Final Assignment

This assignment is part of the final exam. For that reason, please do not collaborate on it. It is all about coding, but you should make a small writeup to explain the results and possibly details of your code. In each case write a code that handles a generic problem and apply the same numerical algorithm to some simple validation cases as well as the "production" runs. Use Autograd or Autodiff to compute the necessary derivatives except (at your discretion) derivatives of the constraint functions.

Exercise 2 may involve more work than you have time for. If so, just do Exercise 1 and think of Exercise 2 as extra credit that you did not have time for.

1. **Gradient descent.**

   Write a code to do gradient descent for unconstrained optimization. It should take a step size to satisfy the sufficient decrease and not-too-small step size conditions

   $$\Delta u \geq \alpha \Delta u_P \qquad \text{(sufficient decrease)}$$
   $$\Delta u \leq \beta \Delta u_P \qquad \text{(not-too-sall)}$$

   If the sufficient decrease condition is violated, decrease (possibly halve) the step size, If the not-too-small condition is violated, increase the step size (possibly double). Make sure when the step size is chosen, both conditions are satisfied. For that, it is necessary that $\beta > \alpha$. Common choice (which you can play with) is $\alpha = .1$ and $\beta = .9$, The predicted decrease (based on the first derivative approximation of u about iterate x_n) is $\nabla u(x_n)(x_n - x_{n+1})$.

   As a model problem, think about $u(x, y) = x^2 + k^2 y^2$. For large k the level curves are very short and fat ellipses, It is easy for gradient descent step sizes to be too large, which makes it a good test problem. Make a 2D plot (a scatterplot) with the iterates x_n and possibly the first guess for x_{n+1}, which might be $x_n - s_{n-1}\nabla u(x_n)$, where s_{n-1} is the step size that was ultimately used for the previous iteration. One way to make this clear is to have a think line segment from the point representing x_n to $x_n - s_{n-1}\nabla u(x_n)$. You don't have to put a point on the other end of this segment. Take as initial guess $(x_0, y_0) = (1, 1)$ and $s_0 = 1$.

   For the production run, take $u = V(R)$ from Assignment 9. Be careful with the notation. What is called x here is a point with $2M$ components. In Assignment 9, this same point is called $R = (x_1, y_1, x_2, y_2, \cdots, x_M, y_M)$,

where point $r_k k$ in $\mathbb{R}^2$ has coordinates $r_k = (x_k, y_k)$. Change the background confining potential to

$$V_0(r) = x^2 + 2y^2 \ .$$

This removes some of the symmetry in the Assignment 9 potential.

Try with a small M (maybe $M = 2$ or $M = 3$) to get the algorithm to work Start (this is important) with a random initial guess with the r_k independent two component Gaussians with mean zero and identity covariance. If you run the algorithm several times on the same problem, you should get the same optimizer, except for the remaining symmetry.

Finally, try your algorithm with larger M, (starting, maybe, at $M = 20$ and experimenting with larger values). Make a 2D scatterplot of the optimized configuration R_*. Note that these have "crystalline" regions where particles arrange themselves in a regular array, but there may be several such regions with different alignments and there may be "outliers" that are not part of a piece of crystal. If you start from different randomized initial configurations, you are likely to get different optimized configurations. See whether this happens for you. Choose stopping criteria so that making them more stringent doesn't change the scatterplots.

2. **Constrained gradient descent.**

Write code for equality constrained gradient descent. The code should take as arguments the objective function u and a list of the constraint functions g_k. Use the (your adaptation of) the algorithm from the notes, which uses the projected gradient and then projection back to the constraint surface. Solve the linear projection equations using a linear equation solver that is part of `numpy`. There are several choices because the matrix you need to invert (or the matrix that is involved in the linear system you have to solve) is symmetric and positive definite.

For a test problem, minimize $u(x, y) = x + 3y$ with the constraint $x^2 + 2y^2 = 1$. You know the analytical solution. Be careful that your algorithm works when the number of constraints is $m = 1$ and the dimension of the tangent space is $d - m = 1$.

If you want (I would) try a larger problem with a linear objective and quadratic constraints. You can easily tell whether what you converge to is a min on the constraint surface (the gradient of the objective and constraints are simple).

For a production run, put $M = 2H$ particles in the potential V described above Add the H constraints that particle $2k$ and particle $2k + 1$ are "bound" together at a distance of $d = .1$. The constraints are

$$g_k(R) = \|r_{2k} - r_{2k+1}\|^2 - .01 = 0 \ .$$

Visualize the optimized configurations using line segments with r_{2k} on one end and r_{2k+1} on the other. Is there any crystal structure observable?