Numerik

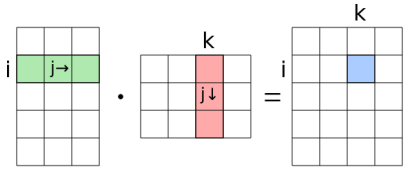
0. Grundlagen
    1. Skalarprodukt

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

Wenn $= 0$ → orthogonal

$Flops(Skalarprodukt) = n * Mult + (n-1) * Addi = 2n - 1$

    2. Matrixprodukt



$[n\ x\ m]\ *\ [m\ x\ p]\ =\ [n\ x\ p]$

$Flops(Matrixprodukt) = n * p * Flops(Skalarprodukt) = n * p * (2m - 1)$

    3. Lineare Algebra: Vektorräume, lineare Abbildungen zwischen diesen
        1. Matrizen A
            1. Spezielle Matrizen
                - Einheits~ = I
                - Diagonal~ = diag(diagonalelemente)
                - Obere Dreiecks~
                - Orthogonale ~ $O: OO^T = I$
                  → Spalten senkrecht zueinander
                  → Beste / niedrigste Konditionszahl 1
                - Mit linearen Abhängigkeiten → Zeilen lassen sich eliminieren
                - Singuläre: Quadratische Matrix mit linearen Abhängigkeiten
            2. Gauß-Verfahren zum Lösen *Linearer Gleichungssysteme (LGS)*
               $Ax = b$ *(Matrix * Vektor = Vektor)*
               → $(\ A\ |\ b\ )$

$$\left(\begin{array}{ccc|c} 2 & 1 & 3 & 1 \\ 4 & 4 & 9 & -4 \\ -2 & 5 & 3 & -1 \end{array}\right)$$

               Numerisch sinnvoll: Mit Spaltenpivotsuche
               → Betragsmäßig größtes Element der Spalte als Pivotelement
               (da Teilen durch große Zahl numerisch günstiger)
               → Führt bei Ergebnismatrix zu besserer Konditionszahl

3. Berechnung der Inversen A^{-1}
    - Für $2x2$-Matrizen

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

    - Für Diagonalmatrizen

$$\begin{pmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_n \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}_1^{-1} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_n^{-1} \end{pmatrix}.$$

    - Für Obere Dreiecksmatrizen
      Ist ebenfalls eine obere Dreiecksmatrix
    - Für Orthogonalmatrizen
      Ist die Transponierte → Ist wieder orthogonal
    - Sonst: $(A \mid I)$ mit Gauß-Verfahren zu $(I \mid A)$
4. Matrizen-Normen
    - Spektralnorm

$$||A||_2 \overset{\text{def}}{=} \sqrt{\lambda_{\max}(A^H A)}$$

    - Spaltensummennorm

$$||A||_1 \overset{\text{def}}{=} \max_{1 \le j \le n} ||a_j||_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$$

      (Maximale Spaltensumme mit Betrag!)
    - Zeilensummennorm

$$||A||_\infty \overset{\text{def}}{=} \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$$

      (Maximale Zeilensumme mit Betrag!)
    - Frobeniusnorm

$$||A||_F \overset{\text{def}}{=} \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

      (Jede Zahl der Matrix quadrieren, summieren, Wurzel)

4. Umformung in numerisch effiziente Form
    1. Substituieren
    2. Bei Substitution einer Inversen → zu LGS umformen ($* Inverse$)
    3. Schritte aufschreiben
5. Least-Squares-Algorithms
   Entwicklung einer Funktion, die möglichst nah an gegebenen Datenpunkten verläuft.
6. Themen der Numerik
    1. Konstruktion durchführbarer Algorithmen
        1. Genauigkeit/Fehler der berechneten Werte
        2. Effizienz: Komplexität/Geschwindigkeit
        3. Stabilität

7. Fehlerarten
    1. Eingabefehler
    2. Rundungsfehler
    3. Verfahrensfehler (Fehler im Algorithmus durch Annäherung)
8. Korrekt gestelltes Problem (nach Hadamard)
    1. Lösbar
    2. Lokal eindeutig
    3. Stetig (Stabilität)

9. Erstellen eines numerischen Programms (4,5 Punkte)
    1. Auswahl eines in Frage kommenden Algorithmus
    2. Ein- und Ausgabewerte festlegen
        1. Variablentypen, für Funktionen: *handle*
        2. Sinnvolle/Selbsterklärende Variablennamen wählen
        3. Toleranz *tol*
    3. Kommentare: Programmzweck, Variablenbedeutung, Quelle
    4. Definition der internen Variablen, z.B
        1. Generische Variablen (werden für das Programm benötigt)
        2. Abbruch- und Fallvariablen ε (Beispiel: Division durch Null)
    5. Eigentliches Programm
    6. Numerisches Testen des Programms auf Basis der analytischen Lösung
       Numerische Lösung sollte sich für $h \rightarrow 0$ der analytischen immer mehr nähern.
    7. Optional: Code-Optimierung
        1. Herausziehen von Operationen aus Schleifen
        2. Unnütze Funktionsaufrufe vermeiden
    8. Programm auf potentielle numerische Bugs untersuchen
        1. Division durch Null
        2. Führender Koeffizient bei Polynom wird Null
        3. Negativer Radiant
        4. ... (siehe unten)
    9. Bei der Ausführung
        1. Verwendung eines Echtzeitbetriebssystems für techn. Prozesse
           Ansonsten: Prioritätenvergabe
        2. Auf Sicherheit achten

1. Kondition und Stabilität

   $\widehat{f}$: Algorithmus

   $\widehat{x}$: Gestörte (gerundete) Eingangsdaten

   1. Kondition

      $\|f(\widehat{x}) - f(x)\|$

      „Wie stark schwankt das Problem bei Störung?"

      Empfindlichkeit eines Problems gegenüber gestörten Eingangsdaten

      Hohe Konditionszahl → Hohe Fehlerverstärkung

      1. Absolute Konditionszahl

         $\kappa_{abs} = \left\| f'(x_0) \right\|$

      2. Relative Konditionszahl

         $\kappa_{rel} = \left| \frac{\partial f(x)}{\partial x} * \frac{x}{f(x)} \right|$ (Näherung über Taylor-Reihe)

         Wenn x Vektor → „Teilverstärkungen"/Mehrere Konditionszahlen

      3. Relative Konditionszahl einer Matrix ($f(x) = Ax$)

         $\kappa_{rel}(A) = \left\| A \right\| \left\| A^{-1} \right\|$

         Abhängig von der gewählten Norm

      Beste Konditionszahl $\kappa = 1$ (für Orthogonalmatrizen in der Spektral-Norm)

      „Schelchteste" Konditionszahl $\kappa = \infty$ (für Matrix mit linearen Abhängigkeiten)

   2. Stabilität

      $\left\| f(\widehat{x}) - f(\widehat{x}) \right\|$

      „Wie stark schwankt der numerische Algorithmus bei Störung im Vergleich z. Orig.?"

      Güte des Algorithmus/Approximationsverfahrens im Vergleich zum Original

      Hohe Stabilität → Unempfindlich gegenüber Störungen

   3. Epsilon herausfinden

      $x := \varepsilon \qquad \kappa_{rel} = \left| \frac{\partial f(x)}{\partial x} * \frac{x}{f(x)} \right| = 1000$ → nach ε auflösen

2. Grundarithmetik

   1. Grundrechenarten (+ - * /), einfache Funktionsauswertungen (Wurzel...)
   2. Ersatzarithmetik → Gleitkommaarithmetik
      1. Relative Maschinengenauigkeit τ

         Kleinste Zahl, sodass gilt: $1 + \tau > 1 == true$

      2. Probleme: Underflow, Overflow
      3. Trick:

         Ein- und Ausgangsgrößen sind darstellbar, Zwischenergebnis(se) aber nicht

         → Umformulierung des Rechenwegs/Skalierung der Eingangsgrößen

3. Numerische Fallen (Kontrollfragen)
    1. Produkte
       → Über-/Unterlauf → Skalieren (Umformung des Rechenwegs)
    2. Quotienten
       → Über-/Unterlauf → Regularisierung (Epsilon-Abfrage)/l`Hospital
    3. Summen/Differenzen
       Stellenauslöschung (Gleitkommadarstellung), Schlechte Kondition
       → Anders klammern, da Addition nicht assoziativ
    4.
        1. Wurzel
           Definitionsbereich: Nur positiv mit 0 → Test auf < 0
        2. Logarithmus
           Definitionsbereich Nur positiv ohne 0
           Kleine Werte → Überlauf → Test auf $< \varepsilon$
    5. Arkusfunktionen
        1. Quadrantenbeziehung (→ Mehrfachlösung)
        2. Annahme von Grad statt Bogenmaß
        3. Überprüfung der Eingangsdaten auf Zulässigkeit:
           $if(abs(x) \leq 1\{...\}\ else\ ERROR$
    6. Tangens
       $tan\ x = \frac{sin\ x}{cos\ x}$
        - Periodische Definitionslücke (=Singularität bei $\pi/2 + k * \pi$)
        - Überlaufgefahr

→ Epsilon-Schlauch $if(abs(mod(x,\pi) - \pi/2) < \varepsilon)\ \{WARNING\}\ else\ \{\ tan\ (mod(x,\pi)\ \}$

    7.
        1. Fakultätsfunktionen
           Überlaufgefahr
        2. Binomialkoeffizienten
           → Rauskürzen der Faktoren, Geschickte Berechnung
    8. Nullvergleich
       Rechenfehler → Ergebnis nicht exakt Null
       → statt $if(x == 0)$ lieber $if(x < \varepsilon)$
    9. Polynome
        1. Aufwand (Hohe Potenzen)
        2. Stellenauslöschung durch hohe Potenzwerte
        3. Clusterung der Nullstellen
        4. Parabel wird zur Geraden durch Koeffizient ≈ 0

       → Horner-Schema
        - Weniger Aufwand
            - Normal: (2n-1) Mult. + n Add.
            - Mit Horner-Schema: n Mult. + n Add.
        - Geringerer numerischer Fehler
        - Gefahr von Überlauf/Unterlauf verringert
    10. Eigenwerte nicht über charakteristische Gleichung
        → Vermeidung von Polynomen + Nullstellensuche
        → Dreiecksfaktorisierung (Schur-Zerlegung)
        → Dreiecksmatrix mit Eigenwerten auf Hauptdiagonale

11. Gründe für die Vermeidung von Matrizenmultiplikationen
    - Verschlechterung der Konditionszahl (Linear abhg. Spalten)
    - Fehlerkumulierung
    - Hoher Aufwand
12. Bestimmen des Rangs einer Matrix (Zahl linear unabhängiger Spalten)
    Durch Zählen der Nullzeilen → Test auf 0 (siehe 8.)
    → Angabe der Kondition, Singulärwerte zum Warnen des Nutzers
13. Gründe für die Verwendung orthogonaler Matrizen zur Umformung
    - Minimale Konditionszahl ($=1$)
    - Längeninvarianz unter der euklidischen Norm
    - Elemente liegen im Intervall [-1;1] → leichtere Berechnungen
    - Spalten und Zeilen haben die Länge 1!
    - Leichte Berechnung von Normen
    - Inverse ist Transponierte
14. Ursprung schlecht konditionierter Probleme
    - Inverse Probleme (Von Ausgang auf Eingang schließen)
    - Probleme mit Entartung (Parabel zur Gerade, Kugel zu Ebene)
    - Zu hohe Modellordnung/zu viele Parameter
    - Schlechte Experimentplanung
15. Regularisierung → Verbesserung der numerischen Stabilität
    Beseitigen von Singularitäten
    z.B. → Tikhonov Regularisierung → Ersatzprobleme
16. Numerische Verbesserungen zum Lösen eines LGS
    - Numerisch sinnvoll: Mit Spaltenpivotsuche
      → größtes Element der Spalte als Pivotelement
      (da Teilen durch große Zahl numerisch günstiger)
      → Führt bei Ergebnismatrix zu besserer Konditionszahl
    - Skalierung (allgemeiner Präkonditionieren)
17. Testen von Algorithmen
    1. Handrechenbeispiel fürs Prinzip
    2. Beispiele für alle Pfade
    3. Beispiele für Gefahren (..., Numerische Probleme)
    4. Tests mit
        - Schrittweite $h \to 0$ (Verfahrensfehler min. → Analytische Lsg.)
        - Fehlerhaften Eingaben
        - Montecarlo-Methoden + Zufälligen eingaben
        - Laufzeitmessung bei Echtzeitanwendungen
          (Worst-Case-Rechenzeit → Längsten Pfad suchen)
18. Kürzen von Rechenzeit
    1. Zuweisungen machen → Zahl der Klammern/Funktionsaufrufe minimieren
    2. Compilerbetrieb (+ Interpreterbetrieb)
        - Variablen nur 1x berechnen
        - Unnötige Befehle aus Schleifen entfernen
    3. Interpreterbetrieb: Funktionsnutztung statt Schleifen

3. Komplexität
    1. Kosten eines Algorithmus: Anzahl der nötigen Schritte in der Grundarithmetik
    2. Landau-Symbolik

Symbol	$lim\ sup\ \|f(x)/g(x)\|$	Bedeutung
$f(x) = O(g(x))$	$< \infty$	f wächst höchstens so stark wie g ($\leq$)
$f(x) = o(g(x))$	$= 0$	f wächst echt schwächer als g ($<$)
Ω	> 0	Gegenteil von O
ω	$= \infty$	Gegenteil von o
$f(x) \sim g(x)$ bzw. Θ	$= 1$	f wächst proportional zu g

Bemerkungen zur Notation

Bemerkung	Formal korrekt
"=" ist kein Gleichheitszeichen	" $\in$ "
Grenzwertangabe aus dem Zusammenhang erkennbar	"für $x \to \infty$ "

4. Numerisches Differenzieren
    1. Anwendungsszenarien
        1. Funktion nur als Tabelle (nicht geschlossen) gegeben
        2. Ableitungssregeln zwecklos:
            1. Nicht geschlossen lösbar
            2. Entstehender Ausdruck zu kompliziert
        3. (Analog auch über Drehung möglich)
    2. Problem bei Ableitungen
       Störungen („Differenzieren rauht auf")
       –-(Least-Squares-Approximation)→ Polynom –(Ableiten)→ Ableitung
    3. Wichtige Formeln

	f_{k+1}	f_k	f_{k-1}	f_{k-2}	Formel für Ableitung bei $x = k$
1.1		*1*	*-1*		$\frac{1}{\Delta x} * [\_\ \_][\ ]$
1.10	1	-1			
1.11	***0,5***		***-0,5***		
2.1		*1*	*-2*	*1*	$\frac{1}{\Delta x^2} * [\_\ \_\ \_][\ ]$
2.11	***1***	***-2***	***1***		

Die Summe der Gewichte ist immer 0, weil die Ableitung einer konstanten Funktion immer 0 ergeben muss.

Symmetrische Formeln

- **verursachen Time-Lack**

- **in der Klausur benutzen!**

5. Numerische Integration (= Numerische Quadratur)
    1. Anwendungsszenarien
        1. Funktion nur als Tabelle (nicht geschlossen) gegeben
        2. Integrationsregeln zwecklos:
            1. Nicht geschlossen lösbar
            2. Entstehender Ausdruck zu kompliziert
        3. (Oberflächen-/Raumintegrale)
    2. Quadraturformel
        1. s-Tupel aus Knoten und deren Gewichten (s ist Anzahl der Stützpunkte)
        2. Anwendung:

$$Q[f] \stackrel{\text{def}}{=} h \sum_{j=1}^{s} b_j f(x_0 + c_j h) \approx \int_{x_0}^{x_0+h} f(x)\mathrm{d}x$$

        3. Ordnung
           Quadraturformel Q hat Ordnung p, wenn sie für Polynome vom Grad $< p$ (also x^{p-1}) den richtigen Wert liefert.
    3. Wichtige Quadraturformeln

Regel	Ordnung p	Q
Rechteckregel	1	$= h * f_k$
Mittelpunktregel	2	$= h * f_{k+0,5}$
Trapezregel	2	$= h/2 * (f_k + f_{k+1})$
Simpsonregel	3	$= h/6 * (f_k + 4 f_{k+0,5} + f_{k+1}$

Die Summe der Gewichte ist immer 1, weil das Integral einer konstanten Funktion c immer $h * 1c$ ergeben muss.

6. Numerisches Lösen von Differentialgleichungen
    1. Ziel: Lösungsfunktion in Form von Stützwerten angeben
        1. **Gegeben**:
            1. DGL $y' = f(x,y)$
            2. Anfang x_0
            3. Anfangswerte $y_0 = y(x_0)$
               (Bei Differentialgleichungen n-ter Ordnung: n Anfangswerte)
            4. Schrittweite
            5. (Gesuchter Punkt x_1) oder: $x_1 = x_0 + h$
        2. **Gesucht**: Funktionswert $y_1 = y(x_1)$
    2. Methoden
        1. Euler (vgl. Rechteckregel) $y_1 = y_0 + h * f(x_0, y_0)$
        2. ~~Heun (vgl. Trapezregel)~~ $y_1 = \frac{1}{2}(f(x_0, y_0) + y_{1\_Euler})$
        3. Runge-Kutta-4-Verfahren (“klassisch”)

0	0			
$1/2$	$1/2$	0		
$1/2$	0	$1/2$	0	
1	0	0	1	0
	$1/6$	$1/3$	$1/3$	$1/6$

            1. $k_1 = f(x_0, y_0)$
            2. $k_2 = f(x_0 + h/2, y_0 + h/2 * k_1)$
            3. $k_3 = f(x_0 + h/2, y_0 + h/2 * k_2)$
            4. $k_3 = f(x_0 + h, y_0 + h * k_3)$
            5. $y_1 = y_0 + h/6\ (k_1 + 2k_2 + 2k_2 + k_4)$

7. Numerisches Lösen nichtlinearer Gleichungssysteme
    1. Nichtlineares Gleichungssystem ↔ Lineare Gleichungssysteme ($Ax = b$)
    2. Problemarten
        1. Nullstellenproblem $\Phi(x) = 0$
        2. Fixpunktproblem $\Phi(x) = x$
        3. Gleichungssystemproblem $\Phi(x) = b$

        → Können leicht ineinander umgewandelt werden. Deshalb Verfahrensbetrachtung
    3. Fixpunktiteration
        1. Wichtiges Hilfsmittel: Fixpunktsatz von Banach
           Kontraktionseigenschaft:
           $sup \left\| \frac{\partial \Phi(x)}{\partial x^T} \right\| < 1$ eindimensional: $sup\,|\Phi'(x)| < 1$
           (Bei Nullstellensuche leicht erreichbar durch Vorfaktor α.) → Φ
           Dann gilt:
            1. → Existenz eines Fixpunktes
            2. → Konvergente Folge $x_{k+1} = \Phi(x_k)$ genannt Fixpunktiteration
        2. Eigenschaften von Fixpunkten: Erste Ableitung an der Stelle des Fixpunkts
            1. < 1 attraktiver
            2. > 1 repulsiv
            3. $= 1$ neutral
            4. $= 0$ superattraktiv
        3. lokale / globale Konvergenz gegen x^*
            1. Konvergenzordnung p
               Folge heißt konvergent von der Ordnung p, wenn gilt:
               $\left\| x_{k+1} - x^* \right\| \leq c * \left\| x_k - x^* \right\|^p \quad \forall k \in |N$ mit $c > 0$
            2. $c < 1$ $p = 1$ → linear
            3. $p = 2$ → quadratisch
            4. $p = 3$ → kubisch

           Bedeutung: Bei jedem Iterationsschritt ver-p-facht sich die Anzahl der genauen Dezimalstellen:
            - Fixpunktverfahren linear
            - Newton lokal quadratisch :

4. Newton-Raphson-Verfahren
   „Löse Nicht-LGS über mehrere LGS”
   1. Herleitung
      1. Erstes Taylor-Polynom (Linearisierung)
      2. Soll an der Stelle x_{k+1} eine Nullstelle haben
      3. Umformen nach x_{k+1}
   2. Benutzung
      1. In Nullstellenproblem umformen
      2. Im R^1 : 1. Ableitung ermitteln
         Im R^m : Jakobi-Matrix J_k
      3. Iteration
         Im R^1 : $x_{k+1} = x_k - \frac{f(x)}{f'(x)}$
         Im R^m : $x_{k+1} = x_k - J_k^{-1} f(x)$
         Recheneffizient: $x_{k+1} = x_k + \Delta x_k$ mit LGS $J_k \Delta x_k = -f(x_k)$
   3. Probleme
      1. Ableitung ist an der Nullstelle nicht definiert (Wurzeln)
      2. Divergiert → Schrittweitensteuerung
      3. Osziliert
      4. Andere Nullstelle wird gefunden
      5. Berechnung der Ableitung numerisch aufwendig
5. Zwei Weitere Verfahren zur Nullstellensuche
   Keine Verwendung aufgrund von Beschränkung auf R^1
   1. Bisektion/Intervallhalbierung: Binäre Suche
   2. Goldener Schnitt: „Tertiäre” Suche

8. Numerische Optimierungsverfahren
   Problem: Skalare Funktion soll minimal $Q(x) \stackrel{!}{=} \underset{x \in \mathcal{S} \subset \mathbb{R}^n}{\text{Min}}$ werden:
   1. Newton-Verfahren
      1. Herleitung
         1. Formulierung eines Quadratischen Ersatzproblems
            → Zweites Taylor-Polynom:
            $$\tilde{Q}_k(x) = Q(\bar{x}(k)) + g_k^T(x - \bar{x}_k) + \tfrac{1}{2}(x - x_k)^T H_k(x - x_k)$$
            mit Gradient und Hessematrix
         2. Minimierung des Ersatzproblems durch
            $$\frac{\partial \tilde{Q}_k}{\partial x} \stackrel{!}{=} 0_n$$ (da, wo Steigung = 0 ist Minimum von Paraboloid)
         3. Ableiten ergibt:
            $$\begin{aligned} g_k + H_k(x - x_k)|_{x=x_{\text{opt}}} &= 0_p \\ H_k(x_{\text{opt}} - x_k) &= -g_k \\ x_{\text{opt}} - x_k &= -H_k^{-1} g_k \\ x_{k+1} := x_{\text{opt}} &= x_k - H_k^{-1} g_k \end{aligned}$$
         4. Recheneffizient (Numerisch optimiert):
            - Definiere: $\Delta x = x_{k+1} - x_k$
            - LGS: $H_k \, \Delta x = -g_k \rightarrow \Delta x$
            - Lösung: $x_{k+1} = \Delta x + x_k$
         5. Abbruchbedingungen
            - Änderung der Nachkommastellen
            - Änderung des Gradienten
      2. Grafisch
         1. Startpunkt auswählen
         2. Parabel an Startpunkt anlegen
            → Tangentiale + krümmungsmäßige Übereinstimmung
         3. Tiefster Punkt der Parabel $:=$ neuer Startpunkt
         4. Weiter mit Schritt 2, bis Tiefpunkt erreicht

Allgemein:
- Antwortsätze!
- Ordentlich schreiben

Hauptsatz der Numerik nach Dr. Lutz Gröll
- Numerik liebt orthogonale Matrizen
- Numerik hasst inverse Matrizen