Numerisches und Symbolisches Rechnen - Zusammenfassung

Patrick Pletscher

2. Oktober 2004

1. Function Fitting

Gegeben: N Datenpunkte $\{x_i, y_i\}$, $i = 1 \dots N$
Gesucht: Funktion $f(x)$ welche die Punkte approximiert
Ansatz:

$$f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x) \tag{1}$$

Dabei hat die approximierte Funktion M Parameter.

1.1. Lagrange Polynome

Falls $M = N$, also die Funktion gleich viele Parameter wie es Datenpunkte gibt, haben soll.

$$l_k(x) = \frac{(x_1 - x) \dots (x_{k-1} - x)(x_{k+1} - x) \dots (x_N - x)}{(x_1 - x_k) \dots (x_{k-1} - x_k)(x_{k+1} - x_k) \dots (x_N - x_k)}$$

Für unseren Ansatz (1) setzen wir:
$\phi_k = l_k$
$\alpha_k = y_k$
Das Polynom $l_k(x)$ nimmt an jeder Stützstelle x_k genau den Wert y_k an, bei den anderen Stützstellen ist das Polynom gleich 0.
Diese Methode eignet sich nur bei Daten, von denen man weiss, dass sie *polynomiell verteilt* sind, und *kein Rauschen* enthalten.

Fehler der Lagrange Polynome:

$$|y(x) - f(x)| = |\frac{y^n(\xi)}{n!} \prod_{k-1}^{n} (x - x_k)|$$

1.2. Linear Least Squares

Falls $M < N$ verwendet man wieder den Ansatz (1).
Voraussetzung: ϕ_k ist linear in α_k (also z.Bsp. nicht: $e^{\alpha_k x}$)

$$\begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_M(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_M(x_N) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_M \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$\mathbf{Ax} = \mathbf{b}$$

Dieses System ist überbestimmt und nicht exakt lösbar.

Lösung:

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$$

Falls die Kolonnen von A *linear unabhängig* sind, so ist $\mathbf{A}^T\mathbf{A}$ invertierbar und es gilt:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

Beispiel:
Zu approximierende Funktion: $y = ax^2 + bx + c$

Punkte:

x	1	2	3	4
y	3	6	8	11

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 3 \\ 6 \\ 8 \\ 11 \end{pmatrix}$$

LLS und orthonormale Matrizen

Da für Matrizen mit orthonormalen Kolonnen gilt:

$$\mathbf{Q}^T\mathbf{Q} = I = \mathbf{QQ}^T$$

$$\mathbf{Q}^T = \mathbf{Q}^{-1}$$

Jede Matrix $\mathbf{A}$ mit linear unabhängigen Kolonnen kann faktorisiert werden in $\mathbf{A} = \mathbf{QR}$, wobei $\mathbf{Q}$ orthonormal ist und $\mathbf{R}$ eine obere Dreiecksmatrix und invertierbar ist.

Somit vereinfacht sich die LLS Lösung zu folgendem Problem, falls die QR-Zerlegung der Matrix $\mathbf{A}$ bekannt ist:

$$\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}$$

QR-Zerlegung nach Gram-Schmidt

Sei $\mathbf{A} = (\mathbf{a}_1 \dots \mathbf{a}_m)$ eine Matrix mit linear unabhängigen Kolonnen, so kann man sie wiefolgt in eine Matrix $\mathbf{QR}$ zerlegen.

$\mathbf{Q}$:

$$\left.\begin{aligned} \mathbf{q}_1 &= \tfrac{\mathbf{a}_1}{\|\mathbf{a}_1\|} \\ \widetilde{\mathbf{q}_k} &= \mathbf{a}_k - \textstyle\sum_{j=1}^{k-1} \mathbf{q}_j \langle \mathbf{q}_j, \mathbf{a}_k \rangle \\ \mathbf{q}_k &= \tfrac{\widetilde{\mathbf{q}_k}}{\|\widetilde{\mathbf{q}_k}\|} \end{aligned}\right\} k = 2, \dots, n$$

R:

$$\left.\begin{array}{l} r_{11} = \|\mathbf{a}_1\| \\ r_{jk} = \langle \mathbf{q_j}, \mathbf{a_k} \rangle, \quad j = 1, \ldots, k-1 \\ r_{jk} = 0, \quad j = k+1, \ldots, n \\ r_{kk} = \|\widetilde{\mathbf{q}_k}\| \end{array}\right\} k = 2, \ldots, n$$

1.3. Pseudoinverse und Singulärwertzerlegung (SVD)

Pseudoinverse

Gegeben: $\mathbf{Ax} = \mathbf{b}$

Falls die Kolonnen von $\mathbf{A}$ linear abhängig sind, so kann man das LLS Verfahren nicht anwenden, da somit $(\mathbf{A}^T\mathbf{A})^{-1}$ nicht berechnet werden kann.

Es gibt aber ein anderes Verfahren, das auch auf solche Matrizen anwendbar ist:

Lösung: $\mathbf{x} = \mathbf{A}^+\mathbf{b}$, wobei $\mathbf{A}^+$ die Pseudoinverse von A ist

Bestimmung der Pseudoinversen

1. $\mathbf{A}$ hat nur Einträge in der Diagonalen

$$\mathbf{A} = \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_r \\ 0 & \ldots & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & \ldots & 0 \end{pmatrix}$$

$$\mathbf{A}^+ = \begin{pmatrix} 1/\sigma_1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & 0 & \ddots & 0 \\ 0 & \ldots & 1/\sigma_r & 0 & \ldots & 0 \end{pmatrix}$$

   $\mathbf{A}^+$ wird also bestimmt in dem man die Transponierte $\mathbf{A}^T$ von $\mathbf{A}$ bestimmt und danach von allen Diagonalelementen die Inverse bestimmt, dabei wird aber $1/0 = 0$ gesetzt, falls $\sigma_i = 0$.

2. $\mathbf{A}$ allgemeine NxM Matrix
   Man zerlegt die Matrix $\mathbf{A}$ mit der Singulärwertzerlegung in eine Matrix der Form $\mathbf{A} = \mathbf{U\Sigma V}^T$, wobei

   - $\mathbf{U}$ eine MxM orthonormale Matrix ist.
   - $\mathbf{V}$ eine NxN orthonormale Matrix ist.
   - $\mathbf{\Sigma}$ eine NxM Matrix mit spezieller Diagonalform ist.

   Danach ist die Pseudoinverse wie folgt zu bestimmen:

$$\mathbf{A}^+ = \mathbf{V\Sigma}^+\mathbf{U}^T$$

   Wobei $\mathbf{\Sigma}$ die Form von Fall 1 hat und sich demnach davon leicht die Pseudoinverse bestimmen lässt.

Singulärwertzerlegung

$\mathbf{\Sigma}$:

$$\Sigma = \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}$$

wobei $\mathbf{\Sigma_r} = diag\{\sigma_1, \ldots, \sigma_r\}$, dessen Diagonalelemente positiv und alle grösser gleich geordnet sind und $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^T\mathbf{A})}$, wobei $\lambda_i(\mathbf{A}^T\mathbf{A})$ die Eigenwerte von $\mathbf{A}^T\mathbf{A}$ bezeichnet.

$\mathbf{V}$:

$$\mathbf{V} = (\mathbf{v_1}, \ldots, \mathbf{v_N})$$

wobei $\mathbf{v_i}$ die normierten Eigenvektoren von $\mathbf{A}^T\mathbf{A}$ sind.

$\mathbf{U}$:

$$\mathbf{U} = (\mathbf{u_1}, \ldots, \mathbf{u_M})$$

wobei $\mathbf{u_i}$ die normierten Eigenvektoren von $\mathbf{AA}^T$ sind.

1.4. Newton Iteration

Newton Iteration in 1D

Gesucht: Nullstellen von $f(x)$, also $f(x) = 0$ lösen.

Benötigt: $f(x)$, $f'(x)$ und Startwert x_0.

Iteratives Verfahren:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Newton für Systeme von nicht linearen Gleichungen

Gegeben: $\mathbf{f(x)}: \quad \mathbb{R}^n \to \mathbb{R}^n$
Gesucht: Schnittpunkte von Funktionen

Benötigt:

$$\mathbf{f(x)} = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix}, \qquad \mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \ldots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \ldots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Dabei bezeichnet $\mathbf{J}$ die Jaccobi-Matrix.

Iteratives Verfahren:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}^{-1}\mathbf{f}(\mathbf{x}_k)$$

Beispiel:

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + x_2^2 - 4 \\ f_2(x_1, x_2) &= x_1^2 - x_2^2 - 1 \end{aligned} \tag{2}$$

$$\mathbf{J(x)} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1 & 2x_2 \\ 2x_1 & -2x_2 \end{pmatrix}$$

$$\mathbf{x}_0 = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$$

$$\begin{aligned} \mathbf{J}(\mathbf{x}_k)\mathbf{h} &= -\mathbf{f}(\mathbf{x}_k) \\ \begin{pmatrix} 3 & 3 \\ 3 & -3 \end{pmatrix}\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} &= -\begin{pmatrix} 2.25+2.25-4 \\ 2.25-2.25-1 \end{pmatrix} \\ \mathbf{h} &= \begin{pmatrix} 0.0833 \\ -0.25 \end{pmatrix} \end{aligned}$$

und somit gilt:

$$\mathbf{x}_{k+1} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 0.083 \\ -0.25 \end{pmatrix}$$

Newton Iteration in 1D für Extremalwerte

Gesucht: $f'(x) = 0$

Benötigt: $f'(x)$, $f''(x)$ und Startwert x_0.

Iteratives Verfahren:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Newton Iteration für Extremalwerte einer Funktion mehrerer Variablen

$Q(\mathbf{x}): \qquad \mathbb{R}^n \to \mathbb{R}$

Gesucht: $\mathbf{grad}\, Q(\mathbf{x}) = \mathbf{0}$

Benötigt: Wir benötigen die Jaccobi-Matrix von $\mathbf{grad}\, Q(\mathbf{x})$, welche der Hessischen-Matrix von Q entspricht.

$$\mathbf{hess}\, Q(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 Q}{\partial^2 x_1} & \frac{\partial^2 Q}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 Q}{\partial x_1 \partial x_n} \\ \frac{\partial^2 Q}{\partial x_2 \partial x_1} & \frac{\partial^2 Q}{\partial^2 x_2} & & \vdots \\ \vdots & & \ddots & \\ \frac{\partial^2 Q}{\partial x_n \partial x_1} & \frac{\partial^2 Q}{\partial x_n \partial x_2} & & \frac{\partial^2 Q}{\partial^2 x_n} \end{pmatrix}$$

Iteratives Verfahren:

$$x_{k+1} = x_k - (\mathbf{hess}\, Q(\mathbf{x}_k))^T \mathbf{grad}\, Q(\mathbf{x}_k)$$

Bemerkung: Da $\mathbf{hess}\, Q(\mathbf{x})$ symmetrisch ist (nach dem Theorem von Schwarz), muss man $\mathbf{hess}\, Q(\mathbf{x})$ nur transponieren und nicht invertieren.

Non-Linear Least Squares

Gegeben: Überbestimmtes nicht lineares System $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m, m > n$ das man nach der Methode der kleinsten Quadrate lösen soll.
Beispiel:

$$\begin{pmatrix} x_1^2 + x_2^2 - 4 \\ x_1^2 - x_2^2 - 1 \\ x_1 + x_2 - 3 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Das Gauss Prinzip sagt: Finde $\mathbf{x}$ so, dass

$$\mathbf{Q}(\mathbf{x}) = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|_2^2 = min$$

Was gleichbedeutend ist mit:

$$\mathbf{grad}\, \mathbf{Q}(\mathbf{x}) = \mathbf{0}$$

Diese Gleichung ausgedrückt in der Funktion $\mathbf{f}(\mathbf{x})$

$$\mathbf{Q}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i(\mathbf{x})^2$$

Das kann kompakt wie folgt geschrieben werden:

$$\mathbf{grad}\, \mathbf{Q}(\mathbf{x}) = \mathbf{J}_f(\mathbf{x})^T \mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Für das *Beispiel* sieht die Gleichung dann wie folgt aus:

$$\begin{pmatrix} 2x_1 & 2x_2 & 1 \\ 2x_1 & -2x_2 & 1 \end{pmatrix}\begin{pmatrix} x_1^2 + x_2^2 - 4 \\ x_1^2 - x_2^2 - 1 \\ x_1 + x_2 - 3 \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Um dieses System zu lösen gibt es nun mehrere Wege:

- **Gauss-Newton Algorithmus**

  $$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}_f(\mathbf{x}_k)^T \mathbf{J}_f(\mathbf{x}_k))^{-1} \mathbf{J}_f(\mathbf{x}_k)^T \mathbf{f}(\mathbf{x}_k)$$

  Der Term $(\mathbf{J}_f(\mathbf{x}_k)^T \mathbf{J}_f(\mathbf{x}_k))^{-1} \mathbf{J}_f(\mathbf{x}_k)^T$ kommt davon, dass $\mathbf{J}_f$ nicht quadratisch ist und wir davon nicht die Inverse bestimmen können sondern nur mit LLS eine Annäherung an $\mathbf{J}_f^{-1}$. Für die Iteration können wir aber in MATLAB den Backslash-Operator benutzen:

  1. Löse
     $$\mathbf{J}_f(\mathbf{x}_k)\mathbf{h} = -\mathbf{f}(\mathbf{x}_k)$$
     durch
     $$\mathbf{h} = \mathbf{J}_f(\mathbf{x}_k) \backslash -\mathbf{f}(\mathbf{x}_k)$$
  2. Iteriere
     $$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}$$

  Dieses Verfahren hat eine lineare Konvergenz.

- **Newton Methode für non-linear least squares**

  1. Löse
     $$\mathbf{B} := \mathbf{J}_f^T(\mathbf{x}_k)\, \mathbf{J}_f(\mathbf{x}_k) + \sum_{l=1}^{m} f_l(\mathbf{x}_k) \mathbf{hess}\, f_l(\mathbf{x}_k)$$
     $$\mathbf{B}\, \mathbf{h} = -\mathbf{J}_f(\mathbf{x}_k)^T \mathbf{f}(\mathbf{x}_k)$$
     $$\mathbf{h} = -\mathbf{B}^{-1}\, \mathbf{J}_f(\mathbf{x}_k)^T \mathbf{f}(\mathbf{x}_k)$$
  2. Iteriere
     $$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}$$

  Dieses Verfahren konvergiert quadratisch.

- **Steepest descent**
  Die Ausdrücke $\mathbf{B}$ in Newton und $\mathbf{J}_f(\mathbf{x}_k)^T\mathbf{J}_f(\mathbf{x}_k)$ in Gauss-Newton sind ziemlich kompliziert bzw. können teilweise nicht invertiert werden, so dass man stattdessen oft Steepest descent $\lambda\mathbf{I}$ benutzt. Dabei muss λ aber so gewählt sein, dass das Ganze konvergiert. Die Iteration sieht dann wie folgt aus:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda\mathbf{J}_f(\mathbf{x}_k)^T\mathbf{f}(\mathbf{x}_k)$$

- **Levenberg-Marquart**

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}_f^T\mathbf{J}_f + \lambda\mathbf{D})^{-1}\mathbf{J}_f^T\mathbf{f}(\mathbf{x}_k)$$

  wobei $\mathbf{D}$ diagonal ist, oftmals ist $\mathbf{D} = \mathbf{I}$
  Praktisch wird $\mathbf{h}$ wie folgt berechnet:

$$\begin{pmatrix} \mathbf{J}_f \\ \sqrt{\lambda}\sqrt{\mathbf{D}} \end{pmatrix}\mathbf{h} \approx \begin{pmatrix} -\mathbf{f}(\mathbf{x}_k) \\ \mathbf{0} \end{pmatrix}$$

2. Direkte Suchmethoden

Während es bei den analytischen Methoden darum geht, das *Optimum in einem Schritt* zu erreichen, ohne Test, geht es bei den Direkten Methoden darum, dass die Lösung *Schritt um Schritt (iterativ)* angenähert wird, und bei jedem Schritt der Wert der betrachteten Funktion verbessert wird, wenn das nicht der Fall ist, so wird mit trial and error vorgegangen.

2.1. Analytische Methoden

Notwenige Bedingung: Minimiere den Gradienten - System von Gleichungen.
Hinreichende Bedingung:

- in 1D
  Wenn die *zweite Ableitung positiv* ist, so ist es ein *Minimum*, falls *negativ* ein *Maximum*, und falls *Null* ein *Sattelpunkt*.

- in N-Dimensionen
  Die Determinate der Hessischen Matrix muss positiv sein; für ein Minimum muss auch die N-1 Subdeterminate positiv sein, sonst ist es ein Maximum.
  Falls die Determinate der Hessischen Matrix negativ ist, so ist es ein Sattelpunkt.

Die analytischen Methoden können aber auch schief gehen

- *Diskontinuität* von der betrachteten Funktion und ihrer Ableitungen.

- *Differentiation kann unmöglich sein* (z.Bsp. bei Experimenten oder Black-Box Code) oder ungenau (z.Bsp. Daten mit Rauschen).

- Optimum kann ein *Extremwert oder ein Sattelpunkt* sein.

- System von Gleichungen (v.a. nicht linear) kann möglicherweise nicht lösbar sein oder *sehr teuer* zu lösen.

2.2. Nicht-Gradienten basierende Methoden

Direkte Suchmethoden

Wir benutzen den Ausdruck "Direkte Suche" um die *sequenzielle Untersuchung* von Versuchen zu beschreiben. Wobei jeder Versuch mit der "besten" bisherigen Lösung *verglichen* wird, zusammen mit einer *Strategie* (als einer Funktion der bisherigen Resultate), welches der nächste Versuch sein wird.
Dabei werden im Normallfall keine analytischen Betrachtungen gemacht, ausser es gibt einen grossen Vorteil.

Evolutionary Operation Method (EVOP)

Die Schlüsselfunktion davon ist, dass sie nicht numerische Funktionswerte besitzen muss: die *relative Rangierung* von Objekten ist hinreichend.

EVOP Algorithmus

1. Funktion an Stellen $+d$ und $-d$ in jeder Dimension auswerten. (Am Anfang ist z.Bsp. $d = 1$)

2. Neuer Punkt = Punkt mit minimalem Funktionswert

3. Falls der gleiche Punkt selektiert wird, so wird d skaliert (z.Bsp. verkleinert)

4. Iteriere bis ein ε erreicht ist, oder bis zu einer maximalen Anzahl Iterationen

Parallele Methoden

1. Untersuche die Funktionswerte an verschiedenen Punkten

2. Deklariere den Punkt mit dem kleinsten Funktionenwert zum Minimum

- Diese Methoden werden auch als *Gitter Methoden* oder *tabellarische Methoden* bezeichnet.

- *Unglaublich langsam* - Anzahl der Versuche ist umgekehrt proportional zur Genauigkeit, aber sie sind *parallel ausführbar*.

Sequenzielle Methoden

In sequenziellen Methoden:

- Versuche werden *sequenziell* gemacht.

- Zwischenresultate werden benutzt um den nächsten Punkt zu lokalisieren.

Sie können klassifiziert werden in:

- Suchmustermethoden

- Simplexmethoden

- Methoden mit lernfähigen Mengen von Suchrichtungen

Sind langsam, proportional zum Logarithmus der Genauigkeit

Hooke und Jeeves: Exploratory Method

Eine Extrapolation entlang einer Linie von der ersten und letzten Bewegung, bevor die Variablen wieder individuell verändert werden.

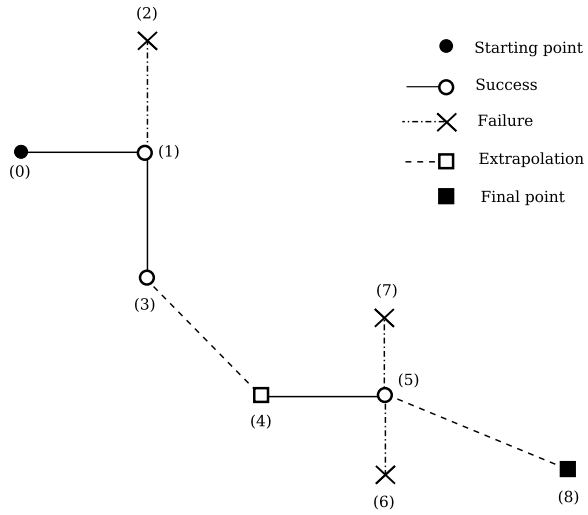


Abbildung 1: *Exploratory Method nach Hooke und Jeeves*

Nummer	Iterat. index k	Richtungs-index i	Variablen-wert x1/x2	Vergl. punkt	Schritt-länge s1/s2	Bemerkungen
(0)	0	0	0 / 9	-	2 / 2	Startpunkt
(1)	0	1	2 / 9	(0)		success
(2)	0	2	2 / 11	(1)		failure
(3)	0	2	2 / 7	(1)		success
(4)	1	0	4 / 5	-	2 / -2	extrapolation
(5)	1	1	6 / 5	(4),(3)		success, success
(6)	1	2	6 / 3	(5)		failure
(7)	1	2	6 / 7	(5)		failure
(8)	2	0	10 / 3	-,(5)	2 / -2	extrapolation

Pattern Search Methods

Verändere theoretische Parameter zu einer Zeit bei Schritten der *gleichen Grösse* und wenn kein vergrössern oder verkleinern eines Parameters das Resultat verbessert, so *halbiert man die Schrittgrösse.*

Pattern search methods sind durch eine Serie von untersuchenden Bewegungen charakterisiert, welche das Verhalten der betrachteten Funktion *in einem Muster von Punkten*, welche alle auf einem *rationalen* Gitter liegen.

Die Schlüsselfunktion ist, dass die Punkte auf einem Gitter bleiben.

Konvergenz:

Theorem 1 (Polak) *Wenn $\{X_k\}$ eine Sequenz ist, welche von einer pattern search Methode erstellt wurde, so befriedigt jeder Ansammlungspunkt, dass*

$$\nabla f(x^*) = 0$$

Die Methode kann nur eine endliche Anzahl von Zwischenpunkten erzeugen, bevor sie die Schrittgrösse halbiert, somit kann der Algorithmus nicht bei einem Punkt stecken bleiben.

Simplexmethode

Ein Simplex ist eine Menge von $n+1$ Punkten in $\mathbb{R}^n$: Im 2D ein Dreieck, in 3D ein Tetraeder usw.
Simplex Algorithmus

1. Ein initialaler Simplex wird erstellt.
2. Ermittle den Knoten mit dem schlechtesten Fit.
3. Spiegle den schlechtesten Knoten an der Mitte der gegenüberliegenden Kante.
4. Der Punkt mit dem besten Resultat wird der Mittelpunkt des nächsten Zyklus.
5. Iteriere bis zur maximalen Genauigkeit oder Anzahl maximaler Iterationen.

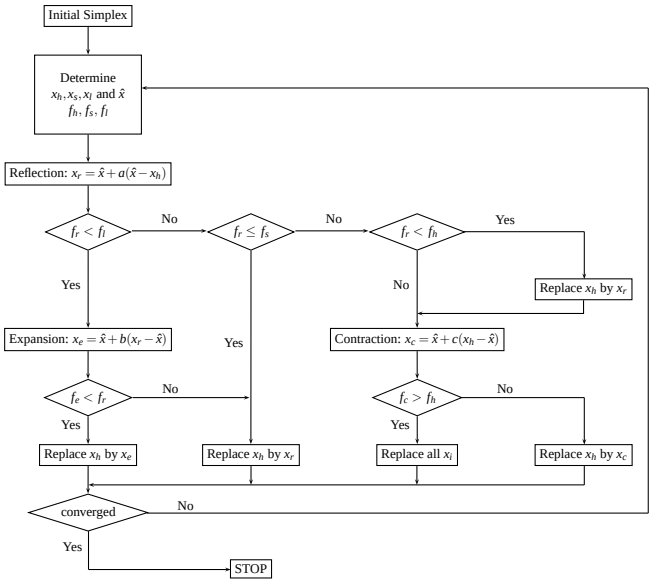


Abbildung 2: *Simplex Algorithmus Flowchart*

Für die Ersetzung aller x_i benutzt man:

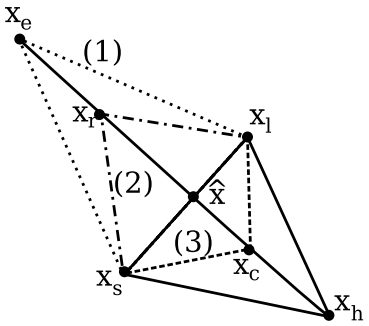
$$x_i = x_i + c(x_l - x_i)$$

Dabei ist:

$\mathbf{x_h}$	schlechtester Punkt, d.h. $f(\mathbf{x_h})$ ist grösser als die Funktion ausgewertet bei allen anderen $\mathbf{x_i}$
$\mathbf{x_s}$	zweit-schlechtester Punkt
$\mathbf{x_l}$	bester Punkt
$\hat{\mathbf{x}} = \frac{1}{n-1} \sum_{i=1}^{n} \mathbf{x_i}\ (i \neq h)$	Mittelwert ohne schlechtesten Punkt

Konvergenz Kriterium:

$$\varepsilon > \frac{\sum_{i=1}^{n} \sqrt{(\mathbf{x_i} - \hat{\mathbf{x}})^T (\mathbf{x_i} - \hat{\mathbf{x}})}}{n}$$



(1) Expansion
(2) Reflection
(3) Contraction

Abbildung 3: *Simplex Algorithmus grafisch*

3. Fouriertransformation

3.1. Fourierreihe

Funktion f soll approximiert werden durch eine Summe von N periodischen Funktionen mit je zwei Parametern:

$$f(x) \approx \sum_{k=0}^{N} a_k \cos(kx) + b_k \sin(kx)$$

wobei die Koeffizienten folgendermassen berechnet werden:

$$\begin{aligned} a_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx \qquad k \neq 0 \\ b_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dx \\ a_0 &= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)dx \end{aligned}$$

b_0 braucht man nicht zu betrachten, da immer gilt $b_0 \sin(0) = 0$.

Eigenschaften der Fourierreihen

Fehler:

$$E = \int_{-\pi}^{\pi}\left[f(x) - \sum_{k=0}^{N} A_k\cos(kx) + B_k\sin(kx)\right]^2 dx$$

Dieser Fehler wird minimiert, wenn $A_k = a_k, B_k = b_k$

im Komplexen

$$e^{ikx} = \cos(kx) + i\sin(kx)$$

$$i^2 = -1$$

Somit ergibt sich für die Fourierreihe:

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx} \qquad \& \qquad c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}dx$$

Wobei gilt:

$$\begin{aligned} c_k &= \frac{1}{2}a_k - \frac{1}{2}ib_k \qquad k \geq 0 \\ c_{-k} &= \bar{c_k} = \frac{1}{2}a_k + \frac{1}{2}ib_k \\ a_k &= c_k + c_{-k} \\ b_k &= -\frac{1}{i}(c_k - c_{-k}) \end{aligned}$$

Weitere Eigenschaften:

$$\begin{aligned} \frac{df}{dx} &= ik\sum_{-\infty}^{\infty} c_k e^{ikx} = ikf \\ \frac{d^2f}{dx^2} &= -k^2 f \\ \left(\frac{d^2}{dx^2} + \frac{d^2}{dy^2}\right)e^{i(k_1x+k_2y)} &= -(k_1^2+k_2^2)e^{i(k_1x+k_2y)} \end{aligned}$$

Analytische Fouriertransformation

Die Fouriertransformierte $\hat{f}$ einer Funktion $f(x)$ ist nun wie folgt definiert:

$$\hat{f}(k) = \int_{-\infty}^{\infty} f(x)e^{-ikx}dx$$

Die Fouriertransformierte einer Funktion $f(x)$ wird im Nachfolgenden mit $\mathcal{F}(f)$ oder $\hat{f}(x)$

3.2. Diskrete Fouriertransformation (DFT)

Gegeben ein Datenvektor $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ der Länge n, die DFT ist definiert als:

Fourier Transformation in Matrixschreibweise:

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(n-1)} \\ 1 & w^3 & w^6 & \ldots & w^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)^2} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

$$\mathbf{Fy} = \mathbf{x}$$

Wobei das (i,j)-Element von $\mathbf{F}$ wie folgt aussieht:

$$f_{ij} = w^{(i-1)(j-1)}$$

Wobei w für jedes n definiert ist als:

$$w = e^{i\frac{2\pi}{n}} = e^{\frac{2\pi i}{n}} \qquad \& \qquad \bar{w} = e^{-\frac{2\pi i}{n}}$$

$$w^n = e^{2\pi i} = 1$$

Die DFT eines Datenvektors $\mathbf{x}$ ist nun:

$$\mathcal{F}(\mathbf{x}) = \mathbf{y} = \frac{1}{\sqrt{n}}\bar{\mathbf{F}}\mathbf{x}$$

Dabei bezeichnet $\bar{\mathbf{F}}$ die Matrix $\mathbf{F}$, wobei aber überall w durch $\bar{w}$ ersetzt wird.

und die Rücktransformation:

$$\mathcal{F}^{-1}(\mathbf{y}) = \mathbf{x} = \frac{1}{\sqrt{n}}\mathbf{Fy}$$

3.3. Fast Fourier Transformation

Mit diesem Algorithmus kann die DFT effizient durchgeführt werden, nämlich in $O(n \log n)$. Gegeben ist also wieder ein Datenvektor $\mathbf{x} = (x_0, \ldots, x_{n-1})$ der Länge n. Dieser Algorithmus wird rekursiv angewendet, bis der Datenvektor $\mathbf{x}$ nur noch Länge 1 hat, von diesem ist die Fouriertransformierte $\mathbf{y}$ dann leicht zu bestimmen, da $\mathbf{x} = \mathbf{y}$ falls die Länge von $\mathbf{x}$ gleich 1 ist.

Fast Fourier Transformations Algorithmus

1. Vektor $\mathbf{x}$ aufteilen in 2 Vektoren $\mathbf{x}', \mathbf{x}''$ mit geraden/ungeraden Indices

$$\mathbf{x}' = (x_0, x_2, x_4, \ldots, x_{n-2})$$

$$\mathbf{x}'' = (x_1, x_3, x_5, \ldots, x_{n-1})$$

2. Fouriertransformierte $\mathbf{y}' = \bar{\mathbf{F}}\mathbf{x}'$ und $\mathbf{y}'' = \bar{\mathbf{F}}\mathbf{x}''$ bestimmen (rekursiv mit FFT).

3. Die beiden Vektoren $\mathbf{y}'$ und $\mathbf{y}''$ wie folgt zusammensetzen:

$$y_j = y'_j + \overline{w}_n^j y''_j \qquad \text{für } j = 0, \ldots, \frac{n}{2} - 1$$

$$y_{j+\frac{n}{2}} = y'_j - \overline{w}_n^j y''_j \qquad \text{für } j = 0, \ldots, \frac{n}{2} - 1$$

Für die Rücktransformation muss w_n an Stelle von $\overline{w}_n$ gewählt werden und $\mathbf{F}$ statt $\bar{\mathbf{F}}$.

4. Faltung

Die Faltung zweier Funktionen f und g, ist definiert als:

$$f * g = \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi$$

Dadurch werden die Punkte von $f(x)$ nach $g(x)$ gewichtet.

Eigenschaften:

$$\begin{aligned} \mathcal{F}(f * g) &= \mathcal{F}(f) \cdot \mathcal{F}(g) \\ f * (g + h) &= (f * g) + (f * h) \\ \frac{d}{dx}(f * g) &= \frac{df}{dx} * g = f * \frac{dg}{dx} \end{aligned}$$

4.1. Diskrete Faltung

Die Funktionen sind durch Punkte $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1})$ und $\mathbf{g} = (g_0, g_1, \ldots, g_{n-1})$ gegeben. Die diskrete Faltung berechnet sich wie folgt:

$$f * g = \begin{pmatrix} f_0 g_0 + f_1 g_{n-1} + f_2 g_{n-2} + \ldots + f_{n-1} g_1 \\ f_0 g_1 + f_1 g_0 + f_2 g_{n-1} + \ldots + f_{n-1} g_2 \\ \vdots \\ f_0 g_{n-1} + f_1 g_{n-2} + f_2 g_{n-3} + \ldots + f_{n-1} g_0 \end{pmatrix}$$

Bemerkungen:

- Das Faltungsprodukt von zwei Vektoren der Länge n hat Länge n.
- Jede Komponenete hat n Terme
- Kosten: $O(n^2)$
- Erste Komponente enthält alle Produkte $f_j g_k$ mit $j+k = 0$ oder $j + k = n$
- l-te Komponente enthalten alle Produkte mit $j + k = l$ oder $j + k = l + n$

Faltungsregel

Die Faltung von f und g ist eine gewöhnliche Multiplikation der Fouriertransformierten Funktionen.

$f * g$ wird berechnet:

1. $c = \mathcal{F}^{-1}(f)$
2. $d = \mathcal{F}^{-1}(g)$
3. Multipliziere c und d Komponente für Komponente
4. $(f * g) = \sqrt{n} \cdot \mathcal{F}(cd)$

4.2. δ Funktion (Dirac Funktion)

Definition

Wir möchten eine Erregung $d_a(x)$, die über einem kleinen Intervall $a-\varepsilon < x < a+\varepsilon$ nicht-Null ist, und sonst überall Null ist, beschreiben. Der gesamte Impuls davon ist dann definiert als:

$$I = \int_{-\infty}^{\infty} d_a(x) dx = \int_{a-\epsilon}^{a+\epsilon} d_a(x) dx \qquad (\varepsilon > 0)$$

Um ein mathematisches Modell der Funktion $d_a(x)$ zu geben, ist es bequem, anzunehmen, dass sie einen konstanten Wert über dem geschlossenen Intervall $[a-\varepsilon, a+\varepsilon]$ annimmt, auch möchten wir diesen Wert so wählen, dass der Gesamtimpuls gleich 1 ist. Also schreiben wir:

$$d_a(x) = \begin{cases} \frac{1}{2\varepsilon} & a - \varepsilon \leq x \leq a + \varepsilon \\ 0 & \text{sonst} \end{cases}$$

Nun betrachten wir eine Idealisierung der Funktion $d_a(x)$, indem wir ε gegen Null gehen lassen:

$$\lim_{\varepsilon \to 0} I = \lim_{\varepsilon \to 0} \int_{-\infty}^{\infty} d_a(x) dx = 1$$

Damit können wir eine idealisierte Ein-Puls Funktion $\delta(x-a)$ definieren, welche die Eigenschaft hat, dass sie einen Impuls von 1 bei der Stelle $x = a$ hat, aber für alle anderen Werte von x Null ist. Die *Definitionseigenschaften* sind deshalb:

$$\begin{aligned} \delta(x - a) &= 0 \qquad x \neq a \\ \int_{-\infty}^{\infty} \delta(x - a) dx &= 1 \end{aligned}$$

oder für $a = 0$:

$$\begin{aligned} \delta(x) &= 0 \qquad x \neq 0 \\ \int_{-\infty}^{\infty} \delta(x)dx &= 1 \end{aligned}$$

Im weiteren wird mit a gerechnet, wobei man aber natürlich für den Fall $a = 0$ a weglassen kann (wie in der Vorlesung).

Weitere Eigenschaften

$$\begin{aligned} \int_{-\infty}^{\infty} f(x)\delta(x-a)dx &= f(a) \\ f(x) &= \int_{-\infty}^{\infty} f(y)\delta(y-x)dy \end{aligned}$$

Die δ Funktion ermöglicht es uns eine diskrete Funktion (z.B. Datenpunkte) als kontinuierliche Funktion zu schreiben.

Beispiel: Datenpunkte $\{(x_1, f_1), \ldots, (x_n, f_n)\}$ können dann als Funktion so geschrieben werden:

$$f(x) = \sum_{i=1}^{n} f_i \delta(x - x_i)$$

4.3. Wichtige Fouriertransformationen und Faltungen

1.

$$\begin{aligned} f(x) &= \delta(x) \\ \hat{f}(k) &= \int_{-\infty}^{\infty} e^{-ikx}\delta(x)dx = 1 \end{aligned}$$

2.

$$f(x) = \text{ square pulse } = \begin{cases} 1 & -L \leq x \leq L \\ 0 & |x| > L \end{cases}$$

$$\hat{f}(k) = \int_{-L}^{L} e^{-ikx}dx = \frac{e^{-ikL} - e^{ikL}}{-ik} = \frac{2\sin(kL)}{k}$$

3.

$$f(x) = \begin{cases} e^{-ax} & x > 0 \\ 0 & x < 0 \end{cases}$$

$$\hat{f}(k) = \frac{1}{a + ik} \qquad a > 0$$

4.

$$f(x) = e^{-a|x|} \qquad a > 0$$

$$\hat{f}(k) = \frac{2a}{a^2 + k^2}$$

$$\begin{aligned} &f(x) \ldots\ldots\ldots\ldots\ldots\ldots\ldots \hat{f}(k) \\ &\tfrac{d}{dx}f(x) \ldots\ldots\ldots\ldots\ldots\ldots ik\hat{f}(k) \\ &f(x-d) \ldots\ldots\ldots\ldots\ldots e^{-ikd}\hat{f}(k) \\ &e^{ixd}f(x) \ldots\ldots\ldots\ldots\ldots \hat{f}(k-d) \end{aligned}$$

$$\hat{\delta} = 1$$

$$\hat{f} = \hat{f} \cdot 1 = \hat{f} \cdot \hat{\delta}$$

$$f(x) = f(x) * \delta(x) = \int_{-\infty}^{\infty} f(y)\delta(x-y)dy$$

4.4. Faltung und Differentialrechnung

Gegeben: Differentialgleichung der Form

$$Lu(x) = h(x)$$

Wobei L ein Differentialoperator ist, (z.Bsp. $L = -\frac{d}{dx^2} + a^2$ oder $L = \frac{d}{dx}$).

1. Forme die ganze Gleichung in den Fourierraum um, dabei ersetzt man am besten die rechte Seite durch $h(x)$, bzw. nach dem Transformieren durch $\hat{h}(k)$ und benutzt die Regel für das Transformieren einer Ableitung:

$$\mathcal{F}\left(\frac{df}{dx}\right) = ik\mathcal{F}(f)$$

2. Man hat nun einen Ausdruck der Form

$$\hat{u}(k) = \hat{f}(k) \cdot \hat{g}(k)$$

   Dabei ist $\hat{g}(k)$ die sogenannte "Green's function".

3. Berechne nun dieses Produkt im Fourierraum, somit erhält man $\hat{u}(k)$.

4. Transformiere $\hat{u}(k)$ zurück in den Realspace.

Beispiel:

$$\begin{aligned} \mathcal{F}\left(-\frac{d^2u}{dx^2} + a^2u(x)\right) &= \mathcal{F}(h(x)) \\ -(ik)^2\hat{u}(k) + a^2\hat{u}(k) &= \hat{h}(k) \\ k^2\hat{u}(k) + a^2\hat{u}(k) &= \hat{h}(k) \\ \hat{u}(k) &= \frac{\hat{h}(k)}{a^2 + k^2} = \hat{h}(k) \cdot \hat{g}(k) \end{aligned}$$

Jetzt müsste man $\hat{u}(k)$ noch zurücktransformieren.

5. Principal Component Analysis

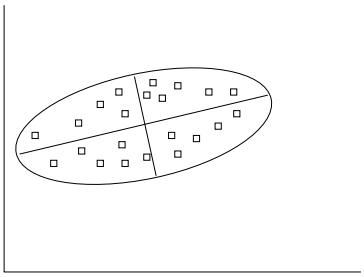


Abbildung 4: *Steigung der Halbachsen beschreibt die Punkte*

5.1. Statistische Ausdrücke

Mittelwert:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Varianz:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Standardabweichung:

$$s = \sqrt{s^2}$$

Kovarianz:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Für mehrdimensionale Daten erstellen wir die Kovarianz-Matrix:

$$Cov(x,y,z) = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Rechenregeln

1. $var(X+Y) = var(X) + var(Y)$ wenn X, Y unabh.
2. $var(N(0,1)) = 1$
3. $cov(X,Y) = 0$ wenn X, Y unabh.
4. $cov(X,X) = var(X)$
5. $cov(X,Y) = cov(Y,X)$
6. $cov(aX,Y) = a \cdot cov(X,Y)$
7. $cov(X+Y,Z) = cov(X,Z) + cov(Y,Z)$
8. $cov(X+Y,Z+W) = cov(X,Z)+cov(X,W)+cov(Y,Z)+cov(Y,W)$

5.2. Principal Components Analysis

Ziel ist es Daten (mit Verlust) zu komprimieren, ohne aber wesentliche Daten zu verlieren.

$\mathbf{x}$ soll eine Menge von Messvektoren sein (also eigentlich eine Matrix) und $\mathbf{y} = \mathbf{M}\mathbf{x}$ eine Transformation vom Vektor $\mathbf{x}$ zu einem neuen Vektor $\mathbf{y}$ mit gewünschten Eigenschaften (welche definiert werden müssen).

Die Kovarianzmatrix von $\mathbf{y}$ ist definiert durch:

$$\mathbf{C}_y = \langle (\mathbf{y} - \langle \mathbf{y} \rangle) \cdot (\mathbf{y} - \langle \mathbf{y} \rangle)^T \rangle$$

Dabei ist $\langle\rangle$ der ensemble average, also der Mittelwert der Vektoren bzw. Matrizen.

Durch Umformen erhält man:

$$\mathbf{C}_y = \mathbf{M}\mathbf{C}_x\mathbf{M}^T$$

Nun möchten wir die Transformation $\mathbf{M}$ so wählen, dass $\mathbf{C}_y$ diagonal ist und die Elemente unkorreliert sind:

$\mathbf{C}_x$ ist symmetrisch und hat eine orthonormale Menge von Eigenvektoren $q_1, \ldots, q_n$. Wir setzen die Spalten von $\mathbf{M}^T$ gleich diese Eigenvektoren.

$$\mathbf{C}_x\mathbf{M}^T = C_x \cdot (q_1, \ldots, q_n) = (\lambda_1 q_1, \ldots, \lambda_n q_n)$$

$$\mathbf{M}\mathbf{C}_x\mathbf{M}^T = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} = \mathbf{C}_y$$

Falls es Korrelationen zwischen den Elementen von $\mathbf{x}$ gibt, so werden einige Eigenwerte verschwinden, diese Komponenten von $\mathbf{y}$ können dann für weitere Betrachtungen weggelassen werden.

Für reale Daten werden die korrelierten Eigenwerte nicht genau Null sein, aber man kann durch die relative Grösse die wichtigen herauslesen und die unwichtigen weglassen. Solches selektieren von Variablenuntermengen ist oftmals der Schlüssel zu erfolgreichem Modellieren.

Algorithmus

1. Daten in Matrizen oder Vektoren konvertieren. Bei Matrizen ist es bequemer, wenn man sie in Vektoren umformt und am Ende wieder zurück. Mehrere solcher Messvektoren (hier n Messungen) der Dimension m werden in einer Matrix $\mathbf{D}$ (hier spaltenweise) angeordnet.
2. Den Mittelwertvektor **avg** bilden:

$$avg_i = \frac{1}{n-1}\sum_{j=0}^{n} d_{ij} \qquad \text{für } j = 0, \ldots, m-1$$

3. Von jedem Messvektor den normalisierten Mittelwertvektor subtrahieren. Dies ergibt uns die normalisierte Matrix $\mathbf{D}_{norm}$
4. Kovarianzmatrix von $\mathbf{D}_{norm}$ bilden

$$\mathbf{C} = \frac{1}{n-1}\mathbf{D}_{norm} \cdot \mathbf{D}_{norm}^T$$

5. Eigenwerte $\mathbf{E}$ und -vektoren $\mathbf{V}$ von $\mathbf{C}$ bestimmen
6. Bestimmte Eigenvektoren $\mathbf{v}$ wählen, andere nicht, z.B. die Eigenvektoren wo die Eigenwerte ungleich Null sind. Durch die gewählten Eigenvektoren ergibt sich eine neue Matrix $\mathbf{Q}$.

Um nun einen Datenvektor $\mathbf{x}$ zu komprimieren benutzt man

$$\mathbf{y} = \mathbf{Q}^T\mathbf{x}$$

Für die Rücktransformation gilt

$$\mathbf{x}' = \mathbf{Q}\mathbf{y}$$

6. Nicht lineare Gleichungen

6.1. Bisektion

Gegeben: $f(x)$ stetig in $[a,b]$, wobei $f(a) < 0$ und $f(b) > 0$. Daraus folgt: $\exists s : f(s) = 0,\ s \in [a,b]$.

Gesucht: Eine Nullstelle von $f(x)$.

Bisektionsalgorithmus

1. $x := (a+b)/2$ /* x = Mitte von Intervall*/
2. Wenn $f(x) > 0, b := x$; sonst $a := x$; /* Intervall neu */
3. Wiederhole Algorithmus, falls $b - a > \varepsilon$, also die gewünschte Genauigkeit noch nicht erreicht wurde

Konvergenz

Der Bisketionsalgorithmus konvergiert linear, da der Fehler in jedem Schritt maximal $\varepsilon < (b-a)\frac{1}{2}$ beträgt.

6.2. Iterative Methoden

Gegeben: Funktion $f(x)$
Gesucht: Nullstelle von $f(x)$

Statt für $f(x) = 0$ das passende x zu suchen, sucht man eine Funktion F und einen Wert s, so dass $F(s) = s$ ist. (Fixpunkt). Dann ist $f(s) = 0$.

Fixpunktiteration

Man startet mit einem Startwert x_0 und iteriert bis die vorgegebene Toleranz erreicht ist. (= Unterschied zwischen zwei x-Werten $< \varepsilon$).

$$x_{k+1} = F(x_k)$$

es gelten:

$$F(s) = s \qquad f(s) = 0$$

Konvergenz

$$\frac{e_{k+1}}{e_k} \approx F'(s)$$

e_k: Fehler bei k-ter Iteration $= x_k - s$

Konvergenzkriterium

$$\begin{aligned} |e_{k+1}| &< |e_k| \\ |F'(s)| &< 1 \end{aligned}$$

Konvergenzgeschwindigkeit

ausgehend von $|\frac{e_{k+1}}{e_k}| \approx |F'(s)|$:

$$|e_k| \approx |F'(s)|^k \cdot |e_0| = \varepsilon \qquad e_0 : \text{ Anfangsfehler}$$

$$\text{Anzahl Schritte } k = \frac{\log_{10}(\frac{\varepsilon}{|e_0|})}{\log_{10}(|F'(s)|)}$$

Um zu Testen wieviele Schritte, bis eine Stelle: $\varepsilon = 0.1, e_0 = 1$.

$$e_{k+1} = F'(s)e_k + \frac{F''(s)}{2}e_k^2 + \frac{F'''(s)}{6}e_k^3 + o(e_k)$$

Wobei $o(e_k)$ Terme höherer Ordnung sind.

Konvergenzraten

linear :	$F'(s) \neq 0,\ \|F'(s)\| < 1$
quadratisch :	$F'(s) = 0,\ \|F''(s)\| < 1$
kubisch :	$F'(s) = F''(s) = 0,\ \|F'''(s)\| < 1$

Ein-Punkt Iterationsmethoden mit hoher Konvergenzrate

Schlüsselidee: Ersetze $f(x) = 0$ durch $h(x) = 0$. Dabei wird $h(x)$ so gewählt, dass $h(x) = 0$ einfach analytisch gelöst werden kann.

1. *Newton-Raphson Iteration*
   Wähle $h(x) = f(x_k) + f'(x_k)(x - x_k)$ (Taylorentwicklung von $f(x_{k+1})$ um $f(x_k)$)

   $$h(x) = 0 \Rightarrow x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

   Konvergiert quadratisch
2. *Halley Iteration*

   $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\left[1 - \frac{1}{2}\frac{f(x_k)f''(x_k)}{f'(x_k)^2}\right]^{-1}$$

   Konvergiert kubisch

7. Zellularautomaten

NxM Zellen, in einem Rechteck angeordnet:

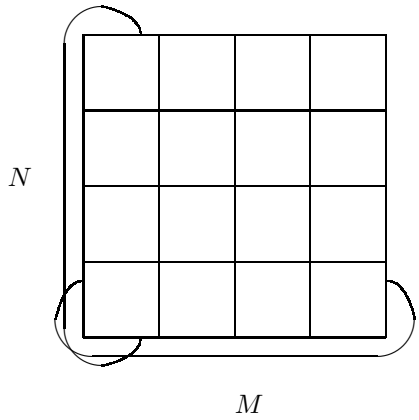


Abbildung 5: *NxM Zellularautomat*

- Zu einem endlosen Band zusammengefügt
- Jede Zelle hat als Wert entweder 0 oder 1

Algorithmus:
Ein Algorithmus gibt Kriterien an, bei welchen eine Zelle ihren Wert ändert, abhängig von ihren acht Nachbarzellen.

8. Numerische Differentiation

8.1. Konstruktion von Ableitungsformeln durch Taylorexpansion

h bezeichnet im Folgenden $\Delta x = x_{i-1} - x_i$

Vorwärtsdifferenzieren

$$f(x_i+h) = f(x_{i+1}) = f(x_i)+hf'(x_i)+\frac{h^2}{2}f''(x_i)+\frac{h^3}{6}f'''(x_i)+o(h^4)$$

daraus folgt:

$$\frac{f(x_{i+1})-f(x_i)}{h} = f'(x_i) + \underbrace{\frac{h}{2}f''(x_i)+\frac{h^2}{6}f'''(x_i)+o(h^3)}_{\text{leading error / truncation error}}$$

Rückwärtsdifferenzieren

$$f(x_i-h) = f(x_i)-hf'(x_i)+\frac{h^2}{2}f''(x_i)-\frac{h^3}{6}f'''(x_i)+o(h^4)$$

daraus folgt:

$$f'(x_i) = \frac{f(x_i)-f(x_{i-1})}{h}+\frac{h}{2}f'(x_i)-\frac{h^2}{6}f'''(x_i)+o(h^3)$$

Grad der Genauigkeit

Der Exponent von $o(h^\alpha)$ ist der Grad der Genauigkeit der Methode/Formel, also der erste Term in h, welcher unnötigerweise addiert/subtrahiert wird, die obigen Methoden, haben also beide erste Ordnung.

Höhergradige Approximationen

Erhält man durch die Kombination von mehreren Taylorexpansions.

$$\begin{array}{rcl} f_{i+1} & = & f_i + hf_i' + \frac{h^2}{2}f_i'' + \frac{h^3}{6}f_i''' + \frac{h^4}{24}f_i'''' + o(h^5) \\ f_{i-1} & = & f_i - hf_i' + \frac{h^2}{2}f_i'' - \frac{h^3}{6}f_i''' + \frac{h^4}{24}f_i'''' + o(h^5) \\ \hline f_i & = & \frac{f_{i+1}-f_{i-1}}{2h} + \frac{h^2}{3}f_i''' + o(h^5) \end{array}$$

Wobei wir die beiden Taylorexpansions subtrahierten. Die Approximation ist nun von zweitem Grad.

Im Allgemeinen werden höhere Grade erreicht indem man mehr Punkte in die Approximation miteinbezieht.

$$f_i' = \frac{f_{i-2}-8f_{i-1}+8f_{i+1}-f_{i+2}}{12h}+o(h^4)$$

Aber Schemas für höhere Grade haben Probleme an den Grenzen, z.Bsp:

$$f_1' = \frac{f_{-1}-8f_0+8f_2-f_3}{12h}+o(h^4)$$

Lösungen:

1. Benutze Approximationen von tieferem Grad bei den Grenzen
2. Benutze einseitige Approximationen

8.2. Generelle Technik für die Konstruktion von Finite differences Formeln

Idee

Kombiniere Taylorexpansionsserien mit Gewichten a_i um eine kompakte Schablone für eine maximale Ordnung zu erhalten.

Taylor Tabelle

Ausgehend von:

$$\begin{array}{rcl} f_{i+1} & = & f_i + hf_i' + \frac{h^2}{2}f_i'' + \dots \\ f_{i+2} & = & f_i + 2hf_i' + \frac{(2h)^2}{2}f_i'' + \dots \end{array}$$

	f_i	f_i'	f_i''
f_i'	0	1	0
$a_0 f_i$	a_0	0	0
$a_1 f_{i+1}$	a_1	$a_1 h$	$a_1\frac{h^2}{2}$
$a_2 f_{i+2}$	a_2	$a_2(2h)$	$a_2\frac{(2h)^2}{2}$

Wir wollen nun f_i' durch f_i, f_{i+1} und f_{i+2} approximieren.

$$f_i' \approx \sum_{k=0}^{2} a_k f_{i+k} \qquad \rightarrow \qquad f_i' - \sum_{k=0}^{2} a_k f_{i+k} = 0 \qquad (3)$$

Es gilt nach Taylorapproximation

$$\begin{array}{rcl} a_0f_i + a_1f_{i+1} + a_2f_{i+2} & = & (a_0+a_1+a_2)f_i + \\ & & +(a_1h + a_2(2h))f_i' + \\ & & + \left(a_1\frac{h^2}{2} + a_2\frac{(2h)^2}{2}\right)f_i'' \quad (4) \end{array}$$

Nun setzen wir die rechte Seite von (4) in (3) ein, somit gilt die Gleichung:

$$f_i' - f_i(a_0+a_1+a_2) - f_i'(1+a_1h+a_2 2h) - f_i''\left(a_1\frac{h^2}{2}+a_2\frac{(2h)^2}{2}\right) = 0$$

Somit kommt man auf folgendes Gleichungssystem:

$$\begin{array}{rcl} a_0 + a_1 + a_2 & = & 0 \\ a_1h + a_2(2h) & = & 1 \\ a_1\frac{h^2}{2} + a_2\frac{(2h)^2}{2} & = & 0 \end{array}$$

Ergibt als Lösungen:

$$\begin{array}{rcl} a_0 & = & \frac{-3}{2h} \\ a_1 & = & \frac{4}{2h} \\ a_2 & = & \frac{-1}{2h} \end{array}$$

Und somit gilt:

$$f_i' \approx \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h}$$

9. Numerische Integration

Da wir die Funktion nur an diskreten Punkten x_i kennen, unterteilen wir das Integral in eine Summe von Integralen:

$$\begin{aligned} I_i &= \int_{x_i}^{x_{i+1}} f(x)dx \\ I &= \sum_{i=0}^{n-1} I_i \end{aligned}$$

9.1. Trapezregel

$$I_i = hf(x_{i+1}) + \frac{h}{2}[f(x_i) - f(x_{i+1})] = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

Für das Integral von f_0 bis f_n ergibt sich:

$$I = \sum_{i=0}^{n} I_i = h\left(\frac{1}{2}f_0 + \frac{1}{2}f_n + \sum_{i=1}^{n-1} f_i\right)$$

Fehler

Trapezregel ist 3. Ordnung $[o(h^3)]$ für ein Intervall.

Für das ganze Intervall ist die Trapezregel 2. Ordnung.

Trapezregel mit Endkorrekturen

Integral von dem Intervall $[a, b]$:

$$I = \frac{h}{2}\sum_{i=0}^{n-1}(f_i + f_{i+1}) - \frac{h^2}{12}(f'(b) - f'(a)) + o(h^4)$$

Also 4.ter Ordnung.

9.2. Simpsonregel

Wir approximieren die Funktion in dem wir eine Parabel $f(x) = Ax^2 + Bx + C$ benutzen (dies braucht 3 Punkte A, B, C).

$$I_i = \int_{x_i}^{x_{i+2}} f(x)dx \approx \frac{h}{3}[f(x_i) + 4f(x_{i+1}) + f(x_{i+2})]$$

$$I = \frac{h}{3}(f(x_0) + f(x_n) + 4\sum_{i=1,3,5,\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,\ldots}^{n-2} f(x_i)$$

Bemerkung: Funktioniert nur, falls (n+1) ungerade ist.

9.3. Rechtecksregel

Benutze Punkt y_i zwischen x_i und x_{i+1}:

$$y_i = \frac{1}{2}[x_{i+1} + x_i]$$

Dann gilt:

$$I_i = \int_{x_i}^{x_{i+1}} f(x)dx = hf(y_i)$$

Um $f(y_i)$ zu erhalten benutzen wird Taylorexpansion, wobei $h = y_i - x_i$:

$$f(y_i) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + o(h^4)$$

Fehler

Rechtecksregel ist 3. Ordnung $[o(h^3)]$ für ein Intervall.

10. Lösen von Differentialgleichungen

Probleme:

- Genauigkeit (des numerischen Schemas)
- Stabilität (des numerischen Schemas)
- Konsistenz (zwischen numersichem & exaktem)

Für die Konsistenz beim Lösen von Differentialgleichungen wird Genauigkeit und Stabilität benötigt.

10.1. Genauigkeit von finite differences Approximationen

Modified wavenumber / Leapfrog

Betrachte eine pure harmonische Funktion der Periode L hier wird vereinfacht mit $L = 1$ gearbeitet. Setze

$$f(x) = e^{ikx}$$

k: wavenumber welche nur die folgenden Werte annehmen kann: $k = 2\pi n$, $n = 0, 1, 2, \ldots, \frac{N}{2}$
Die Ableitung dieser Funktion:

$$\frac{df}{dx} = f' = ikf$$

Uns interessiert nun wie ein bestimmtes finite differences Schema diese Ableitung voraussagt:

$$\frac{df}{dx} = ik'f$$

Wobei k' ein für diese FD typische Proportionalität ist, welche wir nun mit dem exakten k vergleichen wollen. k' ist keine Ableitung!

Beispiel:
Uns interessiert die Genauigkeit folgender FD:

$$\left.\frac{df}{dx}\right|_j = \frac{f_{j+1} - f_{j-1}}{2h}$$

Substituiere nun für $f(x) = e^{ikx}$ & $f(x_j) = e^{ikx_j}$ wobei man für $x_j = \frac{L}{N} = h \cdot j$, dabei ist $j = 0, \ldots, N-1$ benutzt. Nach einigem Umformen bekommt man dann, dass für $k'h$ gilt:

$$k'h = \sin\left(\frac{2\pi n}{N}\right)$$

Dies vergleicht man nun mit der genauen Lösung

$$kh = \frac{2\pi n}{N}$$

10.2. Numerische Stabilität

Numerische Lösung muss begrenzt sein für Probleme wo die exakte Lösung begrenzt ist.
Beispiel:

$$\frac{dy}{dt} = -c^2 y \qquad \Rightarrow \qquad y(t) = e^{-c^2 t}$$

Modell

$$\frac{dy}{dt} = \lambda y$$

λ kann nun aber komplex sein:

$$\lambda = \lambda_R + \lambda_I$$

Für $\lambda_R < 0$ ist das Problem beschränkt, da

$$\frac{dy}{dt} = -|\lambda| y \qquad \Rightarrow \qquad y(t) = e^{-|\lambda| t}$$

Uns interessiert nun für welche λ_R, λ_I ein bestimmtes FD begrenzt ist.

A. Beispiel von Newton Iteration für NLLS

Gegeben die Funktion

$$f(t) = a_0 + a_1 \cdot e^{-bt}$$

Wir definieren die Fehlerfunktion

$$f_k(\mathbf{x}) = x_1 + x_2 \cdot e^{-x_3 t_k} - y_k$$

für jeden der gegebenen Datenpunkte $\{t_k, y_k\}$. Für f_k können wir nun den Gradienten und die Hessische berechnen.

$$\begin{aligned} \text{grad} f_k(\mathbf{x}) &= [1, e^{-x_3 t_k}, -x_2 t_k^{-x_3 t_k}]^T \\ \text{hess} f_k(\mathbf{x}) &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_k e^{-x_3 t_k} \\ 0 & 0 & x_2 t_k^2 e^{-x_3 t_k} \end{bmatrix} \end{aligned}$$

In MATLAB:

```
d = load('ChemData.dat');
ti = d(:,1); yi = d(:,2);
x = [1.8 1.8 0.1]'; fs = [];
for i=1:20
  ee = exp(-x(3)*ti);
  tee = ti.*ee;
  J = [ones(size(ti)) ee -x(2)*tee];
  f = x(1) + x(2)*ee - yi;
  s1 = f'*tee;
  s2 = x(2)*f'*(ti.*tee);
  H = J'*J + [0 0 0; 0 0 -s1; 0 -s1 s2];
  h = H \ (J'*f);
  x = x - h;
  fs = [fs norm(f)];
end
```

B. Beispiel für Principal Component Analysis

```
function Q = get_pc(D, k)

% m is the number of variables,
% n is the number of samples
[m, n] = size(D);

% get the averages for each variable
avgs = sum(D') ./ n;

% create the 'normalized' matrix d
d = D;
for i=1:n
  d(:,i) = d(:,i) - avgs';
end

% create the covariance matrix C
C = d*d' ./ (n-1);

% get the eigenvalue/eigenvector
% decomposition of C
[v, l] = eig(C);

% return the k columns with the
% highest eigenvalues
Q = v(:,m-k+1:m)';
```

C. Hilfestellungen für Fouriertransformation

Potenzen von i

$$\begin{array}{rclcrcl} i^2 &=& -1 & & (-i)^2 &=& -1 \\ i^3 &=& -i & & (-i)^3 &=& i \\ i^4 &=& 1 & & (-i)^4 &=& 1 \end{array}$$

$\mathbf{F}$ und $\bar{\mathbf{F}}$ für $n = 2$:

$$\bar{\mathbf{F}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$