ECE 506 Optimization Theory

Name: Solutions

Problem 1	15	/15
Problem 2	25	/25
Problem 3	20	/20
Problem 4	15	/15
Problem 5	25	/25
Total:	100	/100

Good Luck!

Problem 1 (15 points)
Consider the following function

$$f(x) = \frac{x^4}{4} - 2x^3 + \frac{11}{2}x^2 - 6x.$$

1(a)(**5 points**) Verify that $x = 1, 2, 3$ are stationary points for this function.
1(b)(**5 points**) Determine the local nature (minimum, maximum, saddle point) for each stationary point.
1(c)(**5 points**) What is the global minimum value of $f(.)$?

Answer for 1(a)
We have:

$$\begin{aligned}
\frac{df}{dx}(x) &= x^3 - 6x^2 + 11x - 6 \\
\frac{df}{dx}(1) &= 1 - 6 + 11 - 6 = 0 \\
\frac{df}{dx}(2) &= 8 - 24 + 22 - 6 = 0 \\
\frac{df}{dx}(3) &= 27 - 54 + 33 - 6 = 0
\end{aligned}$$

Thus, $x = 1, 2, 3$ define stationary points.

Answer for 1(b)
We have:

$$\begin{aligned}
\frac{d^2f}{dx^2}(x) &= 3x^2 - 12x + 11 \\
\frac{d^2f}{dx^2}(1) &= 3 - 12 + 11 > 0 && \text{a minimum} \\
\frac{d^2f}{dx^2}(2) &= 12 - 24 + 11 < 0 && \text{a maximum} \\
\frac{d^2f}{dx^2}(3) &= 27 - 36 + 11 > 0 && \text{a minimum.}
\end{aligned}$$

In one-dimension, we cannot have a saddle-point!

Answer for 1(c)
Evaluate the function at the two minimum points to get:

$$\begin{aligned}
f(1) &= \frac{1}{4} - 2 + \frac{11}{2} - 6 = -2.25 \\
f(3) &= -2.25
\end{aligned}$$

Thus, we have the minimum value of $f(1) = f(2) = -2.25$ attained at $x = 1, 3$.

Problem 2 (25 points)
In this problem, we want to discuss ways to measure the performance of optimization algorithms. For real problems, there is no ground truth. In such cases, we can apply some tests to gain more confidence in the results. The discussion given here will apply to any optimization problem. The problem walks you through possible approaches.

2(a)(**5 points**) The first approach for establishing the performance of optimization algorithms is through the use of *simulation*. Explain (briefly) how you would use simulation to assess how an algorithm performs in terms of: (i) accuracy, (ii) robustness, and (iii) efficiency.

Answer:

- Accuracy: Plot $||x - x^*||$ versus the number of iterations and make sure that we have convergence.
- Robustness: Evaluate the accuracy from different, randomly-sampled starting points.
- Efficiency: Plot $||x - x^*||$ versus the number of function evaluations. Efficient methods will converge using a smaller number of function evaluations.

2(b)(**5 points**) Suppose that there is no ground truth. Instead, assume that you have programs (e.g., Matlab functions) that can evaluate: (i) the function value, (ii) the gradient of the function, (iii) the minimal point x, and (iv) the Hessian. How can you use these functions to establish whether the optimization algorithm converged to an actual minimum?

Answer: For a proper minimum, we need to check that the gradient is near zero and that the Hessian is positive definite.

2(c)(**4 points**) For large problems, the Hessian will not be available to you. How can you use the remaining functions to check whether the final result may actually be at a local minimum?

Answer: We can approximate the Hessian using finite-differencing on the gradient function. Please refer to your book for the details on how to approximate the Hessian using finite-differencing. Note that you will need the spacing to be above some minimum value. Alternatively, we can also fit a quadratic function over the function or its gradient (or both) to get an estimate of the Hessian.

2(d)(**4 points**) Suppose that you are only left with a function that returns the function value. How can you check that you may have reached a local minimum?

Answer: In this case, we can use finite differencing to estimate the gradient and the Hessian or we can use least-squares fitting to fit a local quadratic model. A simpler approach would be to take some random samples around x_0 and check to make sure that the function is minimized at x_0. Note that this simpler approach can also be applied to 2(c) as well.

2(e)(**4 points**) Suppose that you are suspecting that your result may be at a local minimum. Describe how to use sampling to partially alleviate this problem. In other words, describe how to use sampling to find the right minimum point.

Answer: The solution will be to use random starting points and then take the minimum from all of them. If another point produces a better minimum, then the algorithm had converged to a local, not a global minimum.

2(f)(**3 points**) Suppose that you are given two optimization algorithms. Slow-Algo is a slow algorithm that works very well. You believe that it is likely to give you the correct answer most of the time. Fast-Algo is a fast algorithm that you believe can do just as well or better than the slow algorithm. You are asked to compare these algorithms on a real-world problem that does not have ground truth. Suppose that you run both algorithms and they return (i) the function values, (ii) gradients, (iii) the minimal values of x, and (iv) the number of function evaluations. How can you use the returned values to establish that Fast-Algo is about as accurate as Slow-Algo but faster?

Answer: The basic idea is to plot the function that is being optimized against the number of function evaluations. If the Fast-Algo graph remains below Slow-Algo, then it is faster and works better. Notice that this simple approach does not require any ground truth or evaluation of gradients or Hessians. Clearly though, we will need to use the gradient and the Hessian values to confirm that a minimum has been reached when the algorithms terminate.

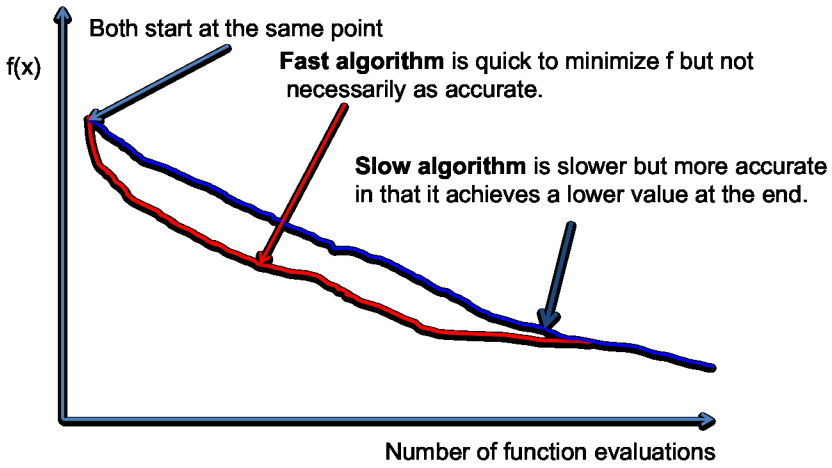


Figure 1: **Demonstration of convergence for slow and fast algorithms.**

Problem 3 (20 points) CG-Steihaug for large-scale optimization.
The advantage of the CG-Steihaug algorithm is that it allows us to handle large-scale problems without the need to compute the Hessians at every step.

Given tolerance $\epsilon_k > 0$;
Set $z_0 = 0$, $r_0 = \nabla f_k$, $d_0 = -r_0 = -\nabla f_k$;
if $||r_0|| < \epsilon_k$ **then**
  return $p_k = z_0 = 0$;
end
for $j = 0, 1, 2, \ldots,$ *MaxIterations* **do**
  if $d_j^T B_k d_j \leq 0$ **then**
    Find τ such that $p_k = z_j + \tau d_j$ solves
      $\min_{p_k} m(p_k) = f + g^T p_k + \frac{1}{2} p_k^T B_k p_k \quad \text{s.t.} \quad ||p_k|| \leq \Delta_k$ **return** p_k;
  end
  Set $\alpha_j = r^T r / (d_j^T B_k d_j)$;
  Set $z_{j+1} = z_j + \alpha_j d_j$;
  if $||z_{j+1}|| \geq \Delta_k$ **then**
    Find $\tau \geq 0$ such that $p_k = z_j + \tau d_j$ satisfies $||p_k|| = \Delta_k$;
    return p_k;
  end
  Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
  Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$
end

Algorithm 1: General framework for CG-Steihaug for trust-region optimization.

3(a)(5 points) Study the code given above and explain how to modify the code so that it will become *Hessian-free*. For your solution, you can assume that you are given a Matlab function for computing ∇f. Furthermore, we use the term *Hessian-free* to refer to the fact that we will avoid computing the full B_k.

Answer: The basic idea is to avoid any evaluations of $B_k d_k$. This can be done using the approximation:

$$B_k d_k \approx \frac{\nabla f_k(x_k + h d_k) - \nabla f_k(x_k)}{h}.$$

We need to use this approximation in 0.7 and 0.11. Note that we will need to do this repeatedly for 0.9.

3(b)(6 points) How many gradient-function evaluations do you need for each time you need to compute Hessian-vector products? How many function evaluations would you need for computing the full $n \times n$ Hessian (with exact derivatives)? What is the speedup factor? **Hint:** For the speed-up factor, you will need to divide the number of function evaluations for the full Hessian by the total number of gradient-function evaluations.

Answer: Here, we are only interested in the number of function evaluations. There is only one extra gradient function evaluation every time we use the approximation. To compute the speedup factor, note that the gradient requires n component function evaluations while the full Hessian requires n^2 evaluations. We thus have a speedup factor of $n^2/n = n$.

However, please note that this speedup may not work out for 0.9. If the optimization step in 0.9 requires m gradient evaluations, we will have a total of $m \cdot n$ evaluations for this step. If m remains much lower than n then the approach is fine. However, if m approaches n, there will be no real speedup. This discussion assumes that the full Hessian is computed once and re-used.

In all cases, in terms of memory, the approximation is great in the sense that it avoids storing the Hessian.

3(c)(6 points) Suppose that it takes an average of N_{avg} minimizations to compute τ in the first **if**-statement. Furthermore, suppose that it takes an average of N_{iters} iterations for the full-loop. On average, how many gradient-function evaluations do you need for executing the algorithm?

Answer: We require $N_{iters} + N_{avg}$ gradient function evaluations. Note that for negative curvature, we exit from the loop. We don't have another iteration in this case. So you only need N_{avg} evaluations to account for this case (it does not multiply N_{tiers}). Since the gradient function evaluation requires n component function evaluations, we get $(N_{tiers} + N_{avg}) \cdot n$ regular function evaluations.

3(d)(3 points) Another effective method for large-scale systems is due to the L-BFGS algorithm. Explain the differences between L-BFGS and CG-Steihaug by listing the advantages and disadvantages of each method.

Answer: As described in 3(c), CG-Steihaug can require a large number of additional function evaluations. These function evaluations are not required for L-BFGS. So, in terms of Computational complexity, L-BFGS could prove to be faster.

On the other hand, CG-Steihaug is Hessian-free. It does not impose any memory requirements for storing the Hessian. This is clearly not the case for L-BFGS. For L-BFGS, we need to store the previous gradient-difference vectors y_i and we use them in an effort to approximate the real Hessian. Thus, the L-BFGS has additional memory requirements. Furthermore, for L-BFGS, we need to make sure that we use sufficient y_i to properly approximate the Hessian.

Overall, we expect that CG-Steihaug will be more accurate. Furthermore, the CG-Steihaug can be effective in handing cases with negative curvature.

Problem 4 (15 points) General framework for trust-region methods.
The most efficient algorithms tend to be local. In other words, they do not solve global optimization problems. In this problem, we want to discuss ways to extend these local methods to cover global optimization problems.

Given $\bar{\Delta} > 0$, $\Delta_0 \in (0, \tilde{\Delta})$ and $\eta \in [0, 1/4]$.
for $k = 0, 1, 2, \ldots,$ *MaxIterations* **do**
  Obtain p_k by approximately solving:
    $\min_{p \in \mathbb{R}^n} m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$ such that: $\|p\| \leq \Delta_k$
  Evaluate the reduction ratio:
    $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
  if $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} = \frac{1}{4}\Delta_k$;
  else
    if $\rho_k > \frac{3}{4}$ *and* $\|p_k\| = \Delta_k$ **then**
      $\Delta_{k+1} = \min(2\Delta_k, \bar{\Delta})$
    else
      $\Delta_{k+1} = \Delta_k$
    end
  end
  if $\rho_k > \eta$ **then**
    $x_{k+1} = x_k + p_k$
  else
    $x_{k+1} = x_k$
  end
end

Algorithm 2: General Framework for trust-region Algorithms

4(a)(8 points) Based on the trust-region algorithm, please explain:

- What is the value of ρ_k when the predicted change is exact?

  Answer: $\rho_k = 1$.

- Suppose that ρ_k is very large. What can you say about the model $m_k(.)$. Is it under-estimating or over-estimating the reduction in $f(.)$?

  Answer: Under-estimating.

- Compared to the model, suppose that $f(.)$ is getting reduced very rapidly **within** the trust-region. How will this affect ρ_k? Will the trust-region increase, decrease, or stay the same?

  Answer: Stay the same. Note that $||p_k|| \neq \Delta_k$.

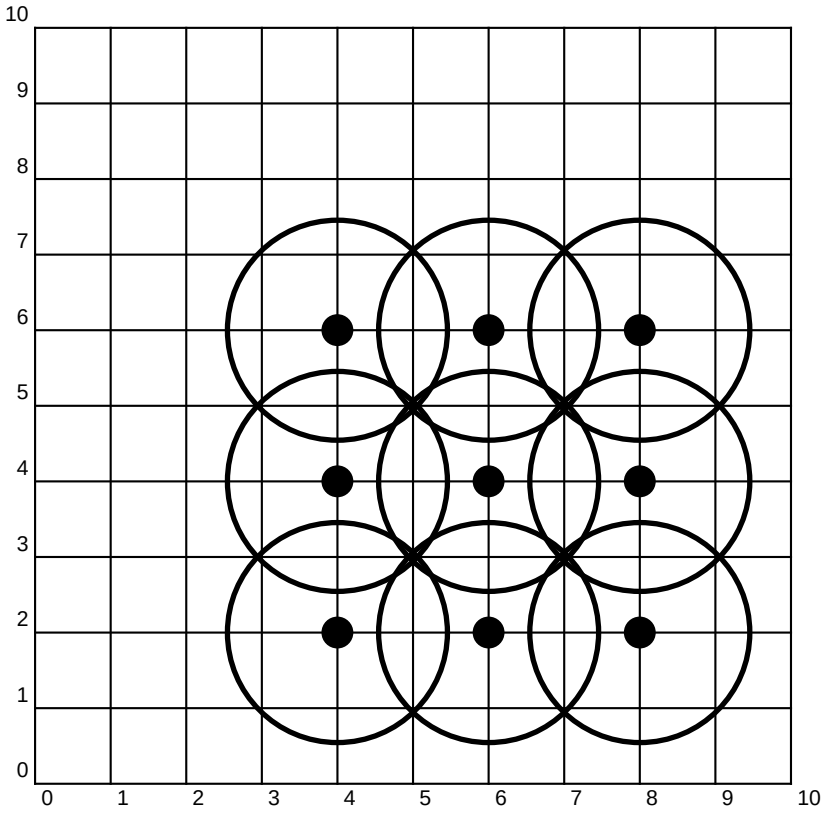
- Compared to the model, suppose that $f(.)$ is getting reduced very slowly. Will the trust-region increase, decrease, or stay the same?

  Answer: Decrease.

4(b)(7 points) Since this is a local-method, your algorithm will not work when you have multiple minima unless you start within the neighborhood of the right minimum! Now, suppose that you know that your algorithm will always converge provided that x_0 is within a certain distance r from the minimum point. Describe how to use this information to design a global algorithm that will always converge to the correct minimum.
Hint: Consider a sampling scheme.

Answer: The basic idea is to start at multiple starting-points and then take the minimum value of all of them. In order for this approach to give you the global minimum, you need to make sure that the starting points are not more than r-distance from a possible minimum. Thus, the starting points will need to use circles of radius r to tile the plane. There are two regular tilings: (i) square tilings (sub-optimal), and (ii) hexagonal tilings (optimal honeycomb-type pattern).

The idea of covering the plane using circles is demonstrated in the figure below.



In this case, from the central circle, note that every possible minimum is within the radius of at-least one circle ($r = \sqrt{2}$ here). So, in order for this to work, the starting points will need to be placed with a regular spacing of $2r/\sqrt{2} = 2$ of each other.

Problem 5 (25 points) Mixed Optimization
We want to minimize $f(x)$ where x is a mixed vector:

$$x = (s,\ q_1, q_2, \ldots, q_r,\ y_1, y_2, \ldots, y_s),$$

where $s \in \{1, 2, \ldots, N\}$ is an **integer state variable**, $q_1, q_2, \ldots, q_r$ are **non-negative integers**, and $y_1, y_2, \ldots, y_s$ are **real numbers**. This a mixed-problem with both integers and real-valued functions. We want to develop an efficient method for solving this type of problem using both stochastic and continuous optimization methods. In this problem, we want to develop a framework for solving this type of problems.

5(a)(4 points) Suppose that we use b-bits for representing $q_1, q_2, \ldots, q_r$. Give the total number of possibilities for $s, q_1, q_2, \ldots, q_r$. Give an example for $r = 3, b = 8, N = 12$.

Answer: We have $N \cdot (2^b)^r = N \cdot 2^{br} = 12 \cdot 256^3$.

5(b)(4 points) Define a neighborhood system that changes a **single** variable at a time by adding or subtracting 1 from $(q_1, q_2, \ldots, q_r)$, while allowing wrap-around. Will this neighborhood system reach all possible vectors? Explain briefly.

Answer: A neighborhood system is to define $(q_1, q_2, \ldots, q_r)$ to be a neighbor of $(q_1, \ldots, q_i \pm 1, \ldots, q_r)$ for any index i. It is clear that this neighborhood system allows us to reach any other state by simply taking unit-steps in each coordinate q_i. For example, for $r = 2$, to reach $(q_1 + \Delta_1, q_2 + \Delta_2)$ from (q_1, q_2), we can take $|\Delta_1|$ steps of $\text{sign}(\Delta_1)$ in q_1 followed by $|\Delta_2|$ steps of $\text{sign}(\Delta_2)$ in q_2.

5(c)(4 points) Recall the Markov-chain transition probability expression:

$$p = \min\left\{1,\ \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}.$$

Recall that convergence requires that we specify $\lambda_n = C\log(1+n)$. Provide a simplified expression for p for this value of λ_n.

Answer: We have:

$$\begin{aligned}
p &= \min\left\{1,\ \frac{\exp \lambda_n f(y)}{\exp \lambda_n f(x)}\right\}, \quad \text{since} |N(x)| = |N(y)| \\
&= \min\left\{1,\ \exp\left[\lambda_n (f(y) - f(x))\right]\right\} \\
&= \min\left\{1,\ \exp\left[C \cdot \log(1+n) \cdot (f(y) - f(x))\right]\right\} \\
&= \min\left\{1,\ \exp\left[\log(1+n)^{C\cdot(f(y)-f(x))}\right]\right\} \\
&= \min\left\{1,\ (1+n)^{C\cdot(f(y)-f(x))}\right\}
\end{aligned}$$

5(d)(13 points) For s, we want to evaluate all possibilities using an exhaustive search. Furthermore, for the q-variables we want to start at a collection of random points. In order for your method to work, assume that you are given an effective **continuous** algorithm for minimizing $f_{s=S, q_1=V_1, q_2=V_2, \ldots, q_r=V_r}(y_1, y_2, \ldots, y_2)$. Provide the pseudo-code for solving the problem. For full-credit, your solution should provide sufficient details on how to handle the stochastic **minimization**.

Answer: In applying Simulated Annealing, we need to minimize f by maximizing $-f$.

```
2.1  Given f_{s=S, q1=V1,q2=V2,...,qr=Vr}(y1, y2, ..., y2).
2.2  Initialize x_min = ∞, f_min = ∞;
2.3  Evaluate all possible states:
2.4  for s = 1, 2, 3, ..., N do
2.5      Generate MaxPoints random starting points:
2.6      for pt = 1, 2, 3, ..., MaxPoints do
2.7          Generate a random initial point:
2.8          for i = 1, 2, 3, ..., r do
2.9              V_i = round (U(0, 1) · 256 − 0.499);
2.10         end
2.11
2.12         Apply Simulated Annealing:
2.13         [x_min1, f_min1] = SimulatedAnnealing(f_{s=S, q1=V1,q2=V2,...,qr=Vr});
2.14
2.15         Update the minimum point:
2.16         if f_min1 < f_min then
2.17             x_min = x_min1, f_min = f_min1;
2.18         end
2.19     end
2.20 end
```

Algorithm 3: Algorithm.

Inputs: $f_{s=S,\, q_1=V_1, q_2=V_2, \ldots, q_r=V_r}(y_1, y_2, \ldots, y_2)$, x_0.
Outputs: x_{max}, f_{max}
Initialize the maximum value:
$f_{max} = f_{s=S,\, q_1=V_1, q_2=V_2, \ldots, q_r=V_r}(x_0)$;
$x_{max} = x_0$, $x = x_0$, $f_x = f_{max}$;
for $i = 1, 2, 3,$ *MaxIterations* **do**
    Randomly pick a neighbor:
    $y = x$;
    $j = \text{Round}(U(0,1) \cdot r - 0.499)$;
    if $U(0,1) < 0.5$ **then**
        $y_{q_j} = y_{q_j} + 1$;
    else
        $y_{q_j} = y_{q_j} - 1$;
    end
    Minimize $f_y = f_{s=S,\, q_1=V_1, q_2=V_2, \ldots, q_r=V_r}(.)$ starting at random point y_0;

    Decide whether to move to the new location using $-f(.)$:
    if $-f_y > -f_x$ **then**
        Jump to the new location and update max:
        $x = y$, $f_x = f_y$;
        $x_{max} = x$, $f_{max} = f_y$;
    else
        Jump with probability p:
        $p = \min\left\{1,\, (1+n)^{C \cdot (-f_y + f_x)}\right\}$;
        if $U(0,1) < p$ **then**
            Jump to y:
            $x = y$, $f_x = f_y$;
        else
            Stay at x.;
        end
    end
end

Algorithm 4: Minimization using Simulated Annealing and given function.