ASSIGNMENT 3

AER1415: Computational Optimization

Atilla Saadat Dehghan
atilla.saadat@mail.utoronto.ca
April 28th 2021

Question 1a)

Consider the unconstrained minimization problem for the convex quadratic:

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}x^TAx - x^Tb,$$

where $A \in \mathbb{R}^{n\times n}$ is an SPD matrix and $b \in \mathbb{R}^n$.

$\because$ A is non-singular and SPD, the optimal solution is

$$\therefore x^\star = -A^{-1}b$$

Subbing in the optimal solution to get the optimal objective function value:

$$f(x^\star) = -\frac{1}{2}b^TA^{-1}b$$

Let $p_k \in \mathbb{R}^n$ be a non-zero search direction at the iterate $x_k \in \mathbb{R}^n$, which is also the negative gradient vector. Also, let $\nabla f_k = Ax_k - b$

$$p_k = -\nabla f_k$$

Determining the next iteration:

$$\begin{aligned} f(x+\alpha p_k) &= \frac{1}{2}(x+\alpha p_k)^TA(x+\alpha p_k) + b^T(x+\alpha p_k) \\ &= \frac{1}{2}x^TAx + \alpha p_k^TAx + \frac{1}{2}\alpha^2 p_k^TAp_k + b^Tc + \alpha b^Tp_k \\ &= f(x_k) - \alpha p_k^Tp_k + \frac{1}{2}\alpha^2 p_k^TAp_k \end{aligned}$$

Differentiating with respect to α, and setting the derivative to zero, we obtain the optimized value for α:

$$\alpha = \alpha_k = \frac{p_k^Tp_k}{p_k^TAp_k}$$

Using the exact minimizer a_k, the next iterate becomes:

$$x_{k+1} = x_k - \alpha_k p_k = x_k - \left(\frac{p_k^Tp_k}{p_k^TAp_k}\right)p_k$$

$\because p_k = \nabla f_k = Ax_k - b$, this equation provides a close-form expression for x_{k+1} in terms of x_k

Question 1b)

Considering the line search minimization problem:

$$\alpha_k = \arg\min_{\alpha \in \mathbb{R}} f(x_k + \alpha p_k)$$

where x_k is the current iterate, and p_k is the current search direction.

Normally, p_k is required to be a descent direction of f at x_k,

$$\therefore f(x_k + \alpha_k) < f(x_k) \ \forall \alpha \in (0, \epsilon]$$

for some $\epsilon > 0$. When f is differentiable, any p_k, such that $\nabla f(x_k)^T p_k < 0$, is a search direction whenever $\nabla f(x_k) \neq 0$. Therefore, the condition to check that $p_k \in \mathbb{R}^n$ is a direction of descent is to check if $\nabla f(x_k) \neq 0$.

Question Q2)

Consider the singly constrained optimization problem,

$$\min_{x \in \mathbb{R}^n} f(x) = x_1^3 + x_2^3$$

$$s.t.\ c_1(x) = x_1 + x_2 = 1$$

The KKT conditions for this problem to be satisfied with a given solution are:

$$\nabla_x \mathcal{L}(x^\star, \lambda^\star) = \nabla f(x^\star) - \lambda^\star \nabla c_1(x^\star) = 0$$

$$\nabla_\lambda \mathcal{L}(x^\star, \lambda^\star) = h_1(x^\star) = 0$$

Solving for the Lagrangian function and setting the partial derivatives to zero (to minimize each variable) yields:

$$\mathcal{L}(x, \lambda) = 3x_1^2 + 3x_2^2 - \lambda(x_1 + x_2 - 1)$$

$$\nabla_x \mathcal{L} = \begin{bmatrix} 3x_1^2 + \lambda \\ 3x_2^2 + \lambda \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1^\star \\ x_2^\star \end{bmatrix} = \begin{bmatrix} \pm\sqrt{\frac{1}{3}\lambda^\star} \\ \pm\sqrt{\frac{1}{3}\lambda^\star} \end{bmatrix}$$

$$\nabla_\lambda \mathcal{L} = -x_1 - x_2 + 1 = 0$$

$$\pm 2\sqrt{\frac{1}{3}\lambda^\star} = 1$$

$\because \lambda^{\star} = -0.75$ does not satisfy the constraint to maintain a real number.

$$\therefore \lambda^{\star} = +0.75$$

$$\therefore \begin{bmatrix} x_1^{\star} \\ x_2^{\star} \\ \lambda^{\star} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.75 \end{bmatrix}$$

$$f(x^{\star}) = 0.25$$

The solution for the state variables and the Lagrangian multipliers also all the satisfies the KKT conditions and set constraints.

Figure 1 displays the 2D contour, generated in *"q2.py"*, of the given optimization problem, showing the first order KKT conditions and that the point is indeed a local minimum that also satisfies the given equality constraint.

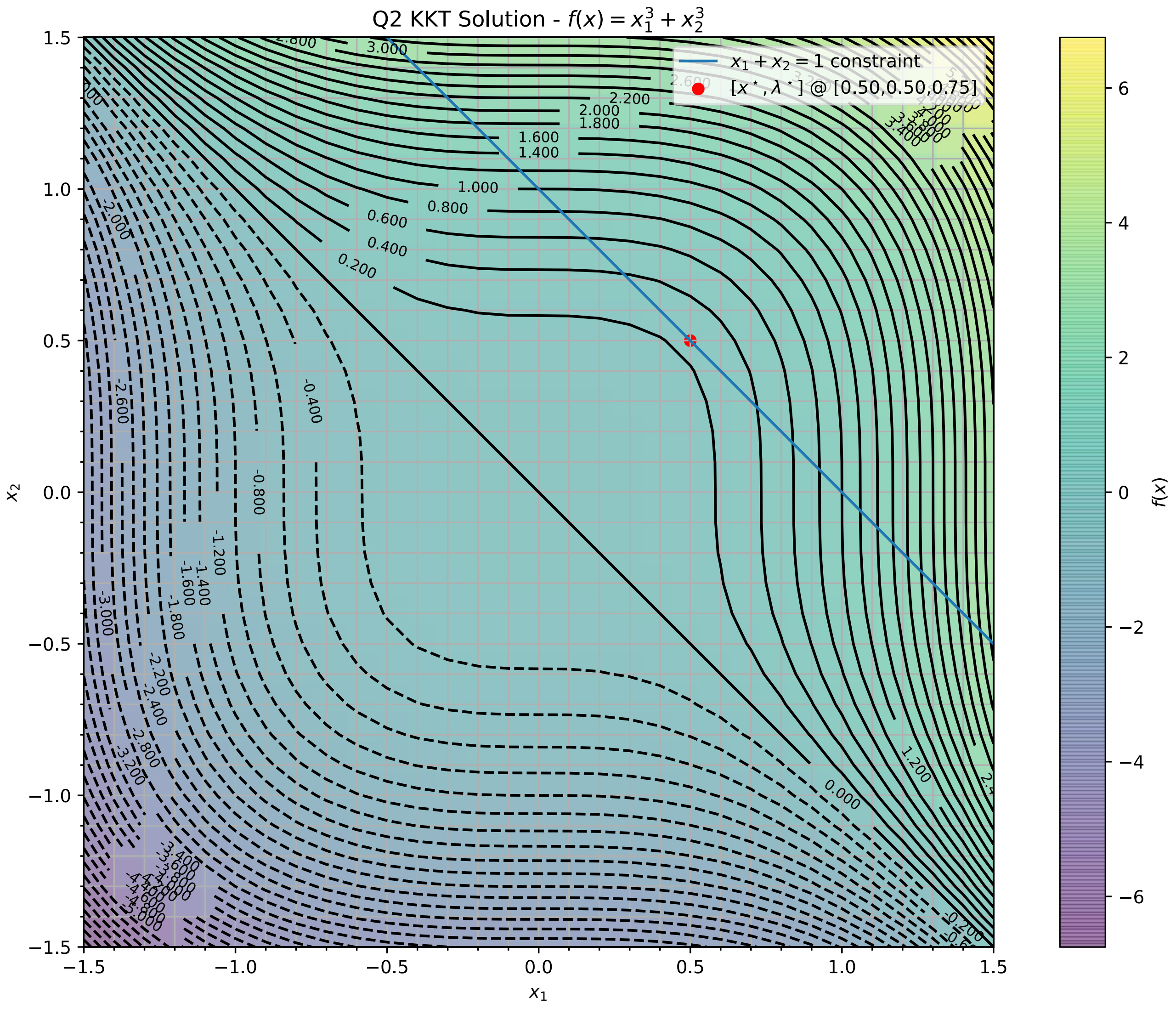

Figure 1: 2D Contour of Q2 Optimization problem with local minimum and equality contraint

Question 3a-b)

Consider the given quadratic programming optimization problem with two equality constraints:

$$\min_{x\in\mathbb{R}^n} 3x_1^2 + 2.5x_2^2 + 3x_3^2 + 2x_1x_2 + x_1x_3 + 2x_2x_3 - x_1 - x_2 - x_3$$

$$\begin{aligned} s.t.\ \ x_1 + x_3 &= 3 \\ x_2 + x_3 &= 0 \end{aligned}$$

Rewriting the problem into the standard convex quadratic form with a set of linear equality constraints yields the following matrices

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}x^T Gx + x^T d$$

$$s.t.\ \ Ax = b$$

where $G \in \mathbb{R}^{n\times n}$ is SPD, $d \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$.

The SPD matrix, G, and its associated $x^T Gx$ matrix multiplication is as follows to fit the form above (calculated by taking the first order gradients and rearranging)

$$G = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}$$

$$d = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

With these defined matrices, the Lagrangian function can be written as

$$\mathcal{L}(x,\lambda) = \frac{1}{2}x^T Gx + x^T d - \lambda^T(Ax - b)$$

The Lagrangian stationary points (KKT conditions), $(x^\star,\lambda^\star)$, can thus be written as

$$\begin{aligned} Gx^\star + d - A^T\lambda^\star &= 0 \\ Ax^\star - b &= 0 \end{aligned}$$

and rearranged into matrix form to easily solve as a system of linear scalar equations

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x^\star \\ \lambda^\star \end{bmatrix} = \begin{bmatrix} -d \\ b \end{bmatrix}$$

The first matrix in this equation is the KKT matrix and is non-singular, therefore there is a unique vector pair,$(x^\star,\lambda^\star)$, exists that satisfies the equation. This outlines the first-order necessary conditions for $x^\star$ to be an optimal solution and are the result of the of the first-order optimality conditions.

Numerically generating the solution and optimal Lagrange multiplier vector in "*q3.py*" yields:

$$x^\star = [1.46154, -1.53846, 1.53846]^T$$
$$\lambda^\star = [6.23077, -2.69231]^T$$

Question 3c)

In some cases, the Lagrange multiplier, $\lambda^\star$, has some interpretations. The value of $\lambda^\star$ (at the minimum $x^\star$) is how fast the quickly the minimum function value at $x^\star$, $f(x^\star)$, would change if the constraint would be relaxed by increasing the equality values, c_i, by a constant rate, where

$$g_1(x) = \ x_1 + x_3 = c_1 = 3$$
$$g_2(x) = \ x_2 + x_3 = c_1 = 0$$

Question 4)

Consider the following optimization problem

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x + b^T x + c$$

$$s.t. \ \ g_1(x) = -x^T x + \delta \geq 0$$

where $c, \delta \in \mathbb{R}, b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$ and SPD.

Let $\mathcal{A}(x^\star)$ be the active set consisting of the indices of the constraints when the equality holds at $x^\star$

$$\mathcal{A}(x^\star) = \{-(x^\star)^T x^\star + \delta = 0\}$$

From the list of KKT conditions in [1], any solution for $x^\star$ satisfies the following first-order conditions, for the Largange multipliers, $i \in \mathcal{A}(x^\star)$. The Lagrangian becomes:

$$\mathcal{L}(x, \mu, s) = \frac{1}{2} x^T A x + b^T x + c - \mu(-x^T x + \delta - s^2)$$

where s is the slack variable.

The first-order KKT conditions are set by differentiating the Lagrangaian with respect to $x, \mu,$ and s, all set to zero:

$$\nabla_x \mathcal{L} = A x^\star + b - \mu^\star x^\star = 0$$

$$\nabla_\mu \mathcal{L} = -{x^\star}^T x^\star + \delta - (s^\star)^2$$

$$\nabla_s \mathcal{L} = -2\mu^\star s^\star = 0$$

$$\mu^\star \geq 0$$

Question 5a)

The PSO algorithm, in *"q5.py"*, is divided into two object classes, one intended to invoke a the general PSO algorithm, and the Particle class which defined the N number of particles created for the swarm. The PSO algorithm utilizes an iterative approach to minimizing an objective function, using the notion (and varying) a particles Position, x, and Velocity, v. Specifically, x_i^k and v_i^k define a certain particle's positions and velocity of the ith particle at the kth time step, respectively. The position of the ith particle is updated at iteration $k+1$ as follows:

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \boldsymbol{v}_i^{k+1}\Delta t$$

$$\boldsymbol{v}_i^{k+1} = \omega \boldsymbol{v}_i^k + c_1 \frac{\boldsymbol{p}_i^k - \boldsymbol{x}_i^k}{\Delta t} + c_2 \frac{\boldsymbol{p}_g^k - \boldsymbol{x}_i^k}{\Delta t}$$

Where:

$p_i^k -$ best position of particle i

$p_g^k -$ swarm best particle position

$c_1 -$ combined cognitive parameter

$c_2 -$ combined social parameter

$\omega -$ inertial weight

The variables $\omega, c_1,$ and c_2 are from the results of a Sequential Least-Squares Programming (SLSQP) minimization algorithm within the Scipy optimize module, with 10^{-4} tolerance and 100 max iteration limit. The initial guesses for these are equal to the best constant values determined for P4 in Assignment 1 ([0.5,1.5,1.5]). The bounds are also set based on the expected ranges for these variables. The result from the optimization results in the minimum cost function value for the updated position with the velocity of these controlled parameters. This ensures the best value for these parameters are used in future iterations of position.

The various particle objects are defined inside the general PSO class. For each particle generated, the new position and cost function values are calculated. From the various particles, the global best is updated within the PSO class, based on the value of the new particle value, to be passed onto later particle calculations. This is iterated many times until the max iteration number is reached.

The PSO algorithm uses the Reflect bounding method to handle the bounding constraints, chosen as it was the best preforming method in Assignment 1 and as outlined in [8].

If $x_{t+1} > x_{max}$ then $x'_{i,t+1} = x_{max} - (x_{t+1} - x_{max})$

Else If $x_{t+1} < x_{min}$ then $x'_{i,t+1} = x_{min} + (x_{min} - x_{t+1})$

This allows the cost function to accept and consider constraints and ensures that they are maintained throughout the calculations. This results in the PSO algorithm solving for a new objective function:

$$\pi(x,\alpha) = f(x) + \rho_k \phi(x)$$

where ρ_k is a positive penalty function parameters and α is the penalty function constant coefficient. The quadratic penalty function is given by:

$$\phi(x) = \sum_{i=1}^{m} \max[0, g_i(x)]^2 + \sum_{i=1}^{q} h_i(x)^2$$

where $g_i(x)$ and $h_i(x)$ are the inequality and equality constraints, respectively, that the function is subject to, which have the forms:

$$g_i(x) \leq 0\,, i = 1,2, \dots m$$

$$h_i(x) = 0\,, i = 1,2, \dots q$$

The penalty function parameter, ρ_k, needs to grow to ∞ as the max iteration number is reached within the PSO algorithm. As stated in [2, p. 498], by driving ρ_k to ∞, we penalize the constraint violations with increasing severity. Therefore, it was chosen that the penalty function will have the form

$$\rho_k = \alpha^k$$

so that ρ_k exhibits an exponentially increate behaviour towards infinity as a function of iteration number. This also entails that α is another parameter that required rough tuning.

Therefore, the PSO algorithm's attempts to minimize:

$$\pi(x,\alpha), x \in \mathbb{R}^n$$

The PSO algorithm is also executed with each parameter for an M number of independent runs, where the mean and standard deviation of the all the results is returned. The best solution (the positions and the cost function value) are thus the mean of the M runs, giving a better indication of the performance of the PSO algorithm with the given parameters. For all subsequent analysis, M will be set to 10. Similarly, the PSO algorithm will run for 100 iterations and define the number of particle numbers with

$$Particle_{num} = \max[200, 10i]$$

where i is the number of positions or estimated state variables.

The pseudocode for this new Hybrid algorithm can be found in Appendix 1

Question 5b)

There exists some advantages and disadvantages when comparing hybrid algorithms that integrate stochastic algorithms with deterministic gradient-based optimization methods. The different optimization methods combine the advantages of various techniques to minimize their deficiencies [3]. Hybrid methods tend to use stochastic methods to find the general location where the global extreme is and then switch to a gradient based method to find the exact global minimum at a faster rate [4]. The disadvantage of hybrid algorithms is that they are often more complicated to implement, and further threshold coefficients need to be defined, such as when to which between stochastic and deterministic search. Integrating stochastic methods would also intrincally increase computational time to convergence as the random nature of stochastic methods might not yield ideal results every iteration, unlike deterministic approaches. Also, it is possible that there might not be a need for the hybrid algorithm, as usually hybrid algorithms are better at large-residual problems. Since often the problem is not known to have small or large residuals at the solution [2], the use of a hybrid algorithm might not be implicitly necessary.

Question 5c)

The Hybrid algorithm described in Section 5a and Appendix A was developed in and the results are shown in Figure 2. From the results, the developed PSO hybrid algorithm performed better than the initial PSO method, as it was able to reach a lower mean (10 runs) objective function value with fewer main loop PSO iterations.

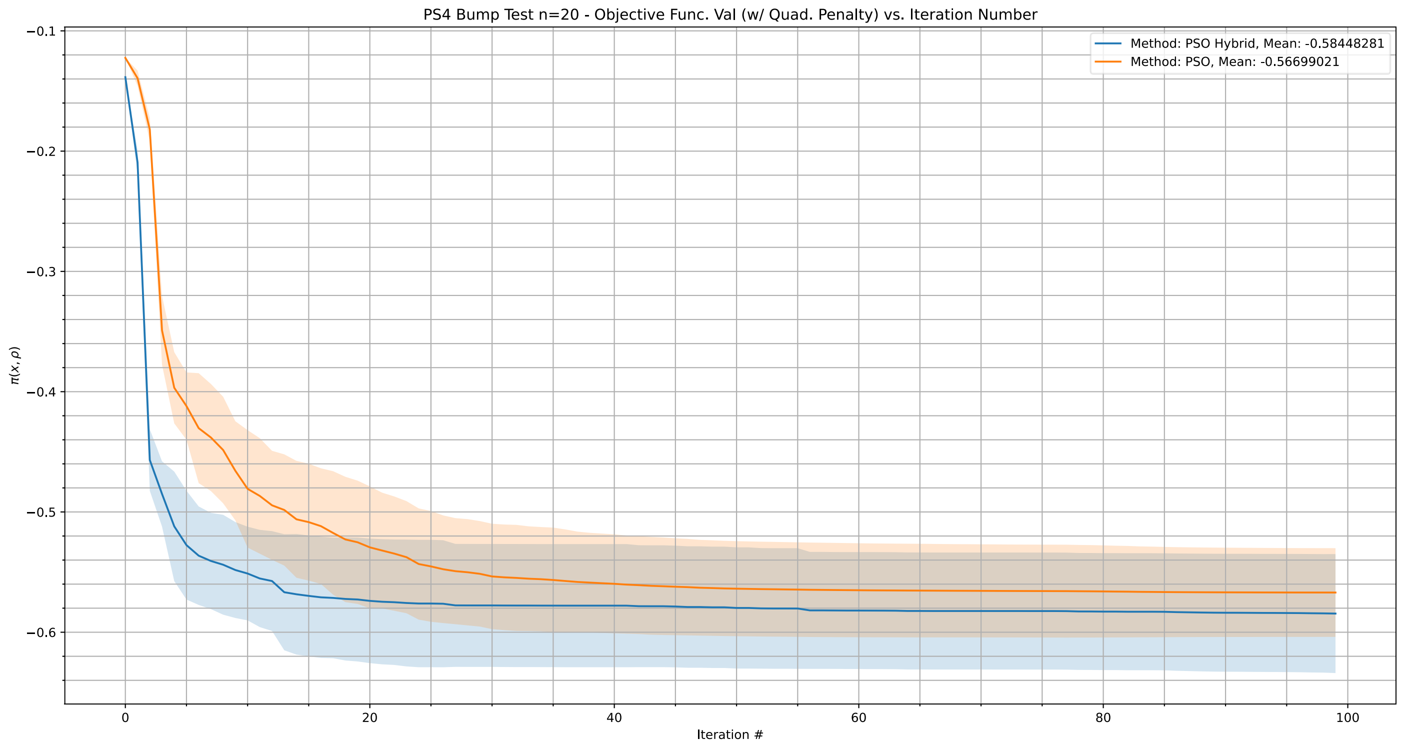

Figure 2: Hybrid PSO - SLSQP Algorithm - P4 Bump Test n=20

Question 6a) Considering the unconstrained minimization problem:

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{N} f_i(x)$$

where $f_i: \mathbb{R}^n \to \mathbb{R}, n \sim \mathcal{O}(10^6), N \sim \mathcal{O}(10^9)$, and the cost of one evaluation of f_i and ∇f_i is $\mathcal{O}(10^{-6})$ seconds. To formulate an efficient gradient-based algorithm, stochastic gradient descent (SGD) will be utilized based on the large size of n and N. "SGD tries to lower the computation per iteration at the cost of an increased number of iterations necessary for convergence" [5]. Since SGD using a uniformly random observation, it eliminates the need to calculate the sum of all the gradients, a step required for normal gradient descent.

The steepest descent (gradient descent) algorithm, for which SGD derives from, has the iterative update rule of

$$x_{k+1} = x_k - \alpha_k \nabla f_t(x_k)$$

where α_k the step-length is determined via line search, practically set by heuristic logic [6]. The SGD algorithm would order training points uniformly at random, t, and at least one pass is performed over the ordered training points, eg:

```
• Uniformly random training points $t = \{1,2,\dots,N\}$
• $x = x_0$
• $while\, !\,(|f(x_{k+1}) - f(x_k)| < \epsilon_a + \epsilon_r |f(x_k)|)$:
    o $for\ (i = 1 \dots N)$:
        ▪ $x_{k+1} = x_k - \alpha_k \nabla f_t(x_k)$
```

This algorithm utilizes the following computations inside the inner iteration loop, all with $\mathcal{O}(10^{-6})$: scalar multiplication, vector addition during each iteration, and ∇f_i evaluation, and two f_i evaluations for the convergence criterion. Therefore, the computation cost requirement per iteration per iteration is meet,

$$x_k - \alpha_k \nabla f_k \to\ 5 * \mathcal{O}(10^{-6})s < \mathcal{O}(10^{-4})s$$

and the overall cost of SGD is $\mathcal{O}(n)$ [6]. Theoretical convergence is also guaranteed as Robbins-Munro's theorem determines that SGD will converge under some assumptions, namely $\sum_k \alpha_k^2 < \infty$ and $\sum_k \alpha_k = \infty$, i.e. decaying learning rates such as $\alpha_{k+1} = \alpha_k / k\ ; \alpha_k > 0$ [6].

The convergence rate of SGD only sublinear and with convex objective functions, the number of iterations to achieve a given accuracy tolerance (ρ) is $\sim \mathcal{O}\left(\frac{1}{\rho}\right)$. Similarly, based on existing literature [7], the convergence rate of SGD for objective functions that have strong convexity satisfies

$$\mathbb{E}[f(x^k)] - f^* = \mathcal{O}\left(\frac{1}{k}\right)$$

and thus, SGD does not have linear convergence rates under strong convexity, unlike gradient descent.

Question 7)

Stochastic gradient descent's (SGD) major computational use derives from the product and vector sum operations, performed using multiple small n floating-point operations, whilst matrix-vector product is problem dependent [2]. Therefore, SGD is recommended for large problems / large datasets, otherwise methods such as QR factorization or singular value decomposition could be used. These methods tend to be less sensitive to rounding errors. SGD is advantageous in larger problems because they do not change the coefficient matrix (unlike factorization methods) and does not "produce fill in the arrays holding the matrix" [2]. SGD also sometime approaches the solution faster due it the stochastic nature of the method. Another advantage of SGD is that since the stochastic instances are independent, their process can be parallelized. A disadvantage is that SGD may be generally slower to converge to a global/local minimum and that it is hard to handle entries that are unobserved (requiring techniques like negative sampling).

Question 8)

From Section 6, the two approaches for deriving sequential quadratic programming (SQP) yields the following algebraic equations to be solved:

a) nonlinear equations (Newton-KKT system) solved using Newton's method:

$$\begin{bmatrix} W(x_k, \lambda_k) & -A(x_k)^T \\ A(x_k) & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) + A_k^T \lambda_\mathrm{k} \\ -h_k \end{bmatrix}$$

where $W(x_k, \lambda_k) = \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$ is the Hessian of the Lagrangian and the Newton step from the iterate (x_k, λ_k) is given by

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix}$$

b) formulating the SQP method to define the problem at (x_k, λ_k)

$$\min \frac{1}{2} p^T W_k p + p^T \nabla f(x_k)$$

$$s.t.\ A_k p + h_k = 0$$

where W_k approximates the Hessian of the Lagrangian; this system has the unique solution, (p_k, l_k), that satisfies

$$\begin{aligned} W_k p + \nabla f(x_k) - A_k^T l_k &= 0 \\ A_k p + h_k &= 0 \end{aligned}$$

Subtracting $A_k^T\lambda_k$ from both sides of the first equation in Approach A's Newton-KKT system yields

$$\begin{bmatrix} W(x_k,\lambda_k) & -A(x_k)^T \\ A(x_k) & 0 \end{bmatrix}\begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -h_k \end{bmatrix}$$

Because of the non-singularity of the coefficient matrix [2], $\lambda_{k+1} = l_k$ and therefore p_k solves both systems defined in Approaches A and B, meaning that both approaches are equivalent derivations of SQP algorithms.

Question 9)

Consider the following optimization problem

$$\min_{x\in\mathbb{R}^n} \int_{[0,1]^d} f(x,\theta)d\theta,$$

where $\theta \in [0,1]^d$ and $f: \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$.

Several challenges are expected when attempting to optimize the give black-box integral function. One such problem is that as the size of d increases, the computational cost of solving the integral because increasingly taxing. Reducing the size of d and the trapezoidal step size would yield better computational performances, however the overall accuracy of the results would reduce relative to the original objective function. This leads to a fundamental trade-off to be compromised between original objective function accuracy and computational time.

Between the discussed methods in AER1415, Nonlinear least-squares optimization is a good candidate. This can be achieved using Riemann Sums to approximate the integral as a summation, eg:

$$\int_{[0,1]^d} f(x,\theta)d\theta \cong \sum_{i=1}^{N} f(x,\theta)\Delta\theta$$

Setting up the objective function in this approximated form sets up the problem to be similar to a least squares regression model of the form:

$$\hat{f}(x,\theta) = \sum_{j=1}^{M} \omega_j \phi_j(x, \beta_j)$$

where ϕ_j is the basis function containing an undetermined parameter β_j, $\theta = \{w,\beta\} \in \mathbb{R}^n$ which has all the undetermined parameters and $n = 2M$ vector length. With this similar objective function, the ℓ_2 loss function to estimate θ [2].

Appendix 1: PSO Hybrid Algorithm Pseudocode

```
def objectiveFunction:
    calculate objective function with quadratic penalty function


class Particle:
    Initialize the particle's position with a uniformly distributed random vector
    Initialize the particle's best-known position to its initial position
    def updatePosition():
        Initialize SLSQP initial guesses for minimization -> x0, which contains:
            c1 – cognitive parameter (confidence in itself)
            c2 – social parameter (confidence in the swarm)
            w – inertial weight (inertia of a particle)
        Initialize SLSQP initial bounds for minimization
        Calculate optimal w,c1,c2 values using Sequential Least Squares Programming
        calculate new particle velocity based on optimal w,c1,c2:
            particle.vel = w*particle.vel + c1*(particle.bestPosition - particle.currentPosition)
+ c2*(swarmBestPosition-particle.currentPosition)
        update particle position with new particle velocity
        update particle position if position exceeds bound using Reflect bound approach


Initialize S number of particles in the swarm
for number of set max. iterations:
    for each particle in swarm:
        calculate objective function with current particle position
        if particle objective function val < particle best:
            update particle best val and position
        if particle objective function val < global best:
            update global best val and position
    for each particle:
        update particle position -> Particle.updatePosition()
```

References

[1] J. Nocedal and S. J. Wright, *Numerical optimization*. New York, NY: Springer, 2006.

[2] P. B. Nair, AER1415 Computational Optimization: List of Case Studies, *University of Toronto*, 2021.

[3] Z. Tang, X. Hu, and J. Périaux, “Multi-level Hybridized Optimization Methods Coupling Local Search Deterministic and Global Search Evolutionary Algorithms,” *Archives of Computational Methods in Engineering*, vol. 27, no. 3, pp. 939–975, 2019.

[4] M. Colaço and G. Dulikravich, “A Survey of Basic Deterministic, Heuristic, and Hybrid Methods for Single-Objective Optimization and Response Surface Generation,” *Thermal Measurements and Inverse Techniques*, pp. 369–420, 2011.

[5] R. Zadeh, “CME 323: Distributed Algorithms and Optimization,” 2015. [Online]. Available: https://stanford.edu/~rezab/classes/cme323/S15/notes/lec11.pdf.

[6] P. B. Nair, AER1415 Computational Optimization: Optimization algorithms for learning from data, *University of Toronto*, 2021.

[7] R. Tibshirani, “Convex Optimization Lecture 9,” 2019. [Online]. Available: https://www.stat.cmu.edu/~ryantibs/convexopt/scribes/stochastic-gd-scribed.pdf.

[8] S. Helwig, J. Branke, and S. Mostaghim, “Experimental Analysis of Bound Handling Techniques in Particle Swarm Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 2, pp. 259–271, 2013.