Northeastern University
Align Seminar – Charlotte
Summer 2019, Nate Derbinsky

Optimal Preparation

1. **Math Warm-up**. Consider the following function (you may want to graph it in your favorite application/website):

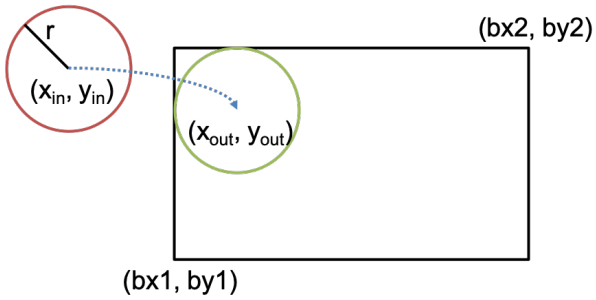
$$f(x) = (x-2)^2 + 7$$

   (a) What is the *minimum value* of this function? (Meaning, what is the smallest number that can be output over all real-valued inputs of x?)

   (b) What real-valued input (x) produces this minimum output? Or, in mathematical terms, solve for x below given the above definition of f:

$$\arg\min_x f(x)$$

2. **Code Warm-up**. Consider the following diagram:



You are to write the code for a function to make sure a supplied circle, centered at (x_{in}, y_{in}) with radius r, is either already within a supplied box, with lower-left corner $(bx1, by1)$ and upper-right corner $(bx2, by2)$, or moves to the *closest* location such that the circle is completely in the box. Here is some pseudocode to get you started ...

```
def get_in_the_box(x_in, y_in, r, bx1, by1, bx2, by2):
    # if circle_is_already_in_the_box():
    #   return x_in, y_in
    # else:
    #   x_out, y_out = compute_closest_point_in_the_box()
    #   return x_out, y_out
```

3. **Math Challenge**. Consider the following function:

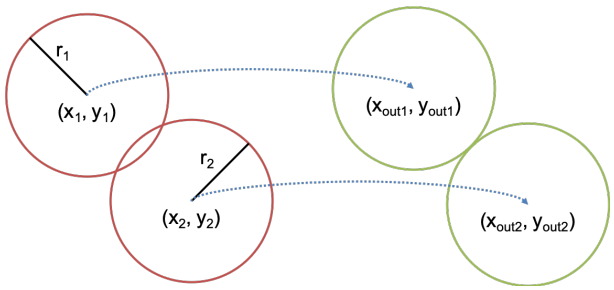
$$f(x) = \frac{k}{2}(x-a)^2 + \frac{\rho}{2}(x-n)^2$$

In terms of the constants (k, a, ρ, n), solve for x below given the above definition of f:

$$\arg\min_x f(x)$$

- Context: moving x away from a costs you $\frac{k}{2}$, but moving x away from n costs you $\frac{\rho}{2}$ – so where should you optimally choose x to incur the smallest penalty?
- Hint: if you know how, take the derivative with respect to x, set equal to 0, solve for x; otherwise, think about how to find the "middle" of the two locations (a and n) that are respectively weighted according to k and ρ (what would be the answer if the weights were both 1? what if they were both the same but not necessarily 1?).

4. **Math & Code Challenge**. Consider the following diagram:



You are to write the code for a function to make sure that two supplied circles (each with their respective center and radius) either are not colliding or each move the *shortest* distance such that they are not colliding. Here is some pseudocode to get you started ...

```
def keep_away(r1, cx1, cy1, r2, cx2, cy2):
    # if circles_not_colliding():
    #   return cx1, cy1, cx2, cy2
    # else:
    #   x_out1, y_out1, x_out2, y_out2 =
    #       compute_closest_non_colliding_locations()
    #   return x_out1, y_out1, x_out2, y_out2
```

Some hints ...

- Geometrically, the movement of each circle should be along the line that connects the centers of the two circles
- Move each circle an equal amount (though in opposite directions)