Praktika zur Vorlesung „Mathematische Optimierung“

Markus Lange-Hegermann **(WiSe 2024/25)**

Bewertungsregeln

Die folgenden Aufgaben fragen nach Ergänzungen am Code. Geben Sie sich Mühe, dass Ihre Ergänzungen die folgenden 3 Adjektive erfüllen:

- **effizient**: gemessen in Laufzeit und Auswertungen von Funktionswerten und Gradienten der Zielfunktion. Kommentiere jeweils O-Notation, Entscheidungen für bessere Effizienz, resultierende Laufzeiten und Auswertungen.
- **wartbar**: baue Implementierung geeignet in die vorhandene Codebasis ein (betrachte dazu die Codebasis vorher), mache geeignete und umfangreiche Tests, halte die Anzahl der Codezeilen kurz.
- **lesbar**: kommentiere wichtige Ideen, wähle geeignete Strukturen, separiere Ideen im Code.

Die Lösung wird primär nach diesen Kriterien beurteilt! **Begründen** Sie dabei immer kurz, warum Sie gewisse Entscheidungen getroffen haben. **Vergleichen** Sie Otimierungsergebnisse und Rechengeschwindigkeiten zwischen existierenden und neuen Implementierungen. Machen Sie sich auch sonst selber Gedanken, wie Ihre Lösung hilfreich sein kann. Ich werde signifikant Punkte für Lösungen abziehen, welche mit wenig Aufwand und in der Praxis wenig hilfreich implementiert sind.

Zu bearbeitende Teilaufgaben: Von jeder Aufgabe ist nur eine Teilaufgabe zu bearbeiten. Wenn in der Aufgabenstellung ein λ steht, dann muss in dieses λ die Matrikelnummer eingesetzt werden. Dies ergibt die zu bearbeitende Teilaufgabe. Ansonsten stehen explizite Anweisungen dabei.

Bonuspunkte:

- Einige Aufgaben geben Bonuspunkte.
- In allen weiteren Aufgaben kann man Bonuspunkte erhalten, indem man die auf die eigene Teilaufgabe folgende Aufgabe löst (man springt zurück auf Teilaufgabe 1, wenn man die letzte Teilaufgabe hat). Diese Aufgabe gibt jedoch nur ein Drittel (aufgerundet) der Punkte.
- Einzelne Aufgaben haben kleine Zusatzaufgaben mit Bonuspunkten

Abgaben von Code als git-patch.
Weitere Abgaben (nur nach Absprache) schriftlich per Mail (pdf, txt, ...).

Aufgabe 1. (Simplex-Verfahren - 20 Punkte)
Implementiere das Downhill-Simplex-Verfahren.

Hinweis: Machen Sie sich noch die Bewertungskriterien klar und implementieren Sie dahingehend.

Bonusaufgabe (Bounds im Simplex-Verfahren - 10 Punkte)
Ergänze Bounds und Ungleichungsbedingungen in der Implementierung.

Aufgabe 2. (Optimierungsaufgaben 1 - 10 Punkte)
`lambda a: ((round((a+1)/5)+1)%4)+1`

1. Die Optimierung (e.g. in BFGS) von $x \mapsto x^4$ (und ähnlichen Funktionen, u.a. `test_BFGS4` und `test_BFGS5`) hat hohen x-Fehler. Warum? Was könnte man tun? Bei `test_BFGS5` ist ein weiteres Problem, kommentiere dies.

2. In den Optimierungen sind die Funktionen nicht skaliert. Welche Probleme kann eine fehlende Skalierung (Inputs&Outputs) bei den jeweiligen Methoden mit sich bringen? Wie kann man eine automatische oder manuelle Skalierung mit einbauen? Implementiere diese.

3. Implementiere eine Version des Quasi-Newton Verfahrens mit BFGS, welches mit Bounds umgehen kann. Die Bounds sollen im line search sichergestellt werden.

4. Baue Bounds sinnvoll in SQP und den dazugehörigen Line Search ein. (Die aktuelle Version ist nicht geeignet.)

Aufgabe 3. (Optimierungsaufgaben 2 - 20 Punkte)
`lambda a: ((round((a+1)/11)+2)%6)+1`

1. Implementiere eine Multistart-Optimierung als globale Optimierung. Sie soll mit möglichst allen anderen lokalen Optimierungsverfahren zusammenarbeiten können. Überlege eine Strategie, wie viele und welche Startpunkte gewählt werden.

2. Implementiere SGD und teste dies mit den neuronalen Netzen.

3. Unsere SQP-Implementierung hat Probleme mit Optima am Rand. Woran liegt das? Wie könnte man das Problem beheben? Implementiere dies.

4. Implementiere die Ergänzungen Momentum beim Gradientenabstieg

5. Die Implementierung in SQP verwendet nur einen sehr einfachen Line Search mit den Constraints. Implementiere einen neuen Line Search, der Constraints überprüft, aber auch die starken Wolfe Bedingungen sicherstellt.

6. Implementiere die Konstruktion von Hessematrizen (mit Hilfe der Kettenregel) und darauf aufbauend ein Newton-Verfahren.

Aufgabe 4. (Codeaufgaben 1 - 10 Punkte)
`lambda a: ((round((a+2)/3)+2)%4)+1`

1. Implementiere Memoization. Potentielle Fragen: Vergleiche auf Basis von (etwas teureren) Gleichheitsvergleichen oder auf Vergleich der Referenzpointer? Wie viel Historie muss gespeichert werden? Wie sehr verbessert das die späteren Optimierung?

2. Implementierung 10 weitere Funktionenklassen (exp, sin, ...). Nutze diese, um mehr und spannende Beispiele in späteren Optimierungsproblemen zu entwickeln.

3. Implementiere die Funktion $f(x) = \frac{\sqrt{x^3+2x^2-x+1} \cdot e^{\sin(x^2)}}{\log(x^4+2)+\cos^{-1}\left(\frac{x}{2}\right)}$ als `IDifferentiableFunction`.

4. Manche Unittests haben zufällige Elemente. Welche Vor- und Nachteile bringt dies mit sich? Ändere die Implementierung, so dass diese ohne (die Nachteile des) Zufall auskommt. Wie kann man die Vorteil des Zufalls besser nutzen?

Aufgabe 5. (Codeaufgaben 2 - 30 Punkte)
`lambda a: ((round((a+6)/7)+2)%4)+1`

1. `Intersection` erhöht die Anzahl der Ungleichheitsbedingungen. Warum ist das ungünstig? Welche Auswirkungen kann das haben? Was kann man (heuristisch?) dagegen tun?

2. Werde die Multiplikation mit diagonalen Jakobimatrizen (ReLU, Vektoraddition, Identität, Nullfunktion, ...) los. Mache ähnliche Vorschläge, um Blockdiagonalmatrizen loszuwerden (dieser muss nicht implementiert werden).

3. Spezifiziere Interfaces für Körper, Vektorräume und Matrizen (als lineare Abbildungen) nach Curry-Howard. Nutze diese Schnittstellen, um den Gaußalgorithmus zu implementieren und an Beispielen durchzuführen. Die Rechenschritte sollen mittels Elementarmatrizen durchgeführt werden. Diskutiere die Laufzeit dieser Implementierung und Vergleiche diese mit der üblichen Version des Gaußalgorithmus.

4. Das Erstellen eines Punktes innerhalb einer Menge ist zeitaufwändig. Welche Alternativen hat man, dies schneller zu haben? Warum sind zufällige Startpunkte oft hilfreich? Wie kann man das Finden von Punkten deterministisch gestalten? Welche Vorteile hat dies? Welche Operationen gehen schief, wenn man immer den gleichen Punkt deterministisch wählt?

Aufgabe 6. (Codeaufgaben 3 - 10 Punkte)
`lambda a: ((round((a+3)/29)+1)%4)+1`

1. Kritisiere den Code in Sachen Wartbarkeit. Schlage eine konkrete Verbesserung vor und implementiere sie.

2. Kritisiere den Code in Sachen Geschwindigkeit. Schlage eine konkrete Verbesserung vor und implementiere sie.

3. Kritisiere den Code in Sachen Lesbarkeit. Schlage eine konkrete Verbesserung vor und implementiere sie.

4. Wähle einen (für die folgenden Punkte) nicht-trivialen Teilalgorithmus der vorhandenen Implementierung. Diskutiere von diesem Algorithmus theoretisch die Laufzeit und Speicherverbrauch in $\mathcal{O}$-Notation. Profile den Algorithmus für verschiedene Eingabegrößen. Vergleiche die Profiles mit den theoretischen Vorhersagen.

Aufgabe 7. (Weiterführende Theorieaufgaben - 10 Punkte)
`lambda a: ((round((a+8)/13)+3)%3)+1`

1. Wie kann man mit Pareto-Optimierung umgehen? Welche unserer Optimierungsverfahren bieten sich an, welche nicht? Kommentiere, welche Anpassungen am Code gemacht werden müssten.
2. Wie kann man mit verrauschten Daten umgehen? Welche unserer Optimierungsverfahren bieten sich an, welche nicht? Kommentiere, welche Anpassungen am Code gemacht werden müssten.
3. Wie kann man mit hochdimensionalen zulässigen Mengen umgehen? Welche unserer Optimierungsverfahren bieten sich an, welche nicht? Kommentiere, welche Anpassungen an unseren Code gemacht werden müssten.

Hinweis: hier ist explizit keine Implementierung gefragt. Eine Lösung geht aber auf vorhandene Implementierungen ein.

Aufgabe 8. (GPs - 20 Punkte)
`lambda a: ((round((a+4)/17)+2)%3)+1`

1. Die GP Implementierung berechnet die Ableitungen der Evaluation und der Varianz direkt. Begründe, warum diese Formel korrekt ist. Implementiere dies alternativ mit Hilfe unseres Frameworks nach. Diskutiere Vor- und Nachteile beider Ansätze.

2. In der Implementierung der GPs sind mehrere Dinge nicht numerisch stabil (noise/posdef, keine Benutzung der Cholesky-Zerlegung, …). Nutze dies aus, um in der vorgegebenen Implementierung einen Fehler zu konstruieren. Mache den Code numerisch stabil.

3. Zur Berechnung von `alpha` in den GPs nutzen wir noch nicht die Cholesky-Zerlegung (an anderen Stellen schon). Implementiere diese Berechnung mittel Cholesky-Zerlegung.

Aufgabe 9. (GPs with a view towards BO - 20 Punkte)
`lambda a: ((round((a+16)/19)+1)%2)+1`

1. Implementiere andere GP Kerne, informiere dich dabei, was in der Bayesian Optimization verwendet wird. Mache Vergleiche, ob dies die Qualität der Optimierung verbessert. (Hier sind gewisse Ableitungen zu bestimmen.)

2. Bei BO werden vom GP mache Dinge doppelt ausgerechnet. Welche? Optimiere den Code hier.

Aufgabe 10. (BO - 10 Punkte)
`lambda a: ((round((a+19)/23)+3)%4)+1`

1. Typisch in BO Anwendungen ist, dass schon Daten vorhanden sind, bevor die Optimierung gestartet wird. Erweitere die Schnittstelle, um dies zu erlauben.

2. Implementiere die Acquisition Function namens Expected Improvement. Vergleiche, ob dies die Qualität der Optimierung verbessert.

3. Implementiere die Acquisition Function namens Probability of Improvement. Vergleiche, ob dies die Qualität der Optimierung verbessert.

4. Bei der BO-Berechnung verwenden wir die Standardabweichung von GPs. Diese wird durch eine Wurzel aus der Varianz berechnet. Warum wird die Wurzel nie aus einer negativen Zahl gezogen? Was muss man beim Gradienten der Wurzel beachten? Inwiefern ist das bei uns wirklich ein Problem?

Aufgabe 11. (Diskrete Optimierung 1 - 10 Punkte)
Sie haben freie Wahl bei der Teilaufgabe.

1. Nehmen Sie sich eine vorhandene Implementierung des DPLL-Algorithmus (oder einer Verbesserung davon). Baue eine (eigene, neue) Klasse von Testbeispielen, welche in der Größe variiert werden kann. Vergleiche die Rechenzeiten, wenn die Größe wirklich groß wird.

2. **(Varianten von TSP)**
   Seit das himmlischen Großrechnersystem in die Wolke verlegt wurde, gibt es andauernd Abstürze. Heute ist es mal wieder so weit, dabei wartet der Weihnachtsmann gerade jetzt ganz dringend auf den Ausdruck seiner Rundtour für die Weihnachtsnacht. Würde der Himmelsrechner funktionieren, wäre das alles gar kein Problem, schließlich wurde der Rechner vor gar nicht allzu langer Zeit von Alan entworfen und verfügt über diesen tollen Nichtdeterminismus. Nun aber muss der Weihnachtsmann ein Verfahren entwerfen, mit dem er auch nach irdischen Maßstäben effizient seine Weihnachtstour berechnen kann. Die Distanzen zwischen den einzelnen Orten, zu denen er muss, kennt er natürlich. Aber wie jedes Jahr stellt sich das Problem, dass er an keinem Ort zweimal auftauchen darf (wegen neugieriger Kinder). Außerdem würden seine Rentiere dauerhaft in Streik treten, wenn sie herausbekommen würden, dass sie nicht die kürzeste Route genommen haben. Glücklicherweise ist sein treues Leitrentier Rudolph in der Lage, zu jeder vorgelegten Eingabe von Orten und ihren Distanzen, effizient zu entscheiden, ob es eine Tour mit den geforderten Eigenschaften der Länge höchstens b Kilometer gibt oder nicht. Allerdings wissen Rudolph und der Weihnachtsmann noch nicht, wie sie daraus eine Lösung für das ursprüngliche Problem entwickeln können.
   Formal: Zeige, falls die Entscheidungsvariante des TSP in P ist, so ist auch das TSP in P.

Hinweis: Die zweite Teilaufgabe ist viel weniger Arbeit.

Bonusaufgabe (Diskrete Optimierung 2 - 20 Bonuspunkte)
Sie haben freie Wahl bei der Teilaufgabe und können auch gerne beide Teilaufgaben bearbeiten.

1. Implementieren Sie den DPLL-Algorithmus. Vergleichen Sie die Laufzeiten zu vorhandenen Algorithmen.

2. **(Varianten von BPP)**
   Ein weiteres Problem stellt sich dem Weihnachtsmann bei der Verteilung der Geschenke in seine Weihnachtssäcke. Sparsam, wie man im Himmel ist, soll er mit der minimalen Anzahl von Säcken auskommen, die für die ganzen Geschenke benötigt werden. Allerdings ist noch unklar, wie die Geschenke auf die Säcke verteilt werden müssen. (Früher war das alles kein Problem, aber seit der Wolke …) Jedenfalls hat jedes Geschenk sein eigenes Gewicht und jeder Sack kann maximal b Kilogramm tragen ohne zu reißen. Zum Glück für den Weihnachtsmann, hat eine der Feen im himmlischen Einkauf ein Verfahren entwickelt, mit dem sie effizient entscheiden kann, ob eine bestimmte Menge von Geschenken auf eine bestimmte Anzahl von Säcken mit einer ebenfalls bestimmten zulässigen maximalen Traglast verteilt werden kann oder nicht.
   Formal: Zeige, falls die Entscheidungsvariante des BPP in P ist, so ist auch das BPP in P.

   Hinweis zur BPP-Aufgabe: Vorsicht: schwer! Lösung per Mail direkt an den Dozenten. Volle Bonuspunktzahl nur an die erste (halbwegs richtige) Abgabe. Spätere Abgaben erhalten höchsten 10 oder 5 Bonuspunkte.