# 6.2.5. Newtonverfahren und Minimierung:

Nullstelle einer Funktion f: $R^n \rightarrow R^n$ ,

$$f(x) = f(x_1,\ldots,x_n) = \begin{pmatrix} f_1(x_1,\ldots,x_n) \\ \vdots \\ f_n(x_1,\ldots,x_n) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = 0$$

Jacobi-Matrix der Ableitungen von f:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Newtonverfahren im $R^n$ :

$$x^{k+1} := x^k - inv(J) \cdot f(x^k)$$

Vektor = Vektor – Matrix * Vektor

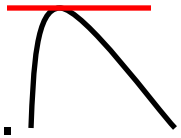
In jedem Schritt ist ein neues Gleichungss. in $J(x^k)$ zu lösen!

# 6.2.6. Minimierungsproblem:

**Gegeben: $F: R^n \rightarrow R$ , $F(x_1,...,x_n) \in R$**

**Gesucht: (relatives) Minimum / Maximum**

Bestimme dazu Nullstelle der Ableitung
(entspricht waagrechter Tangente, also Extremwert).



Gradientenvektor von F:

$$\nabla F = f(x_1,...,x_n) = \begin{pmatrix} f_1(x_1,...,x_n) \\ \vdots \\ f_n(x_1,...,x_n) \end{pmatrix} = \begin{pmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{pmatrix} \overset{!}{=} 0$$

Jacobi-matrix J der ersten Ableitungen von f entspricht der sog. Hesse-matrix H der zweiten Ableitungen von F:

$$H(F) = J(f) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} = \left( \frac{\partial^2 F}{\partial x_i \partial x_j} \right)_{i,j=1}^{n}$$

Newtonverfahren zur Bestimmung der Nullstelle des Gradienten ergibt:

$$x^{k+1} = x^k - inv(H) \cdot \nabla F(x^k)$$

Löse dazu in jedem Schritt ein lineares Gleichungssystem mit sog. Hesse-matrix $H(x^k)$ !

Problem, falls H an der Stelle $x^k$ singulär.
**Billiger:** Quasi-Newton-Verfahren, ersetze H durch billiges B.

# 6.2.7. Nichtlineare Ausgleichsrechnung:
# Gauss-Newton-Verfahren

$$\min_x \left\| \begin{pmatrix} f_1(x_1,\ldots,x_n) \\ \vdots \\ f_m(x_1,\ldots,x_n) \end{pmatrix} - b \right\|_2^2 , \qquad \text{(vgl. min ||Ax-b|| )}$$

Erster Schritt:

Linearisierung mittels Jacobi-matrix und Taylorreihe

$$f(x) = f(x^k) + J(x - x^k) + \cancel{O(h^2)}$$

Damit ergibt sich neue Iterierte $x^{k+1}$ aus der Lösung des linearen Ausgleichsproblems

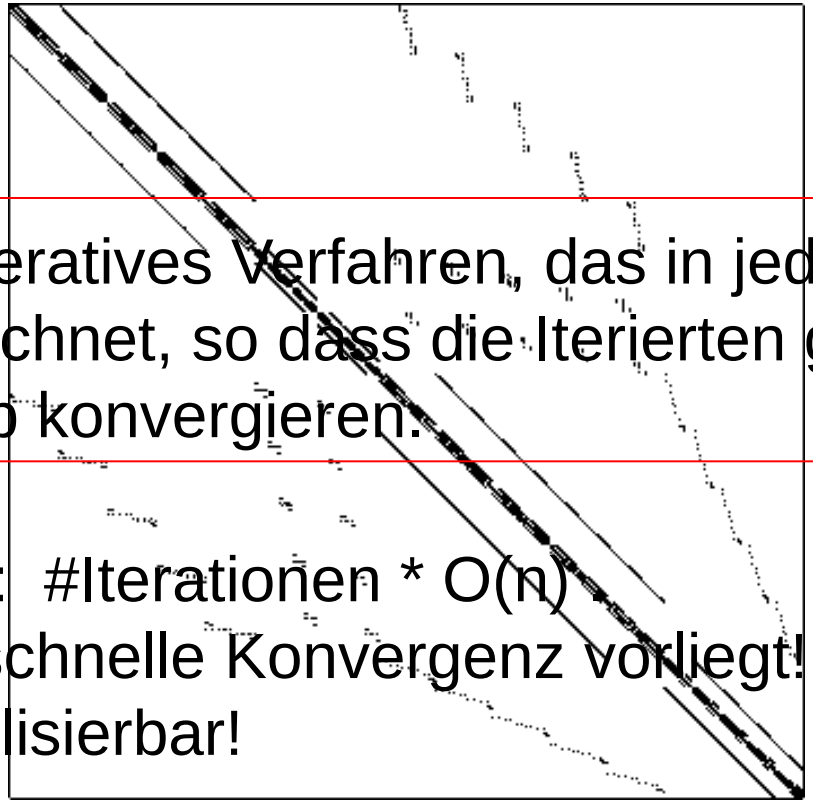
$$\min_x \left\| \left( f(x^k) + J^k (x - x^k) \right) - b \right\|_2^2$$

Neue Linearisierung an der Stelle $x^{k+1}$ ergibt neues lineares Ausgleichsproblem mit neuer Matrix $J^{k+1}$.

# 6.3. Iterative Lösung linearer Gleichungssysteme

Großes lineares dünnbesetztes Gleichungssystem $A x = b$

Gauss-Elimination nutzt in der Regel die Dünnbesetztheit nicht aus und führt meist auf Kosten $O(n^3)$;
Im Gegensatz dazu ist oft nur $O(n)$ Speicherbedarf für Matrix A



Idee: Formuliere iteratives Verfahren, das in jedem Schritt nur Matrix*Vektor berechnet, so dass die Iterierten gegen die Lösung von $Ax = b$ konvergieren.

Gesamtkosten: #Iterationen * $O(n)$ .
Effizient, falls schnelle Konvergenz vorliegt!
Einfach parallelisierbar!

# 6.3.1. Stationäre Methoden:

**Richardson-Verfahren:**
Formulierung eines passenden Fixpunktproblems

$$b = Ax = (A - I)x + x \Leftrightarrow x = b + (I - A)x =: \Phi(x)$$

Daraus ergibt sich die Iteration:

$$x_{k+1} = b + (I - A)x_k = x_k + (b - Ax_k) = x_k + r_k = \Phi(x_k)$$

mit Residuum $r_k$ .

**Frage:**
Wann konvergiert die ausgehend von einem $x_0$ definierte Folge $x_k$ gegen die gesuchte Lösung $\bar{x} = A^{-1}b$ ?

## Betrachte dazu den Fehler im k-ten Schritt:

$$\bar{x} - x_{k+1} = \bar{x} - x_k - (b - Ax_k) = \bar{x} - x_k - (A\bar{x} - Ax_k) = (I - A)(\bar{x} - x_k)$$

Ergibt in der Norm $\left\|\bar{x} - x_{k+1}\right\|_2 = \left\|(I - A)(\bar{x} - x_k)\right\|_2 \le \left\|I - A\right\|_2 \cdot \left\|\bar{x} - x_k\right\|_2$

und daher $\left\|\bar{x} - x_k\right\|_2 \le \left\|I - A\right\|_2^k \cdot \left\|\bar{x} - x_0\right\|_2$

**Daher liegt Konvergenz vor für $\| I - A \|_2 < 1$ .**

Dies entspricht der Kontraktionsbedingung für die Iterationsfunktion $\Phi(x)$:

$$\left\|\Phi(x) - \Phi(y)\right\|_2 \le \left\|I - A\right\|_2 \cdot \left\|x - y\right\|_2$$

Richardsonverfahren ist daher nur sinnvoll, wenn $A \approx I$ !

# Geschickter:

Wende das Verfahren auf das modifizierte Problem an:

$$D = diag(A); \qquad \left(D^{-1}A\right)x = \left(D^{-1}b\right) \quad \Leftrightarrow \quad Ax = b$$

Die Bedingung $D^{-1}A \approx I$ ist besser erfüllt!

Bezeichnung: $\tilde{A} = D^{-1}A$ und $\tilde{b} = D^{-1}b$

ergibt neues Gleichungssystem $\tilde{A}x = \tilde{b}$

Richardson für das tilde-System liefert dann:

$$x_{k+1} = x_k + (\tilde{b} - \tilde{A}x_k) = x_k + D^{-1}(b - Ax_k) \quad \text{oder}$$

$$Dx_{k+1} = Dx_k + (b - Ax_k) = b + (D - A)x_k$$

Dies ist das Jacobiverfahren zur iterativen Lösung von Ax = b

Wesentliche Kosten in jedem Schritt: $Ax_k$

Konvergent, falls $\left\| I - D^{-1}A \right\|_2 < 1$

*Notwendig: Diagonalmatrix D regulär*!

Allgemeinere Idee zur Formulierung einer Iterationsfunktion:

Matrix-splitting

$$L: \quad Unterdiagonalteil \quad von \quad A$$

$$U: \quad Oberdiagonalteil \quad von \quad A$$

$$b = Ax = (L + D + U)x = (L + D)x + Ux$$

führt auf Iteration

$$(L + D)x_{k+1} = b - Ux_k = b - (A - L - D)x_k$$

$$= (L + D)x_k + (b - Ax_k)$$

Dies ist das Gauss-Seidel-Verfahren

$$x_{k+1} = x_k + (L+D)^{-1}(b - Ax_k)$$
$$= (L+D)^{-1}b + \left(I - (L+D)^{-1}A\right)x_k$$

In jedem Schritt ist dabei nur ein Dreiecksgleichungssystem zu lösen.

Das Verfahren ist konvergent, falls $\left\| I - (L+D)^{-1}A \right\|_2 < 1$

Gauss-Seidel ist äquivalent zu Richardson angewendet auf

$$\left((L+D)^{-1}A\right)x = \left((L+D)^{-1}b\right) \quad \Leftrightarrow \quad Ax = b$$

# Gauss-Seidel-Iteration

Versuche immer, die neueste Information zu verwenden!

Jacobi Iteration:

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm} x_m^{(k)} - \sum_{m=j+1}^{n} a_{jm} x_m^{(k)}$$

Gauss-Seidel Iteration: bereits berechnet

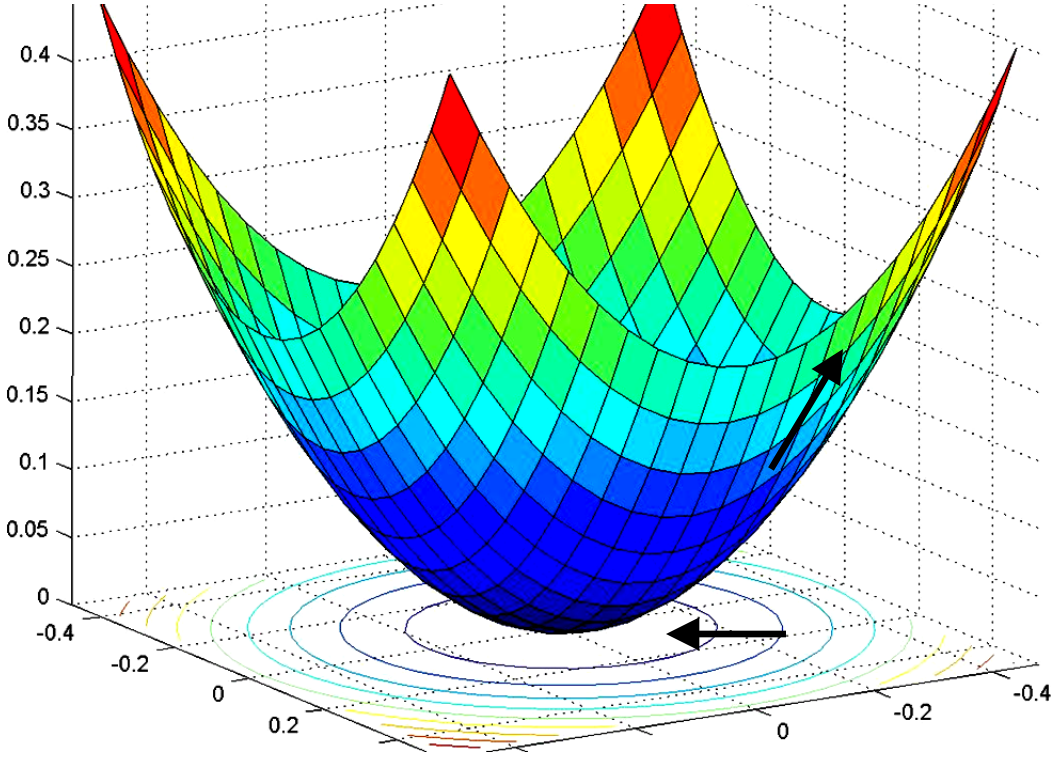
$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm} x_m^{(k+1)} - \sum_{m=j+1}^{n} a_{jm} x_m^{(k)}$$

# 6.3.2. Nichtstationäre Verfahren

**Das Gradientenverfahren für symmetrisch positiv definite Matrix A:**

**Betrachte Funktion F(x): $R^n \rightarrow R$ ,** $$F(x) = \frac{1}{2} x^T A x - b^T x$$

F beschreibt einen Paraboloid im $R^n$:




x-Ebene

-grad(F)

Minimum dieser Funktion ist wieder der Punkt mit waagrechter Tangente, also die Stelle mit Gradient gleich Null:

$$\nabla F(x) = Ax - b = 0 \Leftrightarrow Ax = b$$

Stelle x, an der der Paraboloid sein Minimum annimmt ist gleich der gesuchten Lösung des Gleichungssystems!

**Betrachte Minimierungsaufgabe!**

Von aktueller Stelle $x_k$ aus soll die nächste Iterierte $x_{k+1}$ so gewählt werden, dass sie näher am Minimum liegt.

$$x_{k+1} = x_k + \alpha_k d_k$$

mit Suchrichtung $d_k$ und Schrittweite $\alpha_k$.

Finde Suchrichtung so, dass Funktionswerte kleiner werden:

# Abstiegsrichung ist gegeben durch Richtung des negativen Gradienten!

Denn Richtungsableitung in Richtung $n$ ist gleich $\nabla F \cdot n$, und wird am betragsgrößten für $n = \nabla F$ nämlich $\left\|\nabla F\right\|^2$

Daher ist $n = -\nabla F$ lokal die Richtung des steilsten Abstiegs zum Minimum.

Daher verkleinern sich die Funktionswerte auf jeden Fall, wenn man in diese Abstiegsrichtung geht.

Also wähle $$x_{k+1} = x_k - \alpha_k \nabla F(x_k) = x_k + \alpha_k (b - Ax_k)$$

Noch zu bestimmten ist optimale Schrittweite $\alpha_k$, die am nähesten ans Minimum führt!

Betrachte dazu ein-dimensionale Minimierung:

$$\min_\alpha g(\alpha) := \min_\alpha \left(F(x_k + \alpha(b - Ax_k))\right) =$$

$$\min_\alpha \left(\frac{1}{2}(x_k + \alpha d_k)^T A(x_k + \alpha d_k) - b^T(x_k + \alpha d_k)\right) =$$

$$\min_\alpha \left(\frac{1}{2}\alpha^2 d_k^T A d_k - \alpha d_k^T d_k + \frac{1}{2} x_k^T A x_k - x_k^T b\right)$$

mit Lösung $\alpha_k = d_k^T d_k \,/\, d_k^T A d_k$ , $d_k := b - Ax_k$

Insgesamt:

$$x_{k+1} = x_k + \frac{\|b - Ax_k\|_2^2}{(b - Ax_k)^T A(b - Ax_k)} \cdot (b - Ax_k)$$

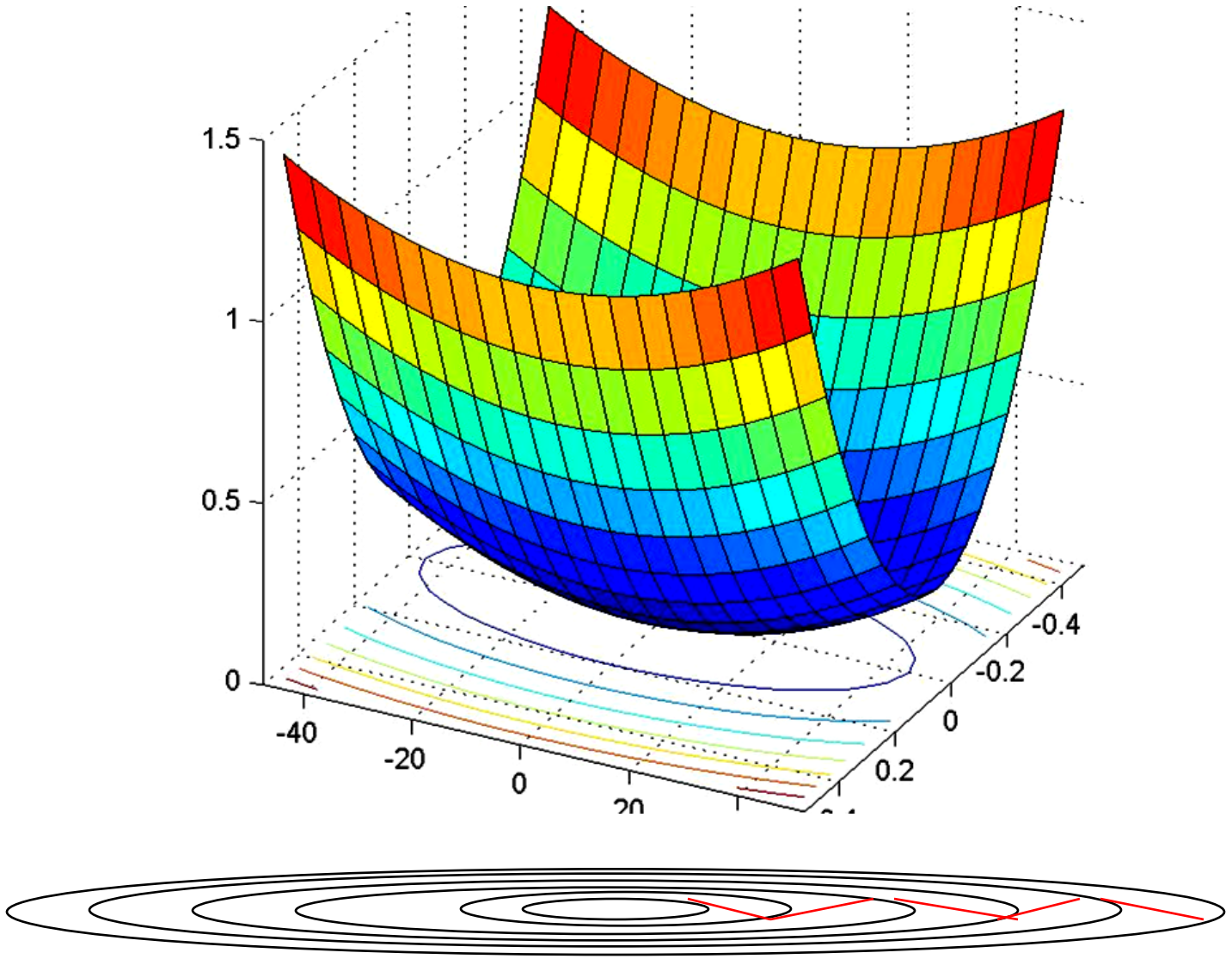
Dazugehörige Fixpunktgleichung ist

$$x = \Phi(x) := x + \frac{\left\| b - Ax \right\|_2^2}{(b - Ax)^T A(b - Ax)} \cdot (b - Ax)$$

mit Fixpunkt $\bar{x} = A^{-1}b$

Ergibt Verfahren des steilsten Abstiegs (‚steepest descent') oder Gradientenverfahren

Nachteil:
Bei stark verzerrtem Paraboloiden ergibt sich sehr langsame Konvergenz.




Für A ≈ I ist der Paraboloid unverzerrt, die Höhenlinien fast kreisförmig → schnelle Konvergenz!

Daher versucht man, Ax=b zu präkonditionieren:

Ersetze Ax=b durch $M^{-1}Ax = M^{-1}b$ mit $M^{-1}A \approx I$

Bessere Variante eines Gradientenverfahren:

Verfahren der konjugierten Gradienten, kurz cg-Verfahren.

Suchrichtung nicht der negative Gradient selbst, sondern die Projektion des Gradienten, so dass alle Suchrichtungen in gewisser Weise orthogonal zueinander sind

Genauer: Suchrichtungen seien A-konjugiert, d.h.

$$d_k^T A d_j = 0 \qquad \text{für} \qquad j \neq k$$

Damit ergibt sich iteratives Verfahren, das nach k Schritten jeweils die beste Näherung an die Lösung in einem k-dimensionalen Unterraum liefert, und daher nach n Schritten fertig ist (in exakter Arithmetik).

# Conjugate Gradient Method

Bessere Abstiegsrichtung, global optimal:

$x_{k+1} := x_k + \alpha_k p_k$ als neue Suchrichtung an Stelle des Gradienten

Projektion des Gradienten,
so dass A-conjugate (orthogonal) zu allen
vorherigen Suchrichtungen:

$p_k \perp Ap_j$ for all $j<k$ oder $p_k \perp_A p_j$ oder $p_k^T A p_j = 0$ für $j<k$

$\alpha_k$ wieder durch 1-dimensionale Minimierung wie vorher
(für verwendetes $p_k$)

# Conjugate Gradient Algorithm

$x_0=0$; $r_0 = b - A\,x_0$ ;

For k=1,2,…: $\beta_{k-1} = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$ ; ($\beta_0=0$)

$p_k = r_{k-1} + \beta_{k-1}\, p_{k-1}$;

$\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$ ;

$x_k = x_{k-1} + \alpha\, p_k$ ;

$r_k = r_{k-1} - \alpha_k\, A p_k$ ;

if $|| r_k || < \varepsilon$ : STOP

# Eigenschaften des cg-Verfahrens:

Bestimme in Unterraum $(b, Ab, A^2b, \ldots, A^kb)$ die beste Lösung mit minimalem Fehler: Optimal!

Algorithmus ist billig – in jedem Schritt nur Ax.

In dieser Form nur möglich für symmetrisch positiv definites A.
Für allgemeine Matrix:

- GMRES (optimal, aber teurer)
- BiCG (nicht optimal, aber billig)

# 6.4. Eigenwerte und Vektoriteration

Eigenvektor $v \neq 0$ und Eigenwert $\lambda$ 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:

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

$\Lambda$ 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 $\lambda_1$ 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 $\dfrac{A^k x_0}{\left\| A^k x_0 \right\|} \to \dfrac{\lambda_1^k \left( c_1 v_1 \right)}{\left\| \lambda_1^k \left( c_1 v_1 \right) \right\|} = \dfrac{c_1}{|c_1|} v_1 = \pm v_1$

$x_k$ 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 mit orthonormalem Q:

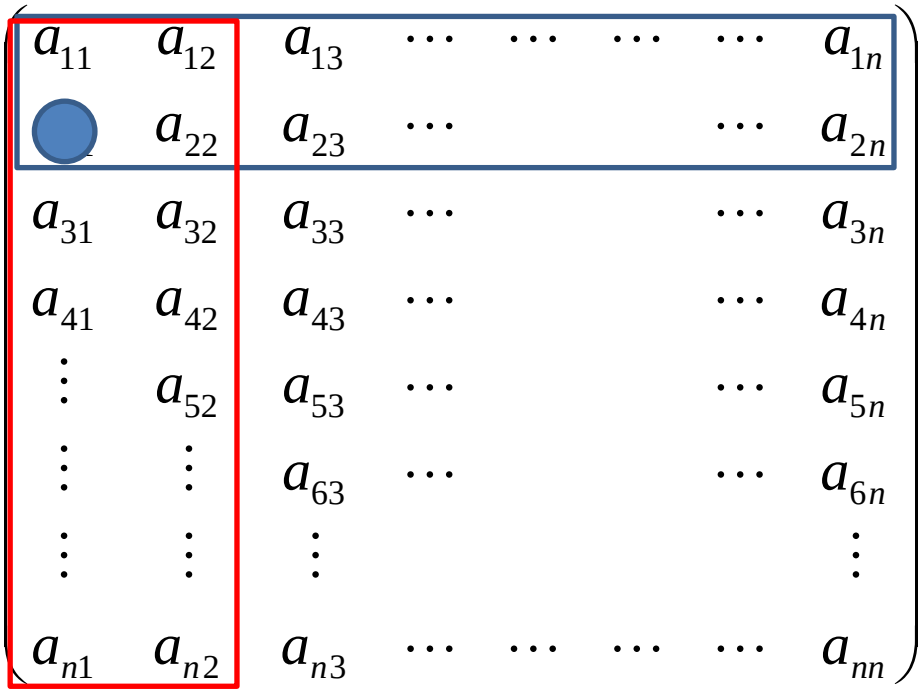
$$A_{neu} = Q*A*Q^T$$

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

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

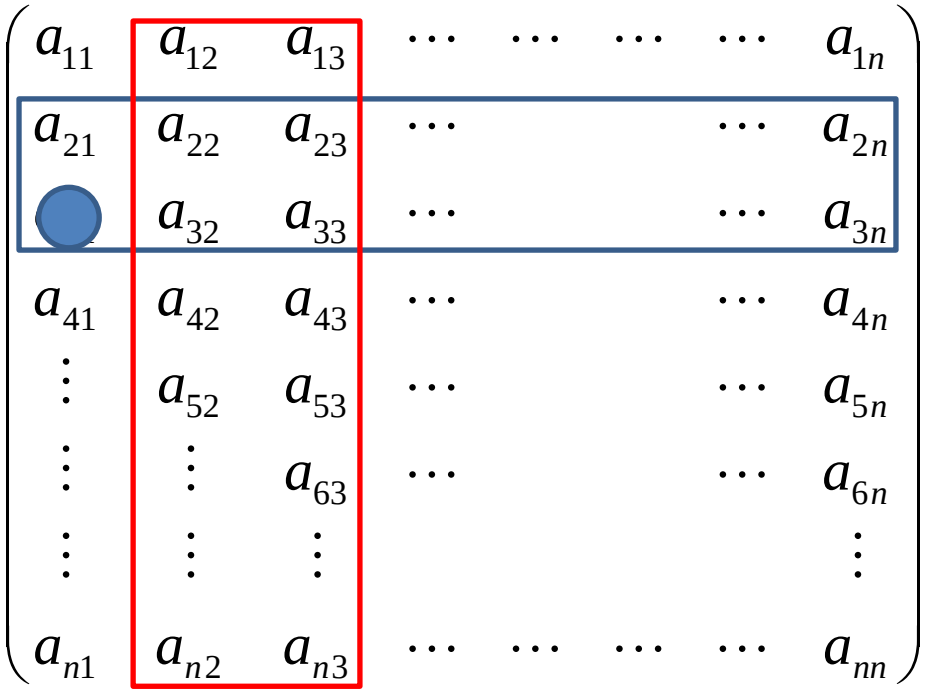
Matrix wie $Q*A*Q^T$, Eigenwert gleich, Eigenvektor wie Qx

# Welches Q? Givens?



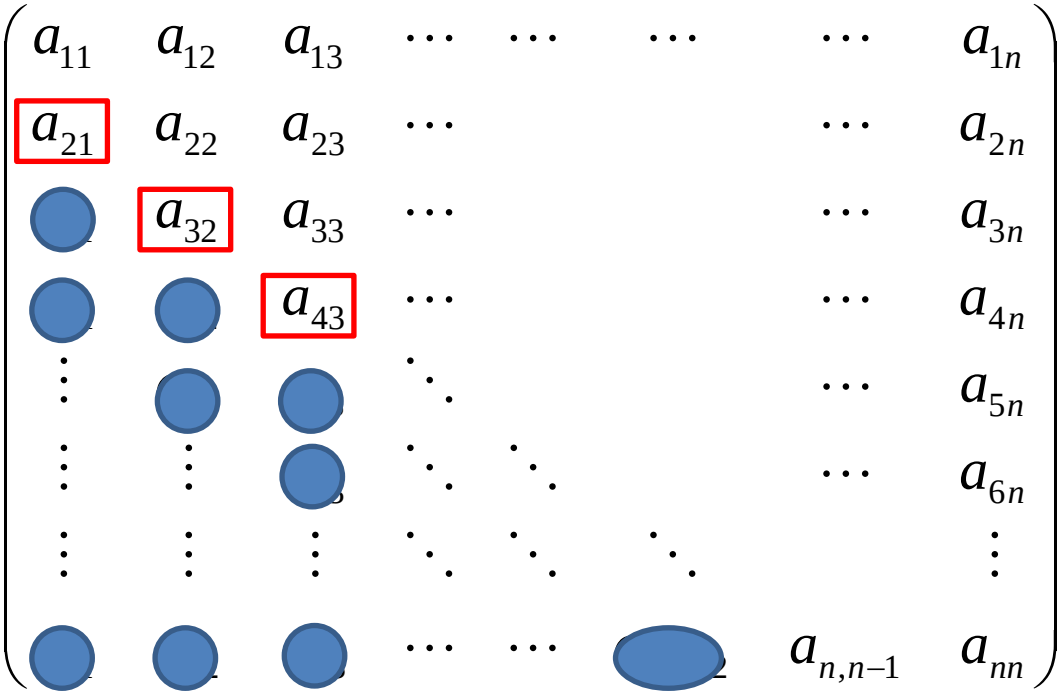

Von links Givens zur Elimination von $a_{21}$?
Anwendung von rechts füllt $a_{21}$ wieder auf!

# 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!

# 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. $\Phi$. 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, ...) → $\pm 1$

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 $\Phi$ ist dann nicht mehr anziehend,
sondern abstoßend!

Erinnerung: Fixpunkt und Ableitung am Fixpunkt sind

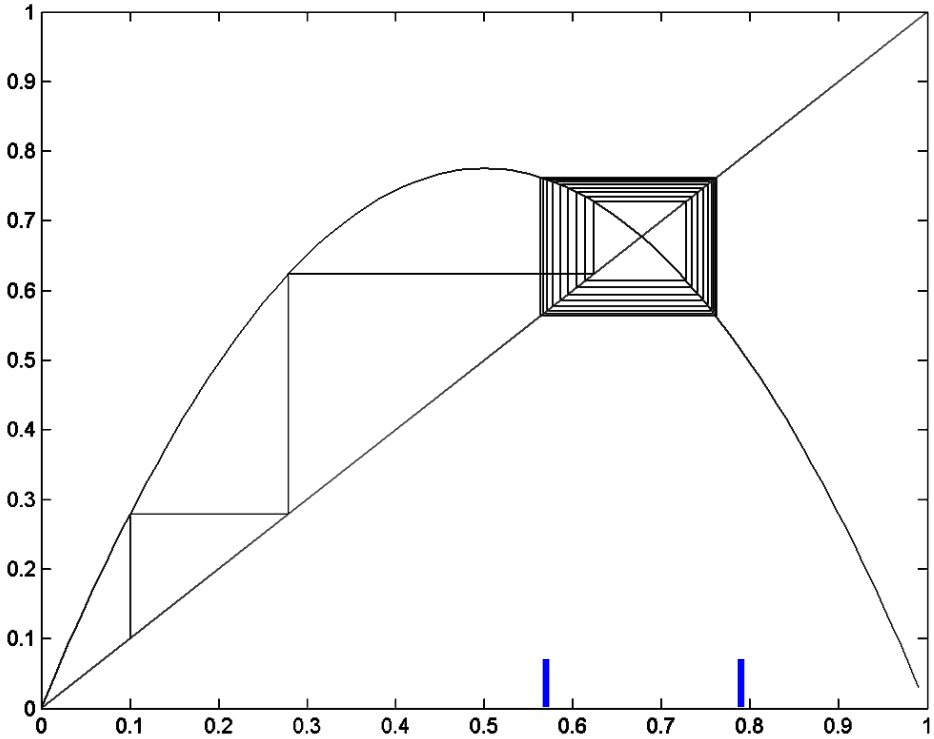
$$\bar{x} = \frac{\alpha - 1}{\alpha}; \qquad \Phi'(\bar{x}) = 2 - \alpha;$$

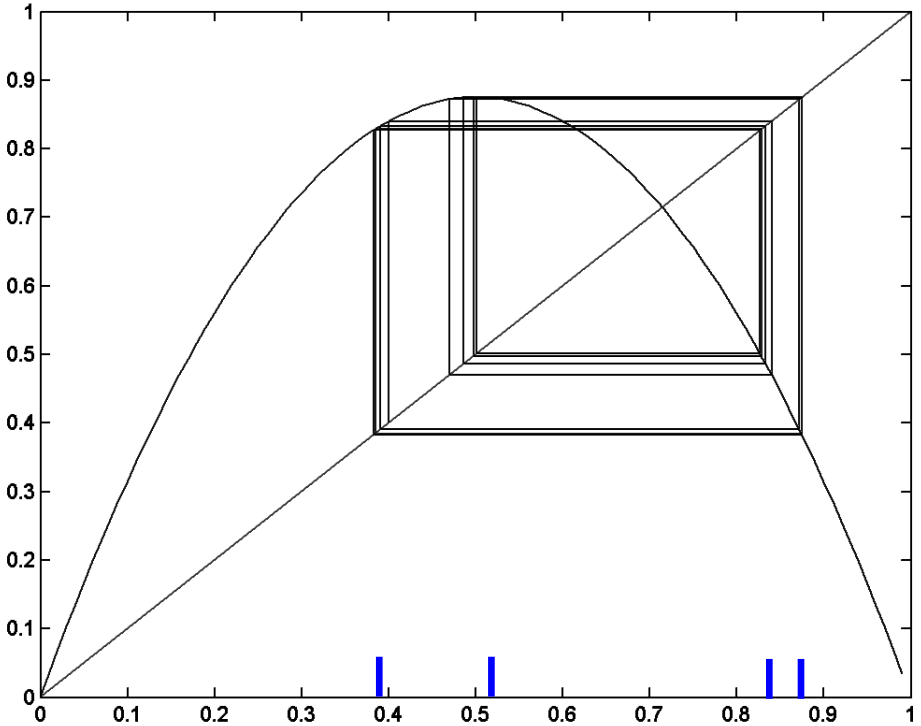
Für $\alpha = 3.1$ erhalten wir zwei Attraktoren, die nun anziehende Fixpunkte der neuen Iterationsfunktion

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

sind.

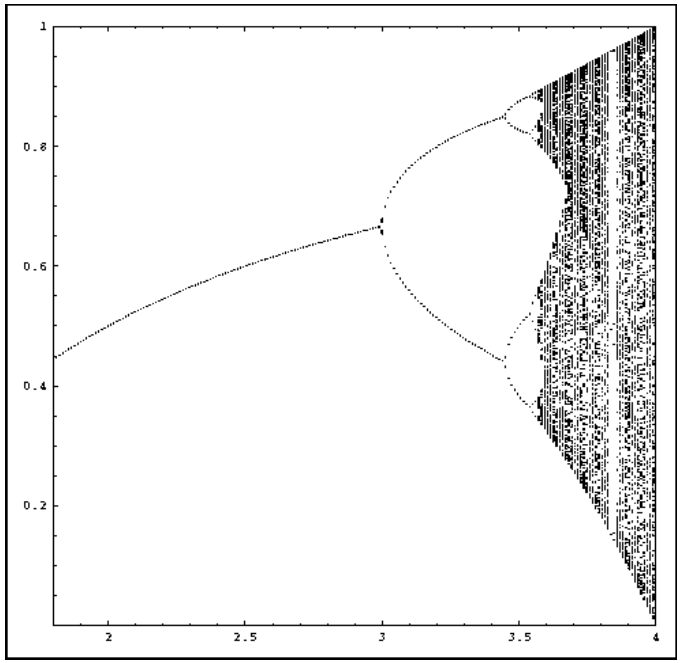
# Konvergente Teilfolgen $x_{2k}$ und $x_{2k+1}$ .




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

(**MATLAB: fixpunkt.m)**

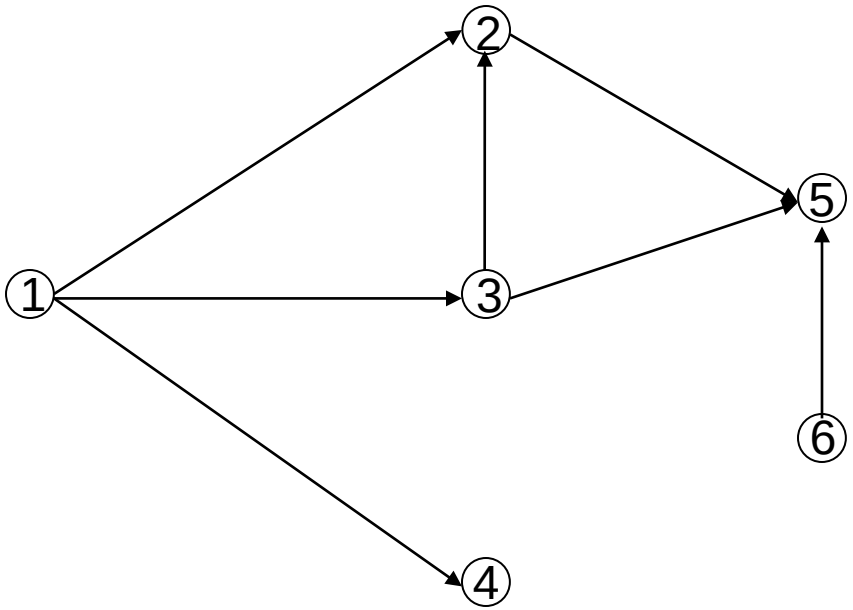



Ist das Orbit-Diagramm der logistischen Parabel $\Phi(x) = \alpha x(1-x)$

Es stellt die Anzahl und Lage der Häufungspunkte der Fixpunkt-Iteration dar in Abhängigkeit von $\alpha$.
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)

# 6.5.6. 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.