formatting shown below (obtained using $N = 50$ and `rng('default')`):

```
Bin  1:  11 16 32 35 45
Bin  2:  3 34 40
Bin  3:  6 22 30
Bin  4:  []
  ⋮
Bin 16:  4 10 18 19 24
```

where we interpret the output as particle 11, described by `x(11)` and `y(11)`, falls into Bin 1, and so on. Test your code by generating results for both a relatively sparse data set, e.g., $N = 20$, and dense data set, e.g., $N = 200$. In cases where a bin holds no particles, no average point should appear on the plot and `'[]'` should be printed as part of the command window output.

(e) How does your code classify a point $x_k = 0, y_k = y_{max}$? What bin *should* this particle logically be assigned to, and (in words or pseudocode only) how could you modify your algorithm to catch cases like these? Suppose each of these particles has a maximum interaction radius of h (i.e., each particle only cares about other particles if their separation distance is $\leq h$). How many bins filled with particles do you need to check to identify *all* particles within distance h of a particle located in Bin 1?

Note: Although you're encouraged to adjust the partitioning values and data set to get a feel for how spatial hashing works, please submit your code with $N = 20$, $h = 0.25$, and values uniformly distributed on a $x_{max} = y_{max} = 1$ grid.

2. **Newton's Method.** A zero-finding method of great interest is *Newton's method*. It uses both the value of the function and its derivative at each step. Suppose f and its derivative are defined at x_c, where "c" denotes current. The tangent line to f at x_c has a zero at $x_+ = x_c - f(x_c)/f'(x_c)$ assuming that $f'(x_c) \neq 0$. Newton's method for finding a zero of the function f is based on the repeated use of this formula. Complete the following function so that it performs as specified.

```
function [xc, fEvals] = Newton(f, x0, delta, fEvalMax)
% f is a HANDLE to a continuous function, f(x), of a single variable.
% x0 is an initial guess to a root of f.
% delta is a positive real number.
% fEvalsMax is a positive integer >= 2 that indicates the maximum
% number of f-evaluations allowed.
%
% Newton's method is repeatedly applied until the current iterate, xc,
% has the property that |f(xc)| <= delta. If that is not the case
```

```
% after fEvalsMax function evaluations, then xc is the current iterate.
%
% This routine computes the derivative of f at each iterate xc by
% using a central difference approximation with small perturbation size.
%
% fEvals is the number of f-evaluations required to obtain xc.
```

Instead of evaluating the derivative of the function, f, analytically, you should use a *central difference approximation*

$$f'(x) = \frac{df}{dx} \simeq \frac{f(x+h) - f(x-h)}{2h} \tag{2}$$

where the perturbation size, h, is 10^{-6}. Verify that your routine works by finding and reporting the zeros of

$$f(x) = 816x^3 - 3835x^2 + 6000x - 3125 \tag{3}$$

Write a script to call your function with different values of x_0 in the range $1.43 \leq x_0 \leq 1.71$ in increments of 0.01. Generate command window output by printing results for each x_0 iteration using exactly the formatting shown below (values for illustration only).

```
x0 = 1.43, evals =  5, xc = 1.471234
x0 = 1.44, evals =  6, xc = 1.471234
  ⋮
x0 = 1.71, evals =  5, xc = 1.661234
```

You may find it helpful to plot the behavior of this function in the range $1.43 \leq x_0 \leq 1.71$ before getting started. Discuss the effect of `delta` on the zeros returned by your function and the number of evaluations required. Using what you know about how Newton's method finds roots, explain the behavior of the values printed in your table when x_0 is in the range $1.61 \leq x_0 \leq 1.62$.

Using the naming convention presented in the syllabus, submit **two** separate files to the CCLE course website: (1) a .pdf of your written report and (2) a .zip file containing all of the MATLAB files written for the assignment. Remember to use good coding practices by keeping your code organized, choosing suitable variable names, and commenting where applicable. Any of your MATLAB .m files should contain a few comment lines at the top to provide the name of the script, a brief description of the function of the script, and your name and UID.