Serie 10

Abgabedatum: Fr. 12.05 (ev. 19.05), in den Übungsgruppen

Forum: https://forum.math.ethz.ch/c/spring-23/numerische-methoden-phys/

Webpage: https://metaphor.ethz.ch/x/2023/fs/401-2664-00L/

1. *Gram–Schmidt-Verfahren und Householder-Transformation*

In der Vorlesung haben wir die Householder-Transformation verwendet um die **QR**-*Zerlegung* einer Matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ zu bestimmen. Ein weiteres, sehr intuitives Verfahren, das sukzessive die Spalten $\underline{a}_1, \ldots, \underline{a}_n$ mit $\underline{a}_i \in \mathbb{R}^m$, von $\mathbf{A}$ orthogonalisiert, ist das Gram–Schmidt-Verfahren. Das Gram–Schmidt-Verfahren ist ein Standardwerkzeug in Beweisen der Linearen Algebra. Die folgenden Algorithmen (in Pseudo-Code) liefern eine **QR**-Zerlegung nach dem *Gram–Schmidt-Verfahren* und dem *modifizierten Gram–Schmidt-Verfahren*:

Gram–Schmidt:

for $j = 1, \ldots, n$ **do**
  $\mathbf{v_j} = \mathbf{A}_{:j}$
  for $i = 1, \ldots, j-1$ **do**
    $\mathbf{R}_{ij} = \mathbf{q_i}^{\mathrm{T}}\mathbf{a_j}$
    $\mathbf{v_j} = \mathbf{v_j} - \mathbf{R}_{ij}\mathbf{q_i}$
  end for
  $\mathbf{R}_{jj} = \|\mathbf{v_j}\|_2$
  $\mathbf{Q}_{:j} = \frac{\mathbf{v_j}}{\mathbf{R}_{jj}}$
end for

Modifiziertes Gram–Schmidt:

for $j = 1, \ldots, n$ **do**
  $\mathbf{v_j} = \mathbf{A}_{:j}$
  for $i = 1, \ldots, j-1$ **do**
    $\mathbf{R}_{ij} = \mathbf{q_i}^{\mathrm{T}}\mathbf{v_j}$
    $\mathbf{v_j} = \mathbf{v_j} - \mathbf{R}_{ij}\mathbf{q_i}$
  end for
  $\mathbf{R}_{jj} = \|\mathbf{v_j}\|_2$
  $\mathbf{Q}_{:j} = \frac{\mathbf{v_j}}{\mathbf{R}_{jj}}$
end for

a) Implementieren Sie die beiden Gram–Schmidt-Verfahren in `ortho.py` und verwenden Sie beide Verfahren, um die **QR**-Zerlegung der Matrix $\mathbf{Z} \in \mathbb{R}^{m\times m}$ mit den Einträgen:

$$\mathbf{Z}_{ij} = 1 + \min(i, j), \quad 0 \le i, j < m$$

für $m = 50$ zu bestimmen.

b) Vergleichen Sie die Güte der beiden Gram–Schmidt-Verfahren in Bezug auf die Orthogonalität der Spalten von **Q**.

c) Warum sind die Gram–Schmidt-Verfahren im Gegensatz zur Householder-Transformation (siehe `ortho.py`) ungeeignete numerische Methoden zur Berechnung von **QR**-Zerlegungen?

d) Schreiben Sie einen Code, der die Matrizen $R_1, \ldots, R_m$ aus der Vorlesung zur Umschreibung des Gram–Schmidt-Verfahrens als Multiplikation mit oberen Rechteckmatrizen berechnet. Verwenden Sie die Matrix **Z** aus Aufgabe a) mit $m = 4$ und geben Sie Matrizen R_k und die Skalarprodukte der orthonomisierten Vektoren in jedem Schritt aus.

Bitte wenden!

2. *Konditionszahl*

Für $A \in \mathbb{R}^{m\times n}$, $m \geq n$, ist die Konditionszahl definiert durch

$$\mathrm{cond}(A) = \frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|}.$$

Sei $A = QR$ die QR-Zerlegung von A mit $R = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}$. Zeigen Sie, dass für die zur euklidischen Norm gehörende Konditionszahl cond_2 gilt:

1. $\mathrm{cond}_2(A) = \mathrm{cond}_2(R) = \mathrm{cond}_2(\tilde{R}) \geq \frac{\max_{i=1,\ldots,n} |r_{ii}|}{\min_{k=1,\ldots,n} |r_{kk}|}$
2. $\mathrm{cond}_2(A^T A) = \mathrm{cond}_2(A)^2$

3. *Zerlegung einer reellen Matrix*

Gegeben seien $M \in \mathbb{R}^{n\times n}$ und eine Matrix $G \in \mathbb{R}^{m\times n}$, $m \leq n$, die vollen Rang besitzt. Zeigen Sie:

1. Falls $v^T M v > 0$ für alle $v \neq 0$ mit $Gv = 0$, so ist die Matrix $A = \begin{bmatrix} M & G^T \\ G & 0 \end{bmatrix}$ invertierbar.
2. Falls M symmetrisch und positiv definit ist, existiert eine Zerlegung der Form

$$\begin{bmatrix} M & G^T \\ G & 0 \end{bmatrix} = \begin{bmatrix} L & 0 \\ GL^{-T} & R^T \end{bmatrix} \begin{bmatrix} I_n & 0 \\ 0 & -I_m \end{bmatrix} \begin{bmatrix} L^T & L^{-1}G^T \\ 0 & R \end{bmatrix}$$

   Wieviele Operationen sind zur Lösung eines Gleichungssystems $Ax = b$ mit einer derartigen Matrix nötig?
   Hinweis: Cholesky-Zerlegung von M.

4. *Die Normalengleichungen sind schlecht konditioniert*

Wir betrachten die Matrix:

$$\mathbf{A} = \begin{pmatrix} 1+\varepsilon & 1 \\ 1-\varepsilon & 1 \\ \varepsilon & \varepsilon \end{pmatrix}. \tag{1}$$

In exakter Arithmetik ist die Normalengleichung:

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\underline{x} = \mathbf{A}^{\mathrm{T}}\underline{b} \tag{2}$$

äquivalent zu

$$\mathbf{B}_\alpha \begin{pmatrix} \underline{r} \\ \underline{x} \end{pmatrix} := \begin{pmatrix} -\alpha\mathbf{I} & \mathbf{A} \\ \mathbf{A}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \underline{r} \\ \underline{x} \end{pmatrix} = \begin{pmatrix} \underline{b} \\ \underline{0} \end{pmatrix}. \tag{3}$$

Schreiben Sie ein `Python`-Skript, das die Kondition von $\mathbf{A}$, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, $\mathbf{B}_1$ und $\mathbf{B}_\alpha$ mit $\alpha = \varepsilon\|\mathbf{A}\|_2/\sqrt{2}$ für $10^{-5} < \varepsilon < 1$ plottet. Das `Python`-Modul `numpy.linalg` hat eine Funktion `cond`.

Hinweis: Verwenden Sie das Template `condi.py`.

Siehe nächstes Blatt!

5. *Lineare Ausgleichsrechnung mit linearen Nebenbedingung*

Seien die Matrizen A, C und die Vektoren b, d:

$$A = \begin{bmatrix} [r]5 & -1 & -1 & 6 & 4 & 0 \\ -3 & 1 & 4 & -7 & -2 & -3 \\ 1 & 3 & -4 & 5 & 4 & 7 \\ 0 & 4 & -1 & 1 & 4 & 5 \\ 4 & 2 & 3 & 1 & 6 & -1 \\ 3 & -3 & -5 & 8 & 0 & 2 \\ 0 & -1 & -4 & 4 & -1 & 3 \\ -5 & 4 & -3 & -2 & -1 & 7 \\ 3 & 4 & -3 & 6 & 7 & 7 \end{bmatrix}, \quad b = \begin{bmatrix} [r]-4 \\ 1 \\ -2 \\ 3 \\ 3 \\ 0 \\ -1 \\ 3 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} [r]1 & 3 & -2 & 3 & 8 & 0 \\ -3 & 0 & 0 & 1 & 9 & 4 \\ -2 & 3 & -2 & 4 & 17 & 4 \end{bmatrix}, \quad d = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$$

Berechnen Sie die Lösung x_0 des lineares Ausgleichsproblems $Cx = d$. Schlagen Sie eine andere Lösung x_g von $Cx = d$ vor. Berechnen Sie die Lösung x_a des lineares Ausgleichsproblems $Ax = b$.

Berechnen Sie die Lösung x_m des lineares Ausgleichsproblems $Ax = b$ mit der linearen Nebenbedingung $Cx = d$.

Geben Sie für x in $\{x_m, x_0, x_g, x_a\}$ aus:

- x
- die Residuen $r_A(x) = |b - Ax|_2$ und $r_C(x) = |d - Cx|_2$
- die Normen $|x|_2$

Welche Eigenschaften besitzen x_m, x_0, x_g und x_a jeweils.

Hinweis: Verwenden Sie das template `lstsqLinConstr.py`.

6. *Polynomfit und die Runge-Funktion*

Die Runge-Funktion ist definiert als:

$$f(x) := \frac{1}{1 + x^2}\,. \tag{4}$$

Wir wollen diese Funktion auf dem Interval $[-5, 5]$ mit einem Polynom $P_n(x)$ von Grad n approximieren. Dazu schreiben wir ein lineares Ausgleichsproblem mit m gleichmässig in $[-5, 5]$ verteilten Punkten $\{(x_i, f(x_i))\}_{i=1}^m$ wie folgt:

$$\mathbf{A}\underline{c} = \underline{b} \tag{5}$$

wobei $\underline{c}$ die $n + 1$ Koeffizienten des Polynoms P_n sind.

a) Finden Sie die Matrix $\mathbf{A}$ und die rechte Seite $\underline{b}$.

b) Lösen Sie das lineare Ausgleichsproblem für Polynome mit Grad $2 \leq n \leq 14$ und jeweils $m = 20$ und $m = 40$.

Hinweis: Diese Aufgabe besitzt kein template.

7. ⋆ **Kernaufgabe: Global Positioning System** ⋆

Bitte wenden!

Modellierung

Global Positioning System ist ein globales Navigationssatellitensystem zur Positionsbestimmung und Zeitmessung. GPS basiert auf Satelliten, die mit codierten Radiosignalen ständig ihre aktuelle Position und die genaue Uhrzeit ausstrahlen. Aus den Signallaufzeiten können GPS-Empfänger dann ihre eigene Position und Geschwindigkeit berechnen. Theoretisch reichen dazu die Signale von drei Satelliten aus, welche sich oberhalb ihres Abschaltwinkels befinden müssen, da daraus die genaue Position und Höhe bestimmt werden kann. In der Praxis haben aber GPS-Empfänger keine Uhr, die genau genug ist, um die Laufzeiten korrekt zu messen. Deshalb wird das Signal eines vierten Satelliten benötigt, mit dem dann auch die genaue Zeit im Empfänger bestimmt werden kann. [a]
Das Signal des Satelliten i enthält seine Zeit und Position:

$$[t_{s,i},\ x_{s,i},\ y_{s,i},\ z_{s,i}]$$

Wir können N Satelliten empfangen und speichern die Messwerte zeilenweise in $\mathbf{X} \in \mathbb{R}^{N\times 4}$. Die Zeit des Empfängers t_r ist im Vergleich zur Signallaufzeit nur unzureichend bekannt und muss ebenso wie die Position bestimmt werden, d.h. wir suchen:

$$\underline{x} := [t_r,\ x_r,\ y_r,\ z_r].$$

Dazu minimieren wir die Residuen r_i, $i = 1 \dots N$ gegeben durch folgende Gleichung:

$$\boxed{r_i := -c^2(t_r - t_{s,i})^2 + (x_r - x_{s,i})^2 + (y_r - y_{s,i})^2 + (z_r - z_{s,i})^2}$$

wobei c die Lichtgeschwindigkeit im Vakuum darstellt.

[a] Aus Wikipedia: `http://de.wikipedia.org/wiki/GPS`

Aufgabenstellung

a) Implementieren Sie den Residuenvektor $F(\underline{x}, \mathbf{X}) : \mathbb{R}^4 \times \mathbb{R}^{N\times 4} \to \mathbb{R}^N$.

b) Implementieren Sie die Jacobi-Matrix $J(\underline{x}, \mathbf{X}) : \mathbb{R}^4 \times \mathbb{R}^{N\times 4} \to \mathbb{R}^{N\times 4}$.

c) Lösen Sie das nichtlineare Ausgleichsproblem mit dem Gauss-Newton Verfahren.

d) Die Genauigkeit der Positionsbestimmung hängt vom Messfehler in der Signallaufzeit, der Anzahl sichtbarer Satelliten und ihrer Verteilung ab. Die Funktion `random_sampling` im Code Template simuliert den Ausfall von Satelliten, was auftritt wenn Satelliten in einer gewissen Himmelsrichtung von einem Hindernis verdeckt werden.

Kommentieren Sie die Beobachtungen für die mit `random_sampling` erstellten Plots mit verschiedenen Toleranzen in der Zeitmessung $\varepsilon_t = 10^{-10}, 10^{-8}, 10^{-7}, 10^{-6}$.

Hinweis: Verwenden Sie das template `gps.py`