

2. Projekt

Quasi-Newton-Verfahren & Gauß-Newton-Verfahren

im Fach

Numerische Optimierung

Juni 2020

Maximilian Gaul

Aufgabe 1

Siehe Programmcode in Project2.m.

Aufgabe 2

$$\begin{aligned}
 & I - \frac{s(A s)^T}{s^T A s} + \frac{A^{-1} y y^T}{y^T s} + \frac{(s - A^{-1} y) s^T A + s(s - A^{-1} y)^T A}{y^T s} - \frac{(s - A^{-1} y) s^T A s(A s)^T}{y^T s s^T A s} + \frac{(s - A^{-1} y) s^T y y^T + s(s - A^{-1} y)^T y y^T}{(y^T s)^2} - \frac{(s - A^{-1} y)^T y s s^T A s(A s)^T}{(y^T s)^2 s^T A s} - \frac{(s - A^{-1} y)^T y s s^T A s}{(y^T s)^2 y^T s} \\
 & I - \frac{s s^T A}{s^4 A s} + \frac{A^{-1} y y^T}{y^T s} + \frac{s s^T A - A^{-1} y s^T A + s(s^T - (A^{-1} y)^T) A}{y^T s} - \frac{s s^T A s s^T A - A^{-1} y s^T A s s^T A + s(s^T - (A^{-1} y)^T) A s s^T A}{y^T s s s^T A s} + \frac{s s^T y y^T - A^{-1} y s^T y y^T + s(s^T - (A^{-1} y)^T) y y^T}{(y^T s)^2} - \frac{(s^T - (A^{-1} y)^T) y s s^T A}{(y^T s)^2 s^T A s} + \frac{(s^T - (A^{-1} y)^T) y s s^T A s s^T A}{(y^T s)^2 y^T s} \\
 & I - \frac{s s^T A}{s^4 A s} + \frac{A^{-1} y y^T}{y^T s} + \frac{2 s s^T A - A^{-1} y s^T A - s y^T}{y^T s} - \frac{2 s s^T A - A^{-1} y s^T A - s y^T}{y^T s s^T A s} \cdot s s^T A + \frac{2 s s^T - A^{-1} y s^T - s y^T}{y^T s s^T A s} \cdot T - \frac{s^T y - y^T (A^{-1})^T}{(y^T s)^2} \cdot s s^T A + \frac{(s^T - y^T (A^{-1})^T) y s s^T A s s^T A}{(y^T s)^2 s^T A s} - \frac{(s^T - y^T (A^{-1})^T) y s s^T y y^T}{(y^T s)^2 y^T s}
 \end{aligned}$$

Aufgabe 3

Wenn die Suchrichtung des BFGS Verfahrens:

$$d = -B \cdot \nabla f(x)$$

keine Abstiegsrichtung ist, d.h. die Bedingung:

$$\nabla f(x)^T \cdot d < 0$$

nicht erfüllt ist, muss das Verfahren 'resettet' werden. In diesem Fall bietet es sich an, die Suchrichtung auf den negativen Gradienten zu setzen:

$$d = -\nabla f(x)$$

Da nun die Abstiegsrichtung nicht mehr zur approximierten Inversen der Hesse-Matrix B passt, muss diese ebenfalls für den nächsten Schritt neu bestimmt werden (bzw. das nächste Update erfolgt dann mit dieser Matrix).

Hierzu bieten sich verschiedene Möglichkeiten an:

- Wie beim Start des BFGS-Verfahrens $B = I$ setzen
 - Hierbei geht jeglicher berechnete Fortschritt verloren, es handelt sich um einen recht naiven Ansatz
- Die Hesse-Matrix einmalig aus Differenzenquotienten des Gradienten bestimmen und anschließend invertieren
 - Hoher Rechenaufwand von $\mathcal{O}(n^2)$ für die Hesse-Matrix und nochmal $\mathcal{O}(n^3)$ für das Invertieren
 - Problem wenn die Hesse-Matrix nicht invertierbar ist bzw. aufgrund von Auslöschung oder anderen numerischen Fehlern die Inverse schlecht konditioniert ist
 - Bisher berechneter Fortschritt geht ebenfalls verloren aber die Approximation der Hesse-Matrix ist sehr genau
- Man könnte, wie in den Vorlesungsfolien beschrieben, $\frac{y^T s}{y^T y} \cdot I_n$ als positiv-definitve Matrix verwenden

Um herauszufinden, welche dieser Methoden am besten geeignet ist (d.h. die richtig Lösung in der kürzesten Zeit findet), wird die Laufzeit des inversen BFGS-Verfahrens bestimmt. Startwerte:

- N-dim. Rosenbrock-Funktion: $\begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix}$
- Himmelblau-Funktion: $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$

- Bazaraa-Shetty: $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$

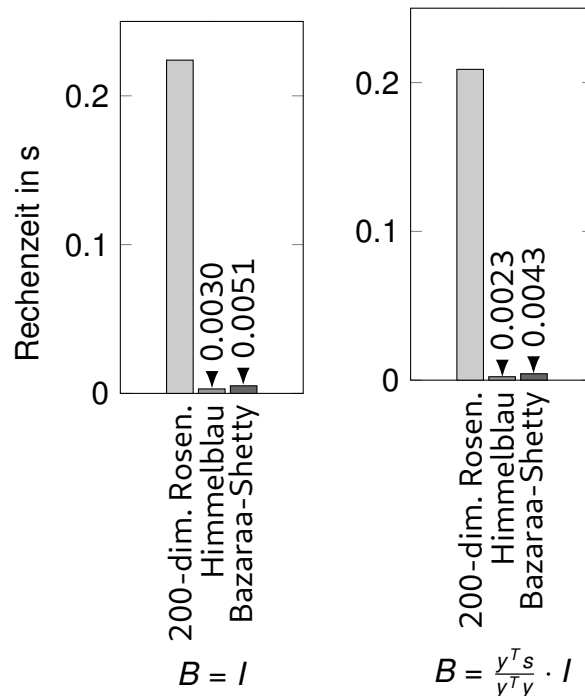


Abbildung 1: Vergleich der Rechenzeit für eine Genauigkeit von 10^{-8} gemittelt über 100 Durchläufe (Intel Core i3-7100U, 8GB RAM, Windows 10 64-Bit)

Alle drei Funktionen profitieren von $B = \frac{y^T s}{y^T y} \cdot I$, daher habe ich diesen Weg in meiner Implementierung gewählt.

Aufgabe 4

Die Ableitung der N-dimensionalen Rosenbrock-Funktion habe ich beispielhaft für $N = 3$ bestimmt:

$$\begin{aligned}
 & \sum_{i=1}^2 (1 - x_i)^2 + 100 \cdot (x_{i+1} - x_i^2)^2 \\
 & = \\
 & (1 - x_1)^2 + 100 \cdot (x_2 - x_1^2)^2 + (1 - x_2)^2 + 100 \cdot (x_3 - x_2^2)^2 \\
 \nabla f_3 = & \begin{bmatrix} -2 \cdot (1 - x_1) + 200 \cdot (x_2 - x_1^2) \cdot (-2x_1) \\ 200 \cdot (x_2 - x_1^2) - 2 \cdot (1 - x_2) + 200 \cdot (x_3 - x_2^2) \cdot (-2x_2) \\ 200 \cdot (x_3 - x_2^2) \end{bmatrix}
 \end{aligned}$$

Man erkennt eine Regel: Der Gradient besteht aus drei Teilen. Der erste Eintrag im Gradienten ist:

$$-2 \cdot (1 - x_1) + 200 \cdot (x_2 - x_1^2) \cdot (-2x_1)$$

Alle weiteren Einträge (bis auf den letzten an Position $N - 1$) sind:

$$200 \cdot (x_i - x_{i-1}^2) - 2 \cdot (1 - x_i) + 200 \cdot (x_{i+1} - x_i^2) \cdot (-2x_i)$$

Der letzte Eintrag ist:

$$200 \cdot (x_N - x_{N-1}^2)$$

Die Ableitung ist in Projekt2.m in der Funktion f_rosen_mult_deriv_func definiert. Rechenzeit des in InverseBFGS.m implementierten Verfahrens:

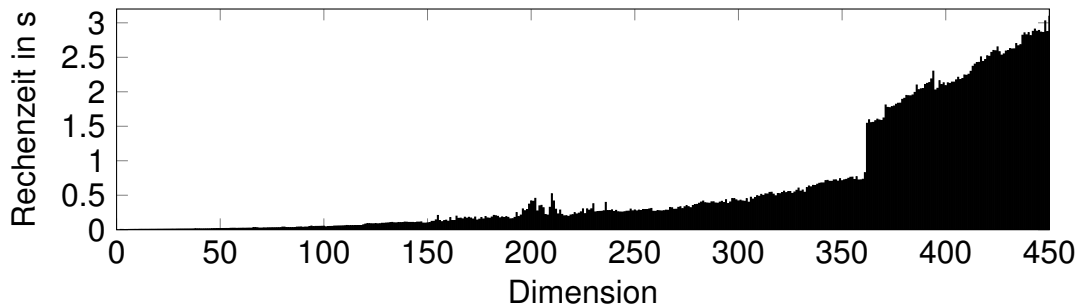


Abbildung 2: Rechenzeit der N-dimensionalen Rosenbrock-Funktion mit Startwert $[-1, \dots, -1]^T$ für eine Genauigkeit von 10^{-8} gemittelt über 100 Durchläufe (Intel Core i3-7100U, 8GB RAM, Windows 10 64-Bit)

Rechenzeit von fminunc mit Option "HessUpdate" = "bfgs":

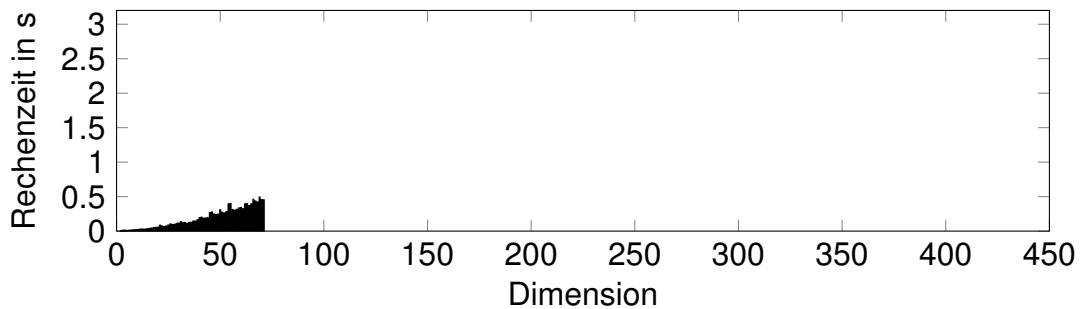


Abbildung 3: Rechenzeit der N-dimensionalen Rosenbrock-Funktion mit Startwert $[-1, \dots, -1]^T$ für eine Genauigkeit von 10^{-8} gemittelt über 100 Durchläufe (Intel Core i3-7100U, 8GB RAM, Windows 10 64-Bit)

Man erkennt, dass mit höheren Dimensionen die Rechenzeit für beide Verfahren drastisch zunimmt.

Möglichkeiten um ein höheres N zu erreichen:

- Nach den Regeln der Analysis lässt sich eine Summe aufteilen in zwei Summen:

$$\sum_{i=1}^{N-1} (1 - x_i)^2 + 100 \cdot (x_{i+1} - x_i^2)^2$$

$$= \sum_{i=1}^a (1 - x_i)^2 + 100 \cdot (x_{i+1} - x_i^2)^2 + \sum_{i=a+1}^{N-1} (1 - x_i)^2 + 100 \cdot (x_{i+1} - x_i^2)^2$$

Die Berechnung des Minimums dieser zwei Summen kann auf z.B. zwei Threads aufgeteilt und am Ende wieder zusammengefügt werden. Bei ungeradem N muss man sich entscheiden wie die Summe aufgeteilt wird, ein Thread bearbeitet dann eine Dimension mehr als der andere. Bei besonders großem N können auch diese beiden Summen wiederum aufgeteilt und somit auf noch mehr Threads verteilt werden.

- Verwenden von sparse-Matrizen und Vektoren durch die sich die Rechenzeit unter Umständen reduzieren kann. Sparse-Datenstrukturen verwenden eine spezielle Repräsentation der Werte in denen Einträge mit 0 effizienter gespeichert werden. Aufgrundessen beschleunigt sich die Berechnung von Matrix-Vektor-Produkten (die beim BFGS-Verfahren sehr oft verwendet werden)
- Gegebenfalls genauere Untersuchungen über Kondition und Stabilität der Operationen bzw. wie diese verbessert werden können. Durch Auslöschung in der Update-Formel könnte es unter Umständen zu fehlerhaften Richtungsvektoren kommen die sich bei besonders großen Problemen potenzieren und somit die Konvergenz verlangsamen.

Aufgabe 5

Aufgabe 6

Siehe GaussNewton.m.

Aufgabe 7

Siehe Projekt2.m.

Für die erste Funktion $f(t, x_1, x_2) = x_1 \cdot e^{x_2 \cdot t}$ erhält man mit dem gegebenen Datensatz und dem Startwert $x_0 = [1, 1]^T$ die Werte $x_1 = 1.9950$, $x_2 = -1.0095$ nach 10 Iterationen. Abbruchkriterium war $\|J(x)^T \cdot r(x)\| \leq 10^{-8}$.

Für die zweite Funktion $g(t, x_1, x_2, x_3) = x_1 \cdot e^{-(x_2^2 + x_3^2) \cdot t} \cdot \frac{\sinh(x_3^2 \cdot t)}{x_3^2}$ erhält man mit dem gegebenen Datensatz und den Startwerten:

- $x_0 = [10, 0.05, 0.1]^T$
- $x_0 = [7, 0.125, 0.25]^T$
- $x_0 = [3, 0.1, 0.05]^T$

die Werte

- $x_1 = 3.5355$, $x_2 = 0.0546$, $x_3 = 0.1539$

nach 117, 247 bzw. 609 Iterationen. Abbruchkriterium war $||J(x)^T \cdot r(x)|| \leq 10^{-8}$.

Allgemein reagiert die Funktion g sehr empfindlich auf kleine Änderungen in den Startwerten. Das liegt vermutlich an den stark unterschiedlichen Werten der einzelnen Faktoren ($x_1 \cdot e^{-(x_2^2 + x_3^2) \cdot t}$ ist sehr klein während $\frac{\sinh(x_3^2 \cdot t)}{x_3^2}$ sehr groß wird) aufgrundessen es zu numerischer Instabilität kommt (z.B. zeigt Matlab dann NaN).

Aufgabe 8

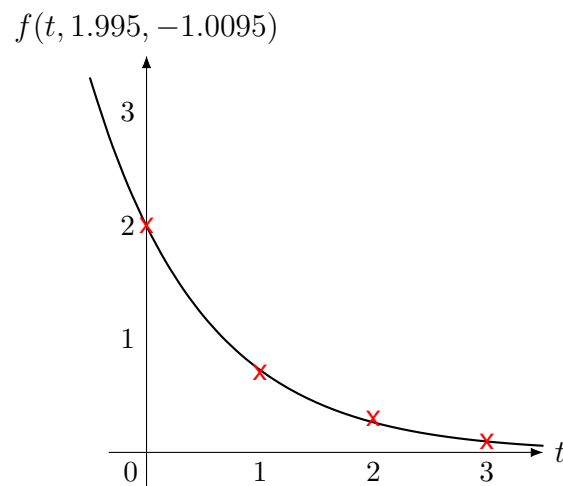


Abbildung 4: Modellfunktion und Datensatz für $f(t, x_1, x_2) = x_1 \cdot e^{x_2 \cdot t}$

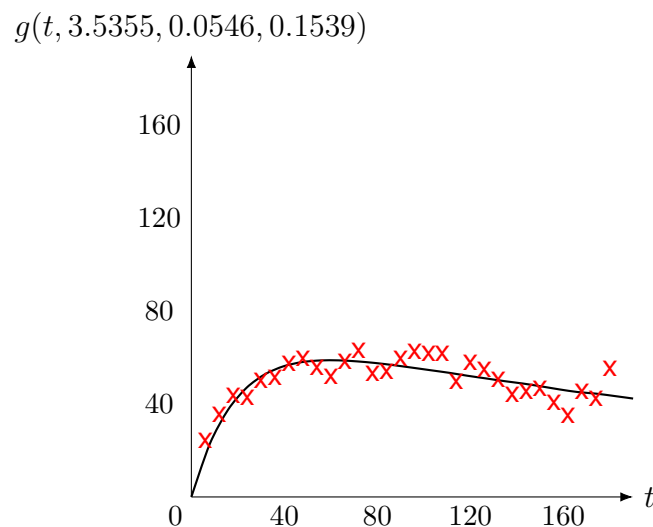


Abbildung 5: Modellfunktion und Datensatz für $g(t, x_1, x_2, x_3) = x_1 \cdot e^{-(x_2^2 + x_3^2) \cdot t} \cdot \frac{\sinh(x_3^2 \cdot t)}{x_3^2}$

Aufgabe 9

Bei einem Least-Squares-Problem wird der Abstand von einer Funktion f zu den gegebenen Datenpunkten (t, y) minimiert, die Zielfunktion (nach Foliensatz) ist daher:

$$f_{LeastSquares}(x) = \sum_{i=1}^m (f(t_i, x_1, x_2, \dots, x_n) - y_i)^2$$

Es werden also alle Residuen:

$$r(x) = \begin{bmatrix} f(t_1, x_1, \dots, x_n) - y_1 \\ f(t_2, x_1, \dots, x_n) - y_2 \\ \vdots \end{bmatrix}$$

quadriert und aufsummiert. $f_{LeastSquares}(x)$ ist somit der erste Parameter, den InverseBFGS erhält.

Weiterhin ist die Jacobi-Matrix definiert als:

$$J = \begin{bmatrix} \frac{df}{dx_1}(t_1, x_1, \dots, x_n) & \frac{df}{dx_2}(t_1, x_1, \dots, x_n) & \dots \\ \frac{df}{dx_1}(t_2, x_1, \dots, x_n) & \frac{df}{dx_2}(t_2, x_1, \dots, x_n) & \dots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Nach Foliensatz ist der Gradient definiert: $\nabla f(x) = 2 \cdot J(x)^T \cdot r(x)$. Dies ist der zweite Parameter, den das BFGS-Verfahren erhält.