6.4. Eigenwerte und Vektoriteration

Eigenvektor $v \neq 0$ und Eigenwert λ einer quadratischen Matrix A erfüllen die Gleichung

$$A\,v = \lambda\,v$$

Daher ist die durch den Vektor v bestimmte Gerade durch den Nullpunkt eine sog. Fixgerade der durch

$$x \rightarrow A\,x$$

definierten Abbildung.

Für eine symmetrische Matrix A gilt:

Die Eigenvektoren der n Eigenwerte von A bilden eine Orthonormalbasis des R^n.

Eigenvektormatrix V liefert eine Diagonalisierung der Matrix A.

Die durch A definierte Abbildung wird in dieser Basis trivial:

$$A\,v = \lambda\,v \quad \rightarrow \quad A\,V = V\,\Lambda,$$

Λ ist Diagonalmatrix, mit den Eigenwerten als Diagonaleinträge

$$V^T A V = \Lambda = \mathrm{diag}(\lambda_1 , \ldots , \lambda_n)$$

Ein Eigenwert erfüllt die Gleichung $\det(A - \lambda I) = 0$,
da $A - \lambda I$ singulär: $(A - \lambda I)\, v = 0$

Also sind Eigenwerte genau die Nullstellen des *charakteristischen Polynoms* $p(\lambda) := \det(A - \lambda I)$.

Eigenwerte von A ←→ Nullstellen eines Polynoms p(x)

Die Matrix $A := \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \ddots & \vdots & -a_1 \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & 1 & 0 & -a_{n-2} \\ 0 & \cdots & 0 & 1 & -a_{n-1} \end{pmatrix}$

hat das charakteristisches Polynom

$$p(x) = (-1)^n \left(a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + x^n \right)$$

Daher lassen sich Nullstellen von Polynomen berechnen, indem man die Eigenwerte der obigen Matrix A berechnet!

Eigenwertberechnung ist numerisch stabiler als Nullstellenberechnung bei Polynomen!

Vektoriteration

Vektoriteration ist eine einfache Fixpunktiteration zur Berechnung des betragsgrößten Eigenwerts einer Matrix:

Sei A symmetrisch positiv definit ($x \neq 0 \rightarrow x^T Ax > 0$);

Start mit $x_0 \neq 0$, $x_{k+1} := \frac{Ax_k}{\|Ax_k\|}$;

Die Iterationsfunktion: $\Phi(x) = Ax / \|Ax\|$

hat Fixpunkt v, wenn $v = Av / \|Av\|$

d.h. v ist Eigenvektor von A zu Eigenwert $\lambda = \|Av\|$,
denn dann gilt : $A v = \|Av\| * v$.

Offensichtlich ist dann $x_k = \text{const} * A^k x_0$

Außerdem kann x_0 in der Basis der Eigenvektoren dargestellt werden:

$$x_0 = c_1 v_1 + \ldots + c_m v_m$$

wobei λ_1 maximaler Eigenwert von A zu Eigenvektor v_1 ist.

Daher gilt $Av_j = \lambda_j v_j$ und

$$A^k x_0 = c_1 A^k v_1 + \ldots + c_m A^k v_m =$$

$$= c_1 \lambda_1^k v_1 + \ldots + c_m \lambda_m^k v_m =$$

$$= \lambda_1^k \left(c_1 v_1 + \ldots + c_m \underbrace{(\lambda_m / \lambda_1)^k}_{\to 0} v_m \right) \overset{k \to \infty}{\to} \lambda_1^k \left(c_1 v_1 \right)$$

Normiert man,dann folgt $$\frac{A^k x_0}{\left\| A^k x_0 \right\|} \to \frac{\lambda_1^k \left(c_1 v_1 \right)}{\left\| \lambda_1^k \left(c_1 v_1 \right) \right\|} = \frac{c_1}{|c_1|} v_1 = \pm v_1$$

x_k ist normierter Vektor zu $A^k x_0$.
Dann bleibt in x_k nur die Komponente zum stärksten Eigenwert übrig, nämlich v_1 .

Also $x_k \rightarrow$ const $\cdot v_1$ konvergiert gegen Eigenvektor,
und daher auch

$\| Ax_k \| \rightarrow \lambda_1$ konvergiert gegen größten Eigenwert.

Ähnliches gilt für allgemeine Matrizen, solange die Matrix einen eindeutigen betragsgrößten Eigenwert hat

(also z.B. nicht : $\pm\lambda_1$ sind beide Eigenwerte)

Beispiel:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1/2 \end{pmatrix}^k \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2^{-k} \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

ist Eigenvektor zu Eigenwert $\lambda = 1$

Beispiel:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}^k \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ (-1)^k \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ \pm 1 \end{pmatrix}$$

Eigenvektoren zu Eigenwerten $\lambda = 1$ und -1

Für innere Eigenwerte:

Wende Vektoriteration auf die Matrix $(A-\sigma I)^{-1}$ an.

Liefert deren größten Eigenwert.

Dies ist der Eigenwert von A, der am nächsten bei σ liegt.

Dies ist die sog. Inverse Iteration.

Problem: In jedem Schritt ist ein Gleichungssystem zu lösen
$(A-\sigma I)$ schlecht konditioniert, ev. singulär!

Anwendungen

Resonanzen, siehe Tacoma, London Millenium Bridge

Energieniveaus in der Quantenmechanik

Nullstellen von Polynomen

Biegen eines Balkens - Hauptträgheitsachsen

Stabilität eines Systems, Differentialgleichungen

Modellreduktion

Analyse von Graphen, Pagerank

Unterschiedliche Aufgabestellung:

Alle Eigenwerte/vektoren oder nur einige!

Eigenwerte alleine oder mit Eigenvektoren

QR-Verfahren

Gesucht sind alle Eigenwerte/vektoren!

1. Schritt: transformiere A auf Tridiagonalform (obere Hessenberg) ohne die Eigenwerte zu verändern?

Am besten orthogonale Basistransformation mir orthonormalem Q:

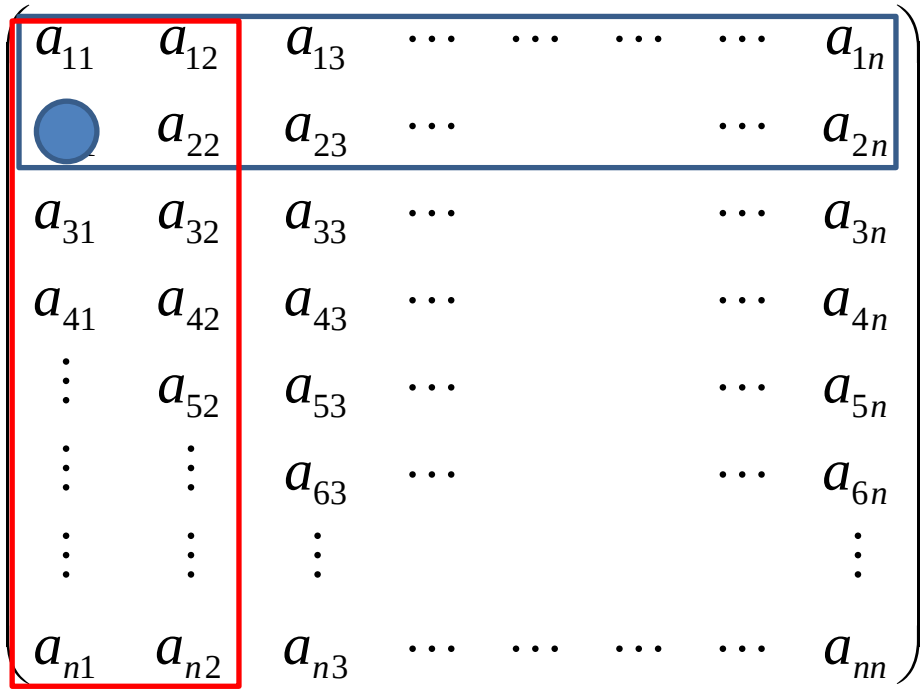
$$A_{neu} = Q*A*Q^T$$

$$Ax = \lambda x \Leftrightarrow AQ^T(Qx) = \lambda Q^T(Qx) \Leftrightarrow$$

$$\Leftrightarrow A_{neu}y = QAQ^T y = \lambda y, \qquad y = Qx$$

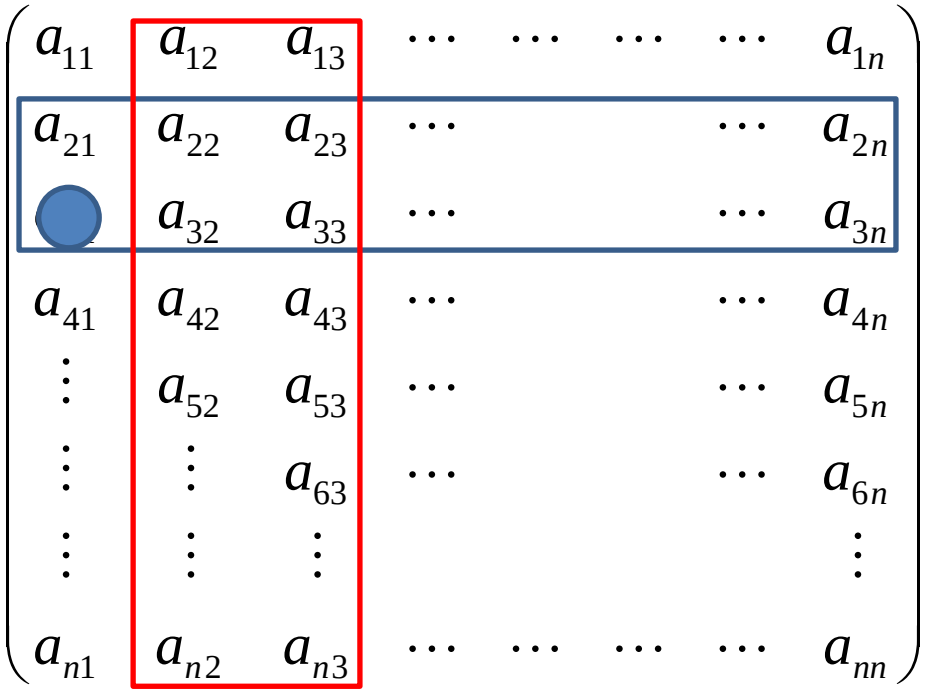
Matrix wie $Q*A*Q^T$, Eigenwert gleich, Eigenvektor wie Qx

Welches Q? Givens?



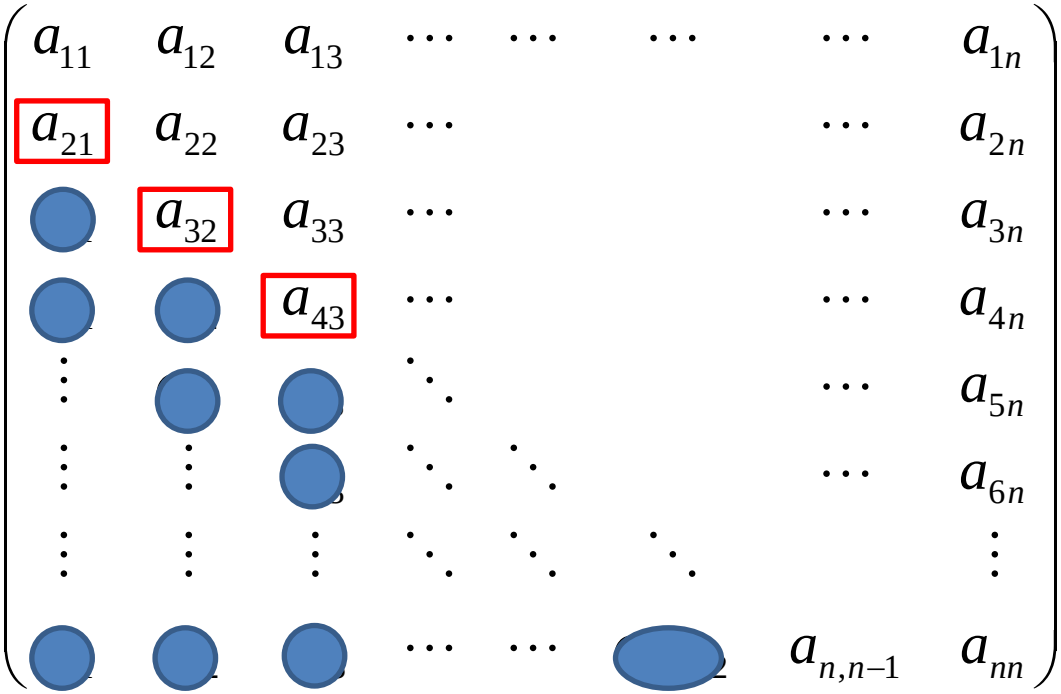
Von links Givens zur Elimination von a_{21}?
Anwendung von rechts füllt a_{21} wieder auf!

Besser:



Von links Givens zur Elimination von a_{31}?
Anwendung von rechts verändert a_{21} nicht!

Insgesamt:



Führt im Endeffekt auf Tridiagonalform für symmetrisches A, bzw. obere Hessenbergform für allgemeines A.

QR-Verfahren:

Berechne zu A die QR-Faktorisierung A=QR und setze

$$A_{neu} = RQ$$

Daher gilt $R=Q^TA$.

Damit ist $A_{neu} = Q^TAQ$ mit denselben Eigenwerten!

Wiederhole iterativ.

Im Endeffekt konvergiert A gegen Diagonalmatrix mit den Eigenwerten auf der Diagonalen.

Einige zusätzliche Tricks dabei!

Rayleigh Quotient

$$R(x) = \frac{x^T A x}{x^T x}$$ ist für symmetrisches A eine Funktion

R: $\mathbb{R}^n \rightarrow \mathbb{R}$

Das Maximum von R(x) ist der größte Eigenwert, das Minimum der kleinste.

R(x) ist der Range/Wertebereich der Matrix A.

6.5. Anwendungsbeispiele

6.5.1. Logistische Parabel:

Erinnerung, Iterationsfunktion: $\Phi(x) = \alpha x(1-x)$

Folge der Iterierten (x_0, x_1, x_2, x_3,...) bezeichnet man als Orbit von x_0 bzgl. Φ. Start für $0 < x_0 < 1$.

Der Orbit kann unterschiedliches Verhalten zeigen:
- Konvergenz gegen einen Häufungspunkt
 oder
- es existieren mehrere Häufungspunkte, zwischen denen die Folge hin und her springt, sog. Attraktoren (0.9, -0.9, 0.99, -0.99, 0.999, -0.999, ...) → ± 1

Dann zerfällt der Orbit in einzelne, konvergente Teilfolgen.

Für $1 < \alpha < 2$, hatten wir monoton konvergenten Orbit,
für $2 < \alpha < 3$ alternierend konvergenten Orbit.

Für $\alpha >\approx 3$ ergeben sich Orbits mit mehreren Häufungspunkten.
Der Fixpunkt von Φ ist dann nicht mehr anziehend,
sondern abstoßend!

Erinnerung: Fixpunkt und Ableitung am Fixpunkt sind

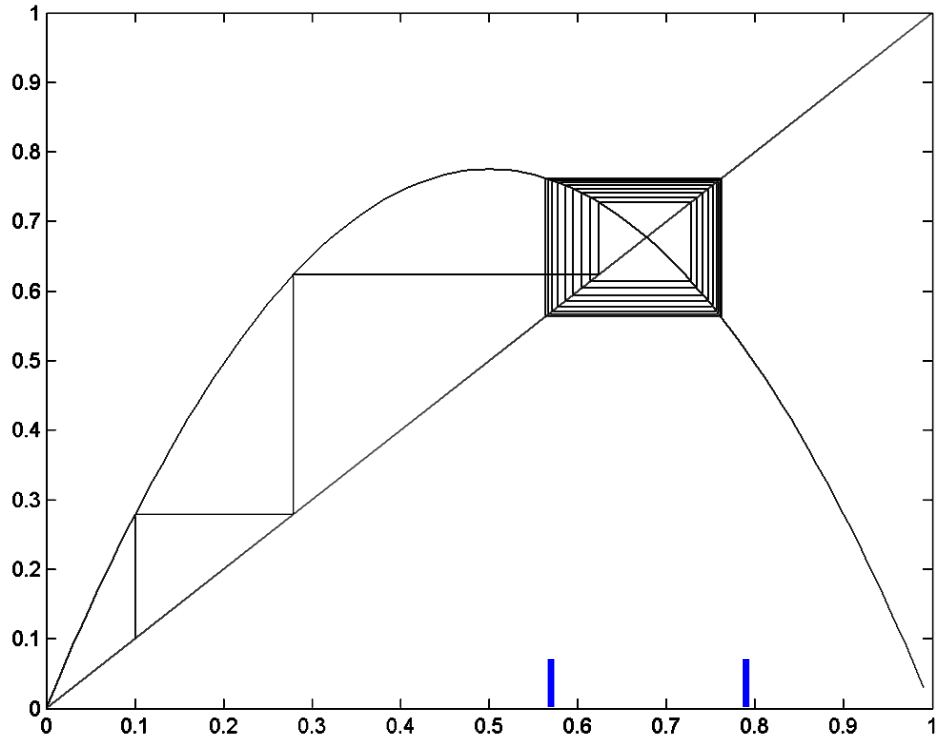
$$\bar{x} = \frac{\alpha - 1}{\alpha}; \qquad \Phi'(\bar{x}) = 2 - \alpha;$$

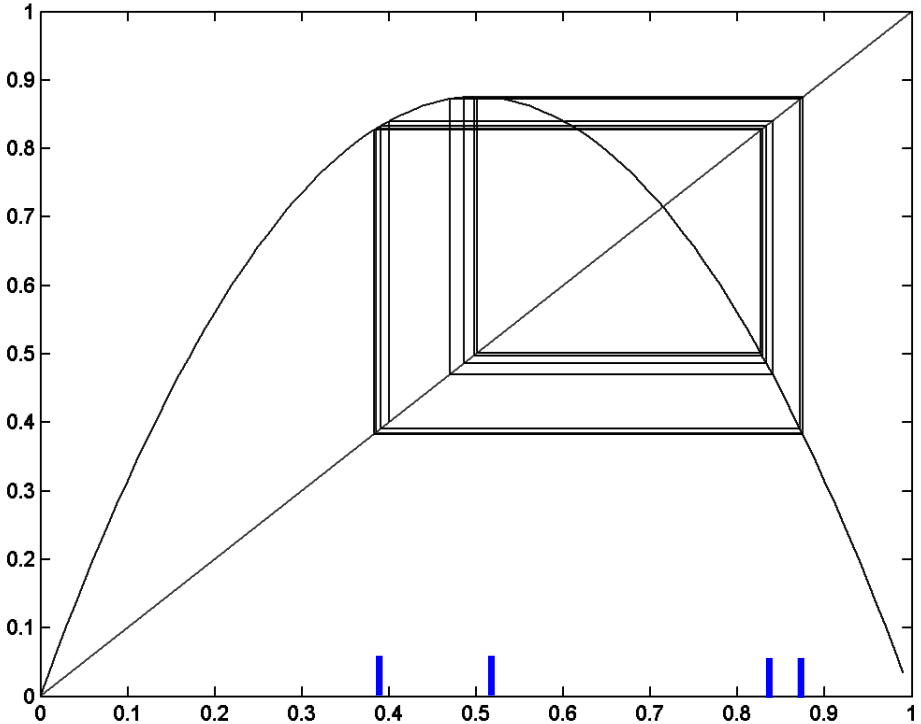
Für $\alpha = 3.1$ erhalten wir zwei Attraktoren, die nun anziehende Fixpunkte der neuen Iterationsfunktion

$$\Phi(\Phi(x)) = \alpha^2 x(1-x)(1-\alpha x+\alpha x^2)$$

sind.

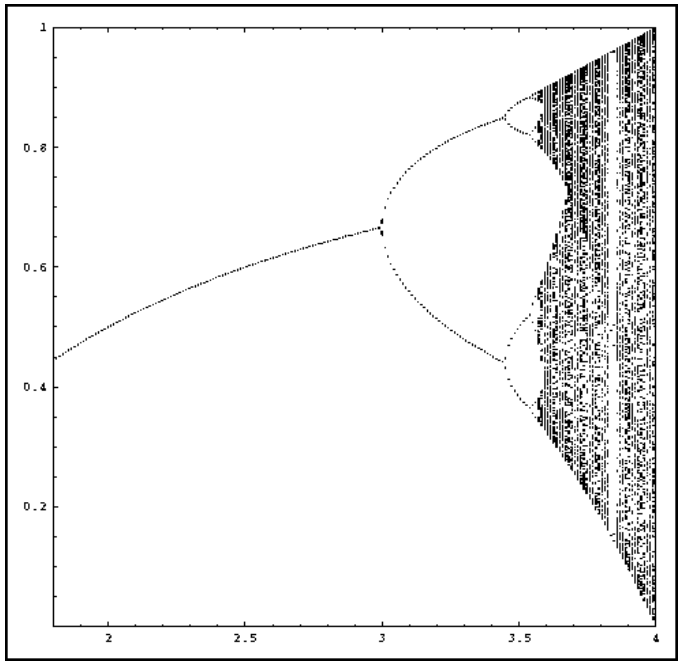
Konvergente Teilfolgen x_{2k} und x_{2k+1} .





Erhöht man α , so treten immer mehr Attraktoren auf, und für $\alpha <\approx 4$ wird das Verhalten chaotisch.

(**MATLAB: fixpunkt.m)**

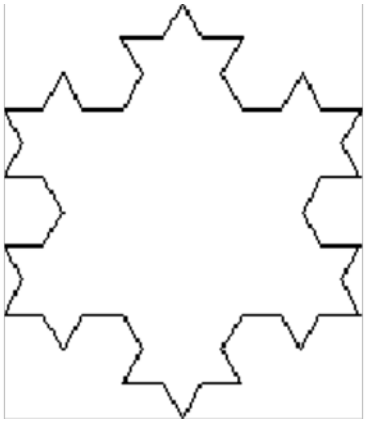


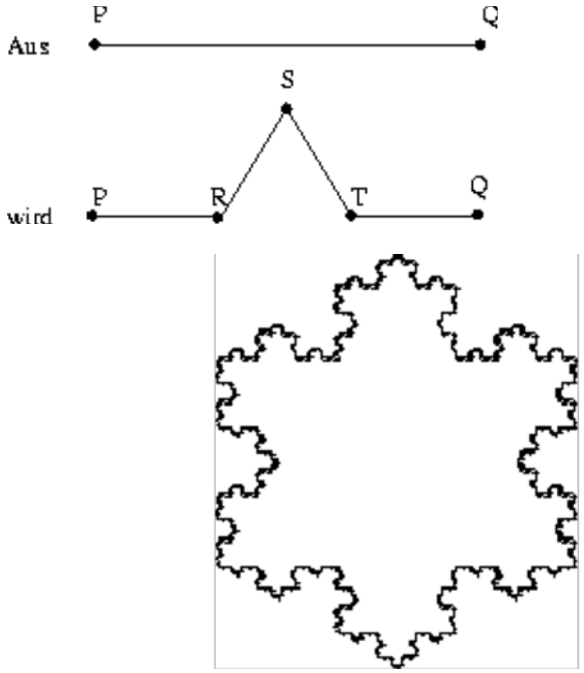
Ist das Orbit-Diagramm der logistischen Parabel $\Phi(x) = \alpha x(1-x)$

Es stellt die Anzahl und Lage der Häufungspunkte der Fixpunkt-Iteration dar in Abhängigkeit von α.
Für $1<\alpha<3$ existiert genau ein Häufungspunkt, und für $3<\alpha<4$ immer mehr und mehr Häufungspunkte des Orbits zu Startwert x_0 .
Bei $\alpha=3$: Verzweigung (Bifurcation)

6.5.2. Weitere fraktale Objekte:

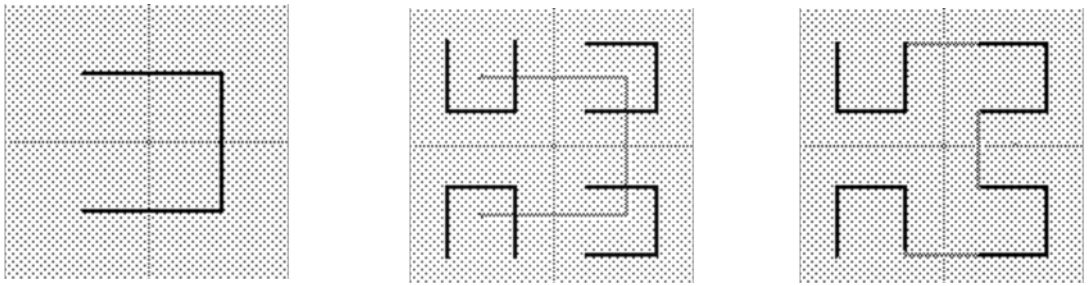
Koch'sche Schneeflocke:

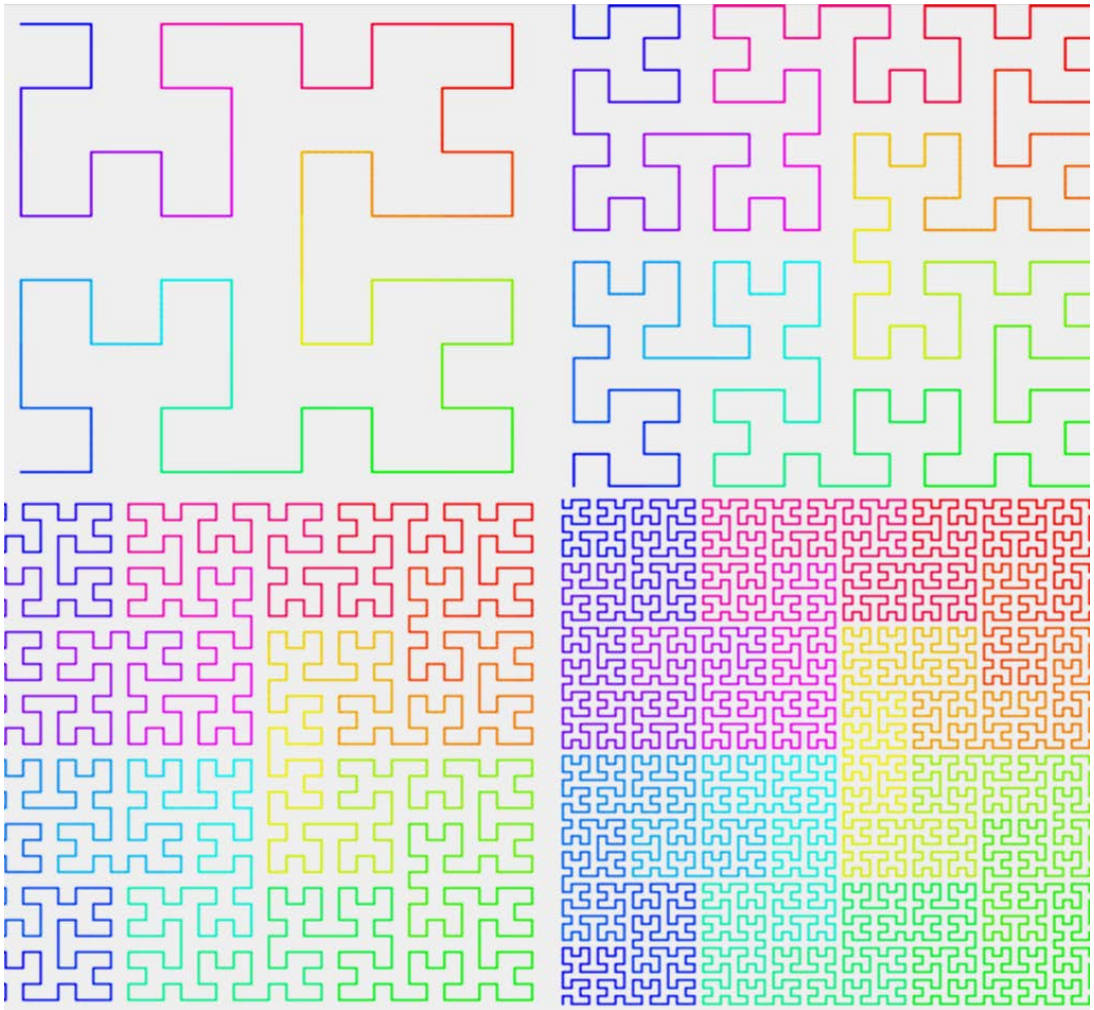




Anwendung: Modellierung in der Computergraphik

Raumfüllende Kurven, Hilbertkurve:



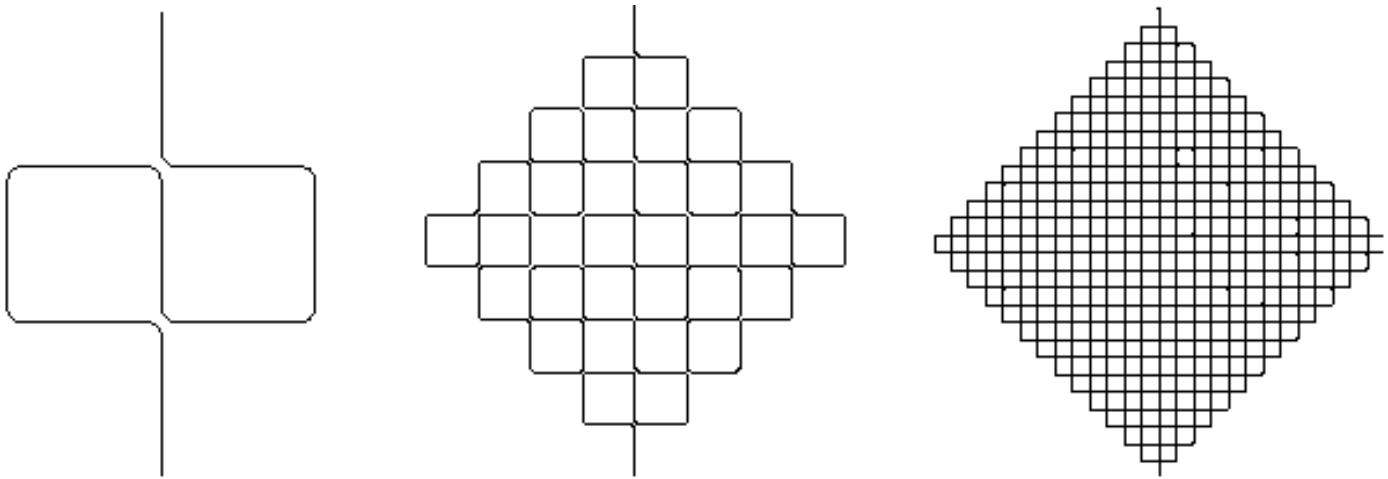


Rekursiv definiert. Im Grenzwert 2-dimensional!

Ermöglicht durch ‚1-dimensionalen' Weg das Durchsuchen eines höherdimensionalen Bereichs.
Anwendungen: z.B. Datenbanken.

Verwandte Themen:
Zelluläre Automaten, Spiel des Lebens,
Modellierung von Eis oder Strömung.

Ähnlich die Peano-Kurve:



6.5.3. Gradientenverfahren und Neuronale Netze:

Einfaches Neuronales Netz besteht aus n Knoten N_1 , ..., N_n ; jeder Knoten N_i empfängt ein Eingangssignal x_i , verstärkt dieses Signal mit einem Faktor w_i und gibt es an einen gemeinsamen Ausgangsknoten weiter, der alle eingehenden Signale aufsummiert zu

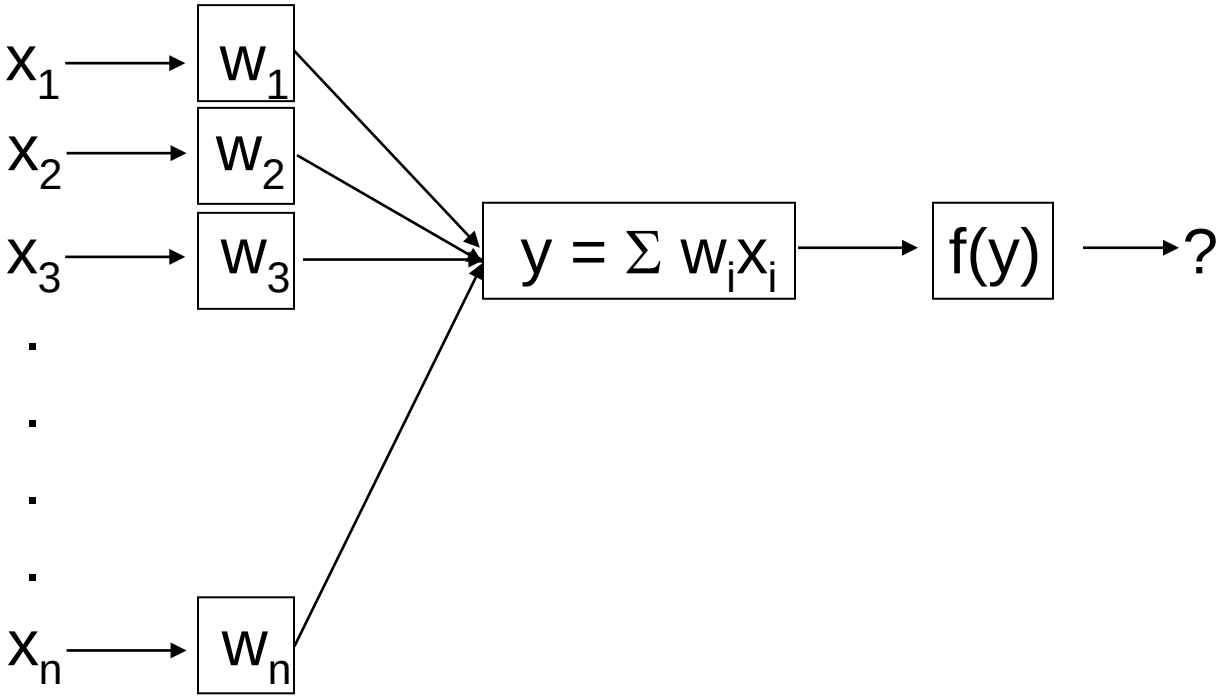
$$y = \sum_{i=1}^{n} w_i x_i$$

Danach kann der Wert y noch weiterbearbeitet werden durch eine Funktion f(.).
Abhängig von dem sich ergebenden Wert von f(y) sollen z.B. Ja/Nein-Entscheidungen getroffen werden:

$$f(y) = \begin{cases} 1 & falls \quad y \geq \alpha \\ 0 & falls \quad y < \alpha \end{cases}$$

entspricht einer Ja/Nein-Antwort

Einschichtiges Neuronales Netz:



Wir wollen mit dem NN einen Vorgang modellieren, bei dem für beliebig viele Beispiele B_j mit Eingabedaten x_j bekannt ist, welcher Ausgangswert y_j zu erwarten ist.

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}; \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}; \quad x^{(j)} = \begin{pmatrix} x_1^{(j)} \\ \vdots \\ x_n^{(j)} \end{pmatrix};$$

Wähle nun die Gewichte w so, dass

$$\sum_{i=1}^{n} w_i x_i^{(j)} = w^T x^{(j)} = x^{(j)^T} w \approx y_j, \qquad \textit{für} \quad j = 1,\ldots,m.$$

Also versuche, das NN durch Wahl der Gewichte so zu modellieren, dass es zu den gewünschten Ausgabewerten führt. Dies entspricht einem linearen Ausgleichsproblem der Form

$$\min_w \|Xw - y\| \qquad \textit{mit} \quad X = \begin{pmatrix} x^{(1)^T} & \cdots & x^{(m)^T} \end{pmatrix}$$

Lösung mit Normalgleichung $X^T X w = X^T y$ oder
QR-Zerlegung $X = QR$

Andere Variante:

Schrittweise iterative Verbesserung der Gewichte bei neuen Beispieldaten:

x_{neu} mit gewünschter Ausgabe y_{neu} soll möglichst gut erreicht werden.

Annahme: Bisherige Beispiele haben zu Gewichten w geführt. Einarbeiten der neuen Beispieldaten. Also gesucht:

Nullstelle oder Minimum von $\left(y_{neu} - w^T x_{neu}\right)^2 \stackrel{!}{=} 0$

Gradient dieser Funktion in Abhängigkeit von w:

$$\nabla\left(y_{neu}^2 + (w^T x_{neu})^2 - 2 y_{neu}(w^T x_{neu})\right) = const * x_{neu}$$

ist Suchrichtung zur Verbesserung der aktuellen Gewichte w:

$$w_{neu} = w_{alt} + \alpha x_{neu}$$

liefert

$$y = w_{neu}^T x_{neu} = w_{alt}{}^T x_{neu} + \alpha x_{neu}^T x_{neu} = y_{alt} + \alpha \left\| x_{neu} \right\|^2 \overset{!}{=} y_{neu}$$

Daher sollte $\alpha = \dfrac{y_{neu} - y_{alt}}{\left\| x_{neu} \right\|^2}$ gewählt werden.

Da wir aber nur eine kleine Änderung in Richtung des gewünschten Resultats erreichen wollen, und die bisherigen Testbeispiele, die zu dem alten w geführt haben, nicht vollständig vernachlässigen wollen, definieren wir eine Lernrate $\eta < 1$ und setzen

$$w_{neu} = w_{alt} + \eta \frac{y_{neu} - w_{alt}^T x_{neu}}{\left\| x_{neu} \right\|^2} x_{neu}$$

Lernregel für NN, verwandt mit Gradientenverfahren!

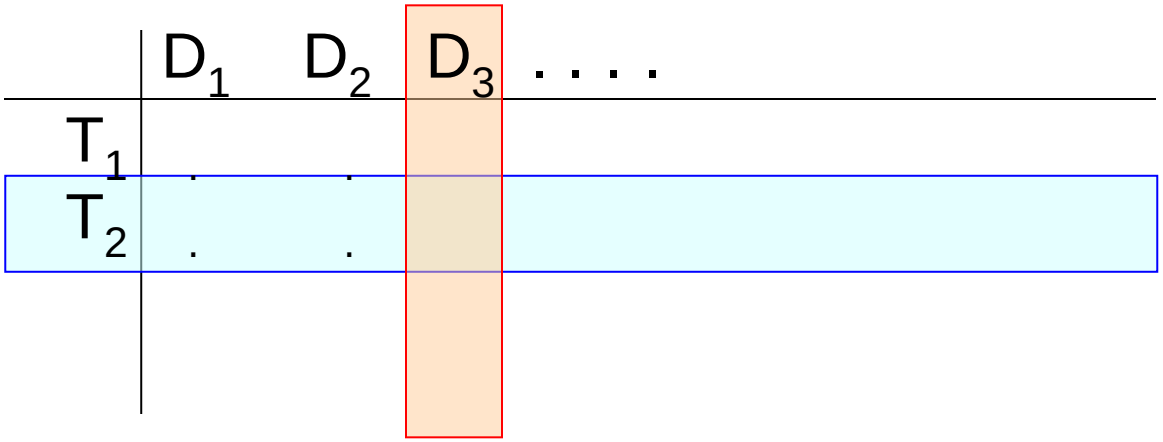
6.5.4. Information Retrieval und Vektoriteration:

Vektorraummodell für ‚Data Mining'

Liste von Dokumenten D_1 , ... , D_n , z.B. WWW-Seiten
Liste von Suchtermen T_1 , ... , T_m , z.B. alphabetisch geordnet:
Aachen, ABC, Auto, Bar, ... , Zug

Jeder Suchbegriff erscheint in jedem Dokument mit einem Bestimmten Gewicht, z.B. Gewicht ~ Häufigkeit.

Sammle diese Gewichte in der sog. Term-Dokument-Matrix (dünnbesetzte Matrix).
j-ter Spaltenvektor dieser Matrix listet für das j-te Dokument D_j die Darin enthaltenen Suchworte → *Vektor d_j .*
k-ter Zeilenvektor t_k listet für den k-ten Suchbegriff die relevanten Dokumente auf.



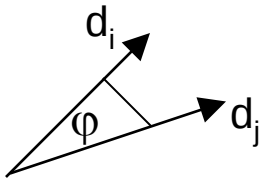
Vergleich zweier Dokumente auf Ähnlichkeit:

Dokument D_i repräsentiert durch Vektor der darin enthaltenen Suchbegriffe d_i , entsprechend d_j .

Vektoren ähnlich (kollinear), wenn Winkel zwischen ihnen sehr klein, d.h.

$$\cos(\sphericalangle(d_i, d_j)) = \frac{d_i^T d_j}{\|d_i\|_2 \cdot \|d_j\|_2}$$

nahe bei 1.



Eine Suchanfrage ist ein Spaltenvektor d, entspricht quasi einem Dokument, das mit anderen Dokumenten auf Ähnlichkeit verglichen werden soll.

Berechne Vektor $t = d^T*D = (d^T*D_1 \;\; d^T*D_2 \;\; ..)$.

Große Komponenten t_j entsprechen Dokumenten D_j , die mit der Suchanfrage am besten übereinstimmen.

Matrix D enthält sehr viel ‚Rauschen', (zufälliges Auftreten von Suchtermen in Dokumenten, Wortwahl, ...)

Modellreduktion:
Ersetze D durch System, das die *wesentlichen* Eigenschaften repräsentiert:

Berechne QR-Zerlegung von D : D=QR
Partitioniere R in der Form

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

dabei sei R_{22} gleich Null oder mit kleinen Einträgen. Dann repräsentieren die Zeilen von R_{22} ‚linear abhängige' Such-begriffe, die eigentlich überflüssig sind.

Ersetze D durch die Approximation kleinen Ranges:

$$\tilde{D} := Q\begin{pmatrix} R_{11} & R_{12} \\ \hline 0 & 0 \end{pmatrix} = (Q_1 \mid Q_2)\begin{pmatrix} R_{11} & R_{12} \\ \hline 0 & 0 \end{pmatrix} = Q_1 \cdot (R_{11} \quad R_{12})$$

und untersuche $t^T = d^T \cdot \tilde{D}$ anstatt $d^T \cdot D$

Verbesserung durch Pivotsuche: D*P=Q*R mit allgemeiner QR-Zerlegung incl. Permutation liefert $\tilde{D} := Q \cdot \tilde{R} \cdot P^T$ und neues t^T

Andere Methode des ‚Denoising' mittels Eigenwerten:

Betrachte Singulärwertzerlegung von A:

$$A = U\,D\,V,$$

dabei sind U und V Eigenvektormatrizen von A^TA und AA^T, und D ist eine rechteckige Diagonalmatrix, deren Diagonaleinträge >= 0 sind:

$$A = U \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & \cdots \\ 0 & \ddots & \ddots & \vdots & \\ \vdots & \ddots & \sigma_k & 0 & \\ 0 & \cdots & 0 & 0 & \ddots \\ \vdots & & & \ddots & \ddots \end{pmatrix} V$$

mit Singulärwerten $\sigma_j > 0$, $\mathrm{Rang}(A)=k$.

Ersetze D durch $\tilde{D}$, indem kleine σ_j durch 0 ersetzt werden.

Ergibt neue Matrix (Term-Dokument-Tabelle)

$$\tilde{A} = U\tilde{D}V$$ mit kleinerem Rang:

Diese neue Matrix enthält wieder die wesentliche Information von A .

A = UDV heißt Singulärwertzerlegung von A.
Und ergibt sich aus der Eigenwert-Zerlegung von A^TA, bzw. AA^T .

6.5.5. Suchmaschine Clever (HITS):

Teile die Dokumente im WWW in zwei Klassen:
- Hubs seien Webseiten, die eine Ansammlung interessanter Links darstellen
- eine angesehene Authority ist eine Webseite, die zu einem Thema interessantes Material liefert, und daher in vielen Hubs auftaucht.

Dies legt iteratives Verfahren zur Bestimmung relevanter Authorities und Hubs nahe:

Starte mit einer Menge von Kandidaten für Hubs und einer Menge von Kandidaten für Authorities. Diese Mengen sind wieder beschrieben durch Vektoren (v_i = Gewicht) mit Einträgen, die gewichtet Hubs oder Authorities darstellen.

Alle Webseiten sind wieder mittels Gewichtung verknüpft durch die Hub-Authority-Matrix B, die zu jeder Authority die Hubs angibt, die auf sie verweisen.

b_{ij} gibt das Gewicht an, mit dem die Authority A_j , j=1,...,n, im Hub H_i , i=1,...,m vorkommt.

Seien die Vektoren $H^{(0)}$ und $A^{(0)}$ Startannahmen für gute Hubs, bzw. Authorities.
Die i-te Komponente von $H^{(0)}:= B*A^{(0)}$ gibt daher an, mit welchem Gewicht der i-te Hub auf eine der Start-Authorities zeigt.

Entsprechend gibt die j-te Komponente von $A^{(0)}:= B^T*H^{(0)}$ an, mit welchem Gewicht die Start-Hubs auf die j-te Authority zeigen.

Verwende wechselseitige verschränkte Definition von gutem Hub, bzw. guter Authority:

Guter Hub zeigt auf viele gute Authorities;
Gute Authority erscheint in vielen guten Hubs.

Dies legt die folgende Iteration nahe:

$$A^{(k+1)} = \frac{B^T H^{(k)}}{\left\| B^T H^{(k)} \right\|} = \frac{B^T \left(B A^{(k)} \right)}{\left\| B^T \left(B A^{(k)} \right) \right\|} = \frac{(B^T B) A^{(k)}}{\left\| (B^T B) A^{(k)} \right\|}$$

oder

$$H^{(k+1)} = \frac{B A^{(k)}}{\left\| B A^{(k)} \right\|} = \frac{B \left(B^T H^{(k)} \right)}{\left\| B \left(B^T H^{(k)} \right) \right\|} = \frac{(B B^T) H^{(k)}}{\left\| (B B^T) H^{(k)} \right\|}$$

Dies entspricht zwei Vektoriterationen, die gegen die Eigenvektoren zu dem maximalem Eigenwert von B^TB , bzw. BB^T konvergieren.

Die Komponenten dieser Eigenvektoren geben dann eine Gewichtung guter Authority-, bzw. Hub-WWW-Seiten an.

Ein Eigenvektor stellt einen Fixpunkt dieser Iteration dar, also eine Ansammlung von Webseiten, die in gewisser Weise abgeschlossen (gesättigt, invariant) ist.

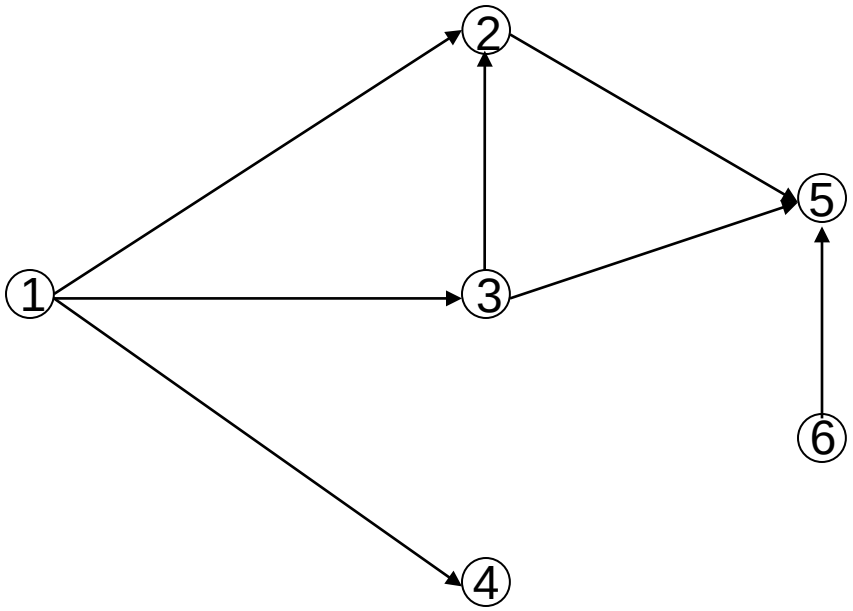
Der gefundene Eigenvektor zu maximalem Eigenwert liefert daher einen Fixpunkt maximalen Gewichts.

Die Einträge in dem Eigenvektor geben an, welches Gewicht eine Webseite in dem invarianten maximalen Eigenvektor hat.

Hohes Gewicht im Eigenvektor ← → Wichtige Webseite

6.5.6. Googles PageRank

Betrachte WWW als gewichteten Graphen mit Knoten (Webseiten) und Kanten (Links):



Erstelle dazu n x n – Matrix (n = Anzahl der Knoten)
Eintrag $a_{i,j} = 1/q$, wenn Kante von i nach j (insgesamt q Kanten von i)

Alle anderen Einträge in i-ter Zeile dann 0

Geht von i gar keine Kante aus, dann $a_{i,j} = 1/n$ für alle j.

$$\begin{array}{l} 1: \\ 2: \\ 3: \\ 4: \\ 5: \\ 6: \end{array} \Rightarrow \begin{pmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = A$$

Die sich ergebende Matrix A ist eine sog. stochastische Matrix,
Da die Summe der Einträge in einer Zeile stets gleich 1 ist.

$$A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = 1 \cdot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

1 ist der größte Eigenwert von A mit Eigenvektor $(1,1,\ldots,1)^T$.

Die Komponenten v_i des Linkseigenvektors v mit $v^T A = 1\, v^T$ beschreiben die Gewichtung eines zufälligen Springens von einer Seite zur nächsten. Dies lässt sich interpretieren als die Bedeutung einer Webseite.

Berechnung von v durch Vektoriteration.

Probleme: 1 ist mehrfacher Eigenwert, Konvergenz!
Veränderung der Gewichtung durch Manipulation

Daher neue Google-Matrix

$$B := sA + (1-s)\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} u^T$$

mit personalisiertem Vektor u, so dass: $u_1 + \ldots + u_n = 1$.

Effekt: B ebenfalls stochastisch

1 ist nun einfacher Eigenwert → Konvergenz eindeutig, schneller!

Mittels u lassen sich „per Hand“ Webseiten gewichten

Google verwendet angeblich s=0.85

In Lösungsvektor v gibt Komponente v_i die Gewichtung der i-ten Webseite an:

PageRank von Seite i.