NUMERISCHE METHODEN

Zusammenfassung der Vorlesung von
Dr. V. Gradinaru

Lukas Cavigelli, Juli 2011
lukasc@ee.ethz.ch

NULLSTELLEN-SUCHE

Gesucht: Werte von x, für die $f(x) = 0$

KONVERGENZORDNUNG

Absoluter Fehler:

$$e_{abs,k} = x_k - x^*$$

Relativer Fehler:

$$e_{rel,k} = \frac{|x_k - x_{k-1}|}{|x_k|}$$

Konvergenzordnung:
Eine konvergente Folge $x_k \in \mathbb{R}^n, k = 0,1,\dots$ konvergiert mit Ordnung p nach $x^* \in \mathbb{R}^n$, wenn

$$\exists C > 0: \underbrace{\|x_{k+1} - x^*\|}_{e_{k+1}} \leq C \underbrace{\|x_k - x^*\|^p}_{e_k} \quad \forall k \in \mathbb{N}_0$$

Fehlergesetz:

$$e_{k+1} \approx C e_k^p$$

mit Konvergenzordnung p.
Bsp: $p = 2, C = 1, e_0 = 10^{-1} \rightarrow e_1 \approx 10^{-2}, e_2 \approx 10^{-4}, \dots$
Numerisch:

$$p \approx \frac{\log(e_{k+2}/e_{k+1})}{\log(e_{k+1}/e_k)}, \qquad e_k = |x_k - x^*|$$

FIXPUNKT-ITERATION

$F(x)$ finden, so dass $f(x) = 0 \Leftrightarrow F(x) = x$.
Es gibt verschiedene mögliche $F(x)$, die das erfüllen. Die Konvergenz ist i.A. für verschiedene F unterschiedlich.
Aufpassen auf Polstellen bzgl. Konvergenz. Abh. vom Startwert.
Danach Fixpunkt-Iteration, also

$$x_{n+1} = F(x_n), \qquad n = 0,1,\dots$$

```
function x = fixIt(F, x0, tol, kmax)
  x(1) = x0;
  for i=1:kmax
    x(i+1) = F(x(i));
    if abs(x(i+1) - x(i)) < tol
      break; end; end
```

Banach'scher Fixpunktsatz:
Sei g eine definierte und stetige Funktion im Intervall $D = [a, b]$, F eine Selbstabbildung ($F: D \rightarrow D \Leftrightarrow F(D) \subset D$) und *Kontraktion*, d.h.:

$$\exists 0 < L < 1: \|F(y) - F(x)\| \leq L\|y - x\| \ \forall x, y \in D$$

$\Rightarrow$ die Fixpunktiteration auf D konvergiert gegen Fixpunkt x^*
Bemerkung: Ist $g \in C^1$, genügt $|F'(x)| \leq L < 1 \forall x \in D$
Arten von Fixpunkten:
- attraktiv, wenn $|F'(x^*)| < 1$
- repulsiv, wenn $|F'(x^*)| > 1$

SEKANTEN-VERFAHREN

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

NEWTON-VERFAHREN

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

Konvergenz quadratisch, wenn $f'(x^*) \neq 0$
Konvergenz linear, wenn $f'(x^*) = 0$

GEDÄMPFTES NEWTON-VERFAHREN

Gedämpftes Newton-Verfahren:

$$x_{k+1} = x_k - \lambda_k \frac{f(x_k)}{f'(x_k)}, \qquad \lambda_k > 0$$

Monotonie-Test: $\lambda_k > 0$: $\|\Delta x_{k+1}\| \leq (1 - \lambda_k/2)\|\Delta x_k\|$
Tentative simplified: $\Delta x_k = f'(x_k)^{-1} f(x_k)$

$$\Delta x_{k+1} = f'(x_k)^{-1} f(x_k - \lambda_k \Delta x_k)$$

Tentative: $\Delta x_k = f'(x_k)^{-1} f(x_k)$

$$\Delta x_{k+1} = f'(x_k - \lambda_k \Delta x_k)^{-1} f(x_k - \lambda_k \Delta x_k)$$

Implementierung:

```
function dampedNewton(a, x0, maxit, lambda_min)
  q=0.5; x0_orig = x0;
  for it=1:maxit
    f=atan(a*x0); df=a/(1+(a*x0)^2); s= f/df; x1=x0-s; lambda=1;
    f=atan(a*x1); df=a/(1+(a*x1)^2); st=f/df;
    while norm(st)>(1-lambda/2)*norm(s)
      lambda=q*lambda;
      if lambda<lambda_min; break; end
      x1=x0-lambda*s; f=atan(a*x1);
      df=a/(1+(a*x1)^2); % tentative simplified Newton corr.
      st=f/df; end
    if abs(s)<1e-6
      plot(x0_orig,a,'*'); hold on; break; end
    x0=x1; end
```

DAS IST NOCH NICHT GUT!!!

QUASI-NEWTON-VERFAHREN

$$\vec{x}_{k+1} = \vec{x}_k - J(\vec{x}_k)^{-1} f(\vec{x}_k), \qquad J(\vec{x}_k) \text{ regulär}$$

1. (nicht-lineares) Gleichungssystem in Standardform bringen:

$$\vec{f}(\vec{x}) = 0, \qquad f: \mathbb{R}^n \rightarrow \mathbb{R}^n, \vec{x} \mapsto \vec{f}(\vec{x})$$

2. (Variante 1) Jacobi-Matrix berechnen

$$J(\vec{x}) = \left(\frac{\partial f_i(x_1, \dots, x_n)}{\partial x_j}\right)_{i,j=1,\dots,n} = \cdots \text{für } n = 2$$

2. (Variante 2) Jacobi-Matrix approximieren

$$\tilde{J}(\vec{x}) = \left(\frac{1}{2h}\left(\vec{f}(\vec{x} + h\vec{e}_i) - f(\vec{x} - h\vec{e}_i)\right)\right)_{i=1,\dots,n} \approx J(\vec{x})$$

3. Quasi-Newton anwenden

```
function X = QuasiNewton( x0, tol, nMax )
  h = 1; X(:, 1) = x0;
  for i=1:nMax % Lineares Gl.-System wird geloest !!
    dx = JacobiM( X(:, i), h) \ f(X(:, i)); % für Variante 2
    X(:, i+1) = X(:, i) - dx;
    if norm(dx) < tol; break; end
  end
end;
```

QR-ZERLEGUNG

GRAM-SCHMIDT

Schmidt'sches Orthogonalisierungs-Verfahren:
$A \coloneqq (a_1, \dots, a_n)$ eine Basis in $\mathbb{R}^n$, $A = QR$ mit $Q = (q_1, \dots, q_n)$ orthogonal und $R = (r_1, \dots, r_n)$ obere Dreiecksmatrix, wobei $\det(R) \neq 0$. Dann bilden die Vektoren $q_1, \dots, q_n$ eine orthonormale Basis von $\mathbb{R}^n$ mit $\text{span}\{q_1, \dots, q_n\} = \text{span}\{a_1, \dots, a_n\}$.
Orthogonale Matrix:
$Q^{-1} = Q^T$ (ebenfalls orthogonal), $QQ^T = I$, $|\det(Q)| = 1$, $\|Q\|_2 = 1$, $\text{cond}(Q) = 1$, Zeilen und Spalten von Q sind paarweise orthogonal und normiert.
Implementierung Gram-Schmidt: schlecht

```
function [Q,R] = gramSchmidt(A)
  n = size(A,1); Q = zeros(n,n); R = zeros(n,n);
  for j=1:n
    v = A(:,j);
    for i=1:j-1
      R(i,j) = Q(:,i)' * A(:,j);   % R(i,j) = Q(:,i)' * v; wäre besser
      v = v - R(i,j) * Q(:,i);
    end
  R(j,j) = norm(v, 2); Q(:,j) = v / R(j,j); end
```

Implementierung modified Gram-Schmidt: besser

```
function [Q,R] = gramSchmidtMod(A)
  n = size(A,1); Q = zeros(n,n); R = zeros(n,n); V = zeros(n,n);
  for i=1:n
    V(:, i) = A(:, i); end
  for i=1:n
    R(i,i) = norm(V(:,i), 2); Q(:,i) = V(:,i) / R(i,i);
    for j=i+1:n
      R(i,j) = Q(:,i)' * V(:,j); V(:,j) = V(:,j) - R(i,j) * Q(:,i); end; end
```

MIT HOUSE-HOLDER-TRANSFORMATIONEN

qr von Matlab ist so implementiert. Ist genauer. bestes.

METHODE DER KLEINSTEN QUADRATE

Approximation der Messreihe y_i durch Modellfunktion b_i, so dass der mittlere quadratische Fehler minimal wird.
Modellfunktion: $f(t) = \sum_{i=1}^n x_i b_i(t)$
Ziel: Finde Koeff. c_i, so dass $\sum_{j=1}^m |f(t_i) - y_i|^2$ minimal.

$$r = Ax - y, \qquad A = \begin{bmatrix} b_1(t_1) & \cdots & b_n(t_1) \\ \vdots & \ddots & \vdots \\ b_1(t_n) & \cdots & b_n(t_n) \end{bmatrix}$$

Normalengleichungen:
Minimiert mittleren quadr. Fehler: $\|r\| = \|Ax - y\| \rightarrow \min!$
Minimum c^* erhält man durch Lösung von $A^TAc = A^Ty$ (N.-Gl.)
Implementierung:
Aufgabenstellung: $i(t) = \sum_{k=1}^3 c_k e^{-ta_k}$, $c_k = ?$

```
% gegebene Daten:
a = [0.25, 0.5, 0.75]; tI = [0:10]'; iI = [0, 8, 12, 13, 13, 12, 10, 8, 7, 6, 5]';
A = exp(-tI*a); % Datenmatrix
% LÖSUNG VIA QR-ZERLEGUNG:
resultat = A\iI;
% LÖSUNG VIA QR-ZERLEGUNG VON HAND:
[Q, R] = qr(A); R0 = R(1:3,1:3); d = Q'*iI; resultat = R0\(d(1:3));
% LÖSUNG VIA NORMALENGLEICHUNGEN:  A'*A*alpha = A'*y
[L, U, P] = lu(A'*A);
resultat = U\(L\P*(A'*iI)); % Loesung der Normalgleichung
```

was ist mit $\Phi(x)$???

KONDITIONSZAHL

Konditionszahl: Beschreibt die maximal Verstärkung, mit der sich eine relative Störung von y auf c auswirkt.
Sei A eine $m \times n$-Matrix mit $m \geq n$, dann:

$$\text{cond}(A) \coloneqq \kappa(A) \coloneqq \|A\|\|A^{-1}\|$$

$$\text{cond}_2(A) = \sqrt{\frac{\lambda_{max}(A^TA)}{\lambda_{min}(A^TA)}} = \frac{\sigma_{max}}{\sigma_{min}}$$

Falls $A = A^T$: $\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}}$
Es gilt für $A = QR$: $\text{cond}(A) = \text{cond}(R) = \sqrt{\text{cond}(A^TA)}$
da stimmt noch was nicht

SINGULÄRWERTZERLEGUNG

Singulärwertzerlegung:
Zerlegung von $A \in \mathbb{R}^{m \times n}$, so dass $A = U\Sigma V^T$ mit orthogonalen Matrizen $U \in \mathbb{R}^{m \times m}$ und $V \in \mathbb{R}^{n \times n}$. Matrix der Singulärwerte $\Sigma \in \mathbb{R}^{m \times n}$ (gleiche Dimension wie A).
Singulärwerte:

$$\sigma_i = \sqrt{\lambda_i(A^TA)} = \sqrt{\lambda_i(AA^T)} \geq 0 \ \forall i, \qquad \#\sigma_i > 0 = \text{Rang}(A)$$

Residuum für LSQ:

$$\|r\|_2^2 = \|Ax - b\|_2^2 = \|U^T(Ax - b)\|_2^2 = \|U^TAx - U^Tb\|_2^2$$
$$= \|U^TU\Sigma V^Tx - U^Tb\|_2^2 = \|\Sigma z - c\|_2^2$$

mit $z = V^Tx$ und $c = U^Tb$. Zusammengefasst:

$$\begin{array}{ccccc} \mathbf{x} \in \mathbb{R}^2 & & \xrightarrow{\quad \mathbf{A} = \mathbf{U\Sigma V}^T \quad} & & \mathbf{b} \in \mathbb{R}^3 \quad \text{(Koordinaten bzgl. alter Basis)} \\ \mathbf{V}^T \downarrow \uparrow \mathbf{V} & & & & \mathbf{U}^T \downarrow \uparrow \mathbf{U} \quad \text{(orthogonale Koordinatenabbildungen)} \\ \mathbf{z} \in \mathbb{R}^2 & & \xrightarrow{\quad \mathbf{\Sigma} \quad} & & \mathbf{c} \in \mathbb{R}^3 \quad \text{(Koordinaten bzgl. neuer Basis)} \end{array}$$

Die optimale Lösung $z^* = \arg\min_z \|\Sigma z - c\|^2$ ist leicht zu berechnen, weil die Gleichungen nun entkoppelt sind.

$$\Sigma = \begin{bmatrix} \Sigma_r & O_{r \times (n-r)} \\ O_{(m-r) \times r} & O_{(m-r) \times (n-r)} \end{bmatrix}, \qquad \Sigma_r = \text{diag}(\sigma_1, \dots, \sigma_r)$$

GAUSS-NEWTON-VERFAHREN

Grundidee:
Residuenvektor linearisieren: $\vec{r}(\vec{x}) \approx \vec{r}(\vec{x}_0) + D\vec{r}(\vec{x}_0)(\vec{x} - \vec{x}_0)$

$$\arg\min\|\vec{r}\| = \arg\min \left\| \vec{r}(\vec{x}_0) + D \cdot \vec{r}(\vec{x}_0) \underbrace{(\vec{x} - \vec{x}_0)}_{\vec{s}} \right\| = \arg\min\|A\vec{x} - \vec{c}\|$$

$A = D \cdot \vec{r}(\vec{x}_0) = J_r(\vec{x}_0), \qquad \vec{c} = -\left(\vec{r}(\vec{x}_0) + D \cdot \vec{r}(\vec{x}_0)(\vec{x} - \vec{x}_0)\right)$
An einem Beispiel gezeigt:
Gegeben:
Zu minimierende Fehlerfunktion $\Phi(\vec{v}) = \Phi(m_x, m_y, r) =$

$$\sum_{i=1}^n \left(\sqrt{(x_i - m_x)^2 + (y_i - m_y)^2} - r\right)^2$$

Und n Datenpunkte $(x_1, y_1) = (4,3), \dots$
Gesucht: $\vec{v} = (m_x, m_y, r)$, so dass $\Phi(\vec{v})$ minimal wird.
1. Linearisieren des von $\vec{v}$ abh. Residuenvektors $\vec{R}$ in $w \in \mathbb{R}^3$
Dazu $\vec{v} = \vec{w} + \vec{s}$ für $s \in \mathbb{R}^3$ schreiben und Taylor in $\vec{w}$:

$$\vec{R}(x_i y_i, \vec{v}) = \begin{pmatrix} \sqrt{(x_1 - m_x)^2 + (y_1 - m_y)^2} - r \\ \sqrt{(x_2 - m_x)^2 + (y_2 - m_y)^2} - r \\ \vdots \end{pmatrix}$$

somit ist $\Phi(\vec{v}) = \|\vec{R}\|_2^2$ und wir erhalten die Jacobi-Matrix:

$$\vec{J}_R = \left(\frac{\partial R_i}{\partial v_j}\right) = \begin{bmatrix} \frac{-(x_1-m_x)}{\sqrt{(x_1-m_x)^2+(y_1-m_y)^2}} & \frac{-(y_1-m_y)}{\sqrt{(x_1-m_x)^2+(y_1-m_y)^2}} & -1 \\ \vdots & \vdots & \vdots \\ \frac{-(x_n-m_x)}{\sqrt{(x_n-m_x)^2+(y_n-m_y)^2}} & \frac{-(y_n-m_y)}{\sqrt{(x_n-m_x)^2+(y_n-m_y)^2}} & -1 \end{bmatrix}$$

Die Linearisierung von $\vec{R}(\vec{v})$ im Punkt $\vec{W} \in \mathbb{R}^3$ ist somit:

$\vec{R}_{lin}(\vec{w}+\vec{s}) = \vec{R}(\vec{w}) + J_R(\vec{w}) \cdot \vec{s}, \qquad \vec{v} = \vec{w} + \vec{s}, \qquad \vec{s} \in \mathbb{R}^3$

2. Lineares Least-Squares-Problem bestimmen:

$$\left\|\vec{R}_{lin}(\vec{v}_i + \vec{s})\right\|_2 = \left\|\vec{R}(\vec{v}_i) + J_R(\vec{v}_i) \cdot \vec{s}\right\|_2 \to \min!$$

Also haben wir in jedem Schritt obiges LSQ-Problem mit der Unbekannten $\vec{s}$, Matrix $J_R(\vec{v}_i)$ und „Messwerten“ $\vec{R}(\vec{v}_i)$.

Implementierung:

```
function[R]=R(x,y,v)
  mx=v(1); my=v(2); r=v(3); R=sqrt((x-mx).^2+(y-my).^2) - r; end
function[DR]=DR(x,y,v)
  mx=v(1); my=v(2); r=v(3); r=sqrt((x-mx).^2+(y-my).^2);
  DR = zeros(length(x), 3); DR(:, 1)=-(x-mx) ./ r;
  DR(:, 2)=-(y-my) ./ r; DR(:, 3)=-ones(1,size(x,2)); end
function v=circleFit(X,v0)
  x=X(:, 1); y=X(:, 2); maxit=50; tol=1e-8; v = v0;
  for i=1:maxit
    delta=-DR(x,y,v) \ R(x,y,v); v=v+delta;
    if abs(delta)<tol*abs(v); break; end; end
```

EIGENWERTPROBLEME

MIT QR-ALGORITHMUS

Gegeben: $A \in \mathbb{R}^{n\times n}$, Gesucht: EW und EV von A

Algorithmus:

$A_0 = A, \qquad A_k = Q_k R_k, \qquad A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$

Der Schritt ist eine Ähnlichkeitstransformation, d.h. das charakteristische Polynom p_λ bleibt dasselbe.

Satz: Falls A nur Eigenwert von versch. Betrag hat, konvergiert die Matrix A_k gegen eine obere Dreiecksmatrix X. EW: $\mathrm{diag}(X)$ Bei betragsgleichen EW entstehen Kästchen auf der Diagonalen, die nicht konvergieren.

Begründung (Satz von Schur): Zu jeder Matrix $A \in \mathbb{R}^{n\times n}$ existiert eine orthogonale Matrix U, so dass:

$$U^T A U = \tilde{R} = \begin{bmatrix} \tilde{R}_{11} & \cdots & * \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{R}_{kk} \end{bmatrix}$$

Eindimensionale Blöcke $\tilde{R}_{ii}$ enthalten die reellen EW von A. Zweidimensionale Blöcke $\tilde{R}_{ii}$ enthalten die konj. kompl. EW, welche man aus dem charakt. Polynom des Blocks ermittelt.

Implementierung:

```
function ev = qrEigenvalues(A,nIter,tol)
 [m,n] = size(A);
if m ~= n; display('Matrix nicht quadratisch'); return; end
if m ==1; ev = A; return; end
for i = 1:nIter; [Q,R] = qr(A); A=R*Q; end % iteration
bOK = 1; i=1; % Pruefen, ob nahezu Dreiecksmatrix erzeugt
while (i<=m && bOK==1); j=1;
  while (j<=i-1 && bOK==1);
    if abs(A(i,j)) > tol; bOK = 0; end;
    j=j+1; end;
  i=i+1; end
% Ergebnis in Rueckgabevariable schreiben
if bOK==1; ev = diag(A); else; ev=[]; display('Fehler'); end
```

KRYLOV-RAUM

Krylov-Unterraum:

$$\mathcal{K}_n(A,b) = \mathrm{span}\{b, Ab, A^2b, \dots, A^{n-1}b\}$$

Power Iteration: Man kann den Eigenvektor zum grössten EW von $A \in \mathbb{R}^{m\times m}$ finden, indem man einen Startvektor v immer wieder durch A abbildet. Die eizelnen Iterationsschritte kann man als Krylov-Matrix schreiben:

$$\mathcal{K}_n = [v \quad Av \quad A^2v \quad \dots \quad A^{n-1}v]$$

Die Powerniteration spannt also einen Krylov-Unterraum auf, wobei in der letzten Spalte der approx. grösste Eigenvektor ist. Problem: Spaltenvektoren nicht orthogonal → Iteration instabil.

Arnoldi: Problem eliminieren durch Aufspannen des Unterraums durch (mit G-S) orthogonalisierte Vektoren $v_1 \dots v_n$

Hessenberg-Matrix: Die Hessenbergmx. H_n wird verwendet um A auf den Krylov-Unterraum zu projizieren. Zerlegung von H:

$$H_n = V^H A V, \qquad V = [v_1 \quad \dots \quad v_n]$$

Die EW der Hessenbergmatrix werden Ritz-Werte genannt. Die Hoffnung ist, dass diese zu einigen EW von A konvergieren. Man kann abe rnicht alle EW bestimmen, weil:

$$H_n \in \mathbb{R}^{n\times n}, \qquad A \in \mathbb{R}^{m\times m}, \qquad n < m$$

Symmetrie: $A = A^T$. Dann $\exists U$ orthonormal : $A = UDU^T$

ARNOLDI-VERFAHREN

Gegeben: $A \in \mathbb{R}^{n\times n}$, bel. Startvektor v_0, Toleranz tol, #EW: k
Gesucht: k grösste EW von $Ax = \lambda x$

Algorithm 1: Arnoldi-Verfahren

for $\ell = 2, 3, 4, \dots$ **do**
  $\boldsymbol{v}_\ell = \mathbf{A}\boldsymbol{v}_{\ell-1}$
  for $j = 1, 2, \dots, \ell-1$ **do**
    $h_{j,\ell-1} = \boldsymbol{v}_j^{\mathrm{H}} \boldsymbol{v}_\ell$
    $\boldsymbol{v}_\ell = \boldsymbol{v}_\ell - h_{j,\ell-1}\boldsymbol{v}_j$
  compute eigenvalues λ of $\boldsymbol{H}$ (the *Ritz values*)
  if *requested number of eigenvalues are converged* **then return**
  $h_{\ell,\ell-1} = \|\boldsymbol{v}_\ell\|$
  $\boldsymbol{v}_\ell = \boldsymbol{v}_\ell / h_{\ell,\ell-1}$

Implementierung:

```
function [ DN , V , Ht ] = arnoldieig(A, v0, k, tol)
dotM = @(u, v) u' * v; normM = @(u) sqrt( dotM(u, u) );
applyA = @(u) A*u; ritzToEV = @(ev) ev;
n = size (A, 1); V = [ v0 / normM(v0) ];
Ht = zeros (1, 0); DN = zeros (k , 1 ); dn = zeros (k , 1 );
for l=1:n
  d = dn; Ht = [Ht, zeros(l , 1 ); zeros(1 , l) ];
  vt = applyA( V(:, l) ); % next Krylov vector
  for j =1:l  % Gram-Schmidt-Orthogonalization
    Ht(j, l) = dotM(V(:, j), vt); vt = vt - Ht(j , l ) *V(:, j); end
  % solve EVP for the small Hessenberg Matrix
  [VV, D] = eig(Ht(1:l, 1:l)); ev = diag(D);
  [abs_ev, i_ev] = sort(abs(ev), 'descend'); ev = ev(i_ev);
  dn(1:min(l, k)) = ritzToEV( ev(1:min(l , k)) ); DN(:,l) = dn;
  if (norm(d-dn) < tol * norm(dn)), break ; end; % konvergiert?
  if l < n  %normalization of vt
    Ht(l+1, l) = normM(vt); V = [V, vt / Ht(l +1 , l)]; end; end
V = V * VV; V = V(:, i_ev);
```

Implementierung für kleinste EW: Erste 3 Zeilen ersetzen:

```
function [DN, V, Ht] = arnoldieig_sm(A, v0, k, tol)
dotM = @(u, v) u' * v; normM = @(u) sqrt( dotM(u, u) );
applyA = @(u) A \ u; ritzToEV = @(ev) 1./ev;
```

Für generalisierte Eigenwertprobleme: $Ax = \lambda Bx$

Gesucht: EW von $\hat{A} = B^{-1}A$ (nicht symmetrisch)
Gegeben: A, B (beide symmetrisch), B auch positiv definit
daher existiert die ***Cholesky-Zerlegung*** (LR-Zerlegung für symm. Matrizen) $B = LL^T$ mit L einer links unteren Dreiecksmatrix.
Daher haben wir: $\tilde{A}y = \lambda y, \ \tilde{A} = L^{-1}AL^{-T}, \ y = L^T x$
Wir verwenden den Algorithmus mit $\tilde{A}$ statt A und $\tilde{v}_i = L^{-T}v_i$
$v_i = Av_{i-1} \Rightarrow L^T\tilde{v}_i = L^{-1}AL^{-T}L^T\tilde{v}_{i-1} \Leftrightarrow \tilde{v}_i = L^{-T}L^{-1}A\tilde{v}_{i-1} = B^{-1}A\tilde{v}_{i-1}$

Algorithm 1: Arnoldi-Verfahren für generalisierte EWPs

for $\ell = 2, 3, 4, \dots$ **do**
  $\tilde{\boldsymbol{v}}_\ell = B^{-1}A\tilde{\boldsymbol{v}}_{\ell-1}$
  for $j = 1, 2, \dots, \ell-1$ **do**
    $h_{j,\ell-1} = \tilde{\boldsymbol{v}}_j^{\mathrm{H}} B \tilde{\boldsymbol{v}}_\ell$
    $\tilde{\boldsymbol{v}}_\ell = \tilde{\boldsymbol{v}}_\ell - h_{j,\ell-1}\tilde{\boldsymbol{v}}_j$
  compute eigenvalues λ of $\boldsymbol{H}$ (the *Ritz values*)
  if *requested number of eigenvalues are converged* **then return**
  $h_{\ell,\ell-1} = \tilde{\boldsymbol{v}}_\ell^{\mathrm{H}} B \tilde{\boldsymbol{v}}_\ell$
  $\tilde{\boldsymbol{v}}_\ell = \tilde{\boldsymbol{v}}_\ell / h_{\ell,\ell-1}$

Hinweis: Nächster EW zu $1/3$ = kleinster EW von $\left(A - I\frac{1}{3}\right)$

LANCZOS-PROZESS

ITERATIVE VERFAHREN FÜR LGS

STEEPEST DESCENT

Positiv definite Matrix generieren: $\tilde{A} = (-\lambda_{min} + \delta)I_n + A$
Symmetrische Matrix generieren: $\tilde{A} = A + A^T$

CONJUGATE GRADIENTS

Abstiegsverfahren zur Minimierung konvexer, quadratischer Funktionen. Ist ein Krylov-Unterraum-Verfahren. Instabil bei Rundungsfehlern → Präkonditionierung nötig.

Grundidee: Lösen des LGS $Ax + b = 0$, wobei $A = A^T \in \mathbb{R}^{n\times n}$ und $x^TAx > 0 \Leftrightarrow$ Minimierung von quadratischer Funktion $f(x) = \frac{1}{2}x^TAx + bx + c$, weil es soll: $\nabla f(x) = 0 = Ax + b$

Konjugierte Richtungen:
$v, w \in \mathbb{R}^n$ heissen A-konjugiert, falls $v^TAw = 0$

Algorithmus: Gegeben: A symm. pos. definit, Startvektor x_0
Start: x_0; $r_0 = Ax_0 + b$; $p_1 = -r_0$ (Gradientrichtung)
Schritt k: $k = 1,2, \dots$
$e_{k-1} = \frac{\langle r_{k-1}, r_{k-1}\rangle}{\langle r_{k-2}, r_{k-2}\rangle}$, $k \geq 2$ (Faktor, der konj. Richtung bestimmt)
$p_k = -r_{k-1} + e_{k-1}p_{k-1}$, $k \geq 2$ (konj. Richtung wird best.)
$\varrho_k = \frac{\langle r_{k-1}, r_{k-1}\rangle}{\langle p_k, Ap_k\rangle}$ (Faktor, der die Schrittgrösse bestimmt)
$x_k = x_{k-1} + \varrho_k p_k$ (neues „Minimum“)
$r_k = r_{k-1} + \varrho_k Ap_k$ (nächste Gradientrichtung)
Abbruchbedingung: $\|r_k\| \leq \|r_0\| \cdot tol$

Implementierung:

```
function [x, k] = cg(A, b, tol)
  r=-b; % Anfangsresiduum
  x=zeros(length(b),1); % Startwert
  p=zeros(length(b),1);
  rTr=r'*r; % rTr = r^T r (in step k+1)
  for k=1:Inf
    if k==1; sigma=0;
    else; sigma=rTr/rTr_old; end
  rTr_old = rTr; % rTr_old = r^T r (in step k)
  p = sigma*p - r; A_rho = A*p;
  rho = rTr/(p'*A_rho); x = x + rho*p; % x (in step k+1)
  r = r + rho*A_rho;% r (in step k+1)
  %r = A*x - b; % real residuum
  rTr = r'*r; % r^T r (in step k+1)
  if (rTr < (tol)^2); break; end; end
```

DFT

DFT

DFT: Berechnet die Abtastwerte des Spektrums eines zeitdiskreten Signals.

$$\hat{x}_l = \sum_{k=0}^{n-1} x_k e^{-i\frac{2\pi kl}{n}}, \qquad l = 0, \dots, n-1$$

DFT detailierter:

$$F[n] = \sum_{k=0}^{N-1} f[k]\omega^{kn} = \sum_{k=0}^{N-1} f[k]e^{-i\frac{2\pi}{N}nk}, \qquad \omega = e^{-i\frac{2\pi}{N}}$$

als Transformationsmatrix:

$$\begin{bmatrix} F[0] \\ F[1] \\ \vdots \\ F[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \dots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \dots & \omega^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ f[N-1] \end{bmatrix}$$

N: Anzahl Abtastwerte. Die Matrix $\mathcal{F}_6$ z.B. ise eine 6×6-Matrix und kann für 6 Abtastwerte verwendet werden.

Zerlegung & Rekursion: Damit ergibt sich dann $O(N \log N)$
Problem kann aufgespaltet werden. Wenn $N \geq 2$ und gerade, dann gilt: $P_6^T\mathcal{F}_5 = \begin{bmatrix} \mathcal{F}_3 & \mathcal{F}_3 \\ \mathcal{F}_3\Omega_3 & -\mathcal{F}_3\Omega_3 \end{bmatrix} = \begin{bmatrix} \mathcal{F}_3 & 0 \\ 0 & \mathcal{F}_3 \end{bmatrix} \begin{bmatrix} I_3 & I_3 \\ \Omega_3 & -\Omega_3 \end{bmatrix}$
P_N bezeichnet die Permutationsmatrix. Jetzt können wir die DFT eines Vektors $y = [y_1 \quad y_2]^T$ berechnen:

$$\mathcal{F}_6 y = P_6 \begin{bmatrix} \mathcal{F}_3 & 0 \\ 0 & \mathcal{F}_3 \end{bmatrix} \begin{bmatrix} I_3 & I_3 \\ \Omega_3 & -\Omega_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_6 \begin{bmatrix} \mathcal{F}_3(y_1 + y_2) \\ \mathcal{F}_3\Omega_3(y_1 - y_2) \end{bmatrix}$$

Implementierung:

blablabla

ZYKLISCHE FALTUNG MIT FFT

Die zykl. Faltung entspricht einer Faltung mit period. Signal.

$$z_k = \sum_{j=0}^{n-1} x_{(k-j)} y_j \Leftrightarrow \hat{z}_l = \hat{x}_l \cdot_{\text{element-wise}} \hat{y}_l, \qquad l = 0, \dots, n-1$$

wobei alle *Indizes modulo n* zu verstehen sind.

Implementierung:

```
function[z]=fft_conv(x,y)
  z=ifft(fft(x).*fft(y));
```

LINEARE FALTUNG

Anwendung: Polynomprodukt $p(x) \cdot q(x)$
mit $p(x) = \sum_{i=0}^{n-1} a_i x^i$ und $q(x) = \sum_{j=0}^{n-1} b_j x^j$
$p(x) \cdot q(x) = \sum_{k=0}^{2n-2} c_k x^k$ mit $c_k = \sum_{j=0}^{k} a_k b_{k-j}$

Direkte Implementierung:

```
function z = polprod_direct(x,y)
  N = length(x); z = zeros(1, 2*N-1);
  for n = 1 : (2*N-1)
    for k= max(0, n-N) : min(n-1, N-1)
      z(n) = z(n) + x(k+1).*y(n-k); end; end;
```

Implementierung mit FFT: mit Zero-Padding

```
function z = polprod_fft(x,y)
```

```
add = zeros(size(x,1),size(x,2)); x = [add x]; y = [add y];
z = ifft(fft(x).*fft(y)); z = z(1:end-1);
```

ZIRKULANTE MATRIX

Zirkulante Matrix:
$A \in \mathbb{R}^{n\times n}$, deren Spalten bzw. Zeilen aus zykl. Permutationen der ersten Spalte/Zeile entwickelt werden. In einer Diag. von links oben nach rechts unten stehen immer dieselben Werte
Lösen eines LGS mit zirkulanter Matrix mittels FFT:
Sei das Gleichungssystem $Au = b$ mit $A \in \mathbb{R}^{n\times n}$
u = ifft(fft(b) ./ fft(c)); % wobei c die erste Spalte von A

INTERPOLATION

Gegeben: Wertepaare (x_i, f_i) mit den Stützstellen x_i und Funktionswerte (Stützwerte) $f_i = f(x_i)$
Gesucht: $P_n(x) \approx f(x)$, n: Grad des Polynoms
Grad des Polynoms: $n+1$ Wertepaare für Polynom von Grad n und $P_n(x_i) = f_i$

LAGRANGE-INTERPOLATION

Idee: $P_n(x_i) = f_i \leftrightarrow P_n(x_i) = f_i\delta_{i,j}$ wobei $\delta_{i,j} = \begin{cases}1, wenn\ i = i\\ 0, sonst\end{cases}$
$\delta_{i,j}$ erzeugen mit: $l_i = c(x-x_1)\cdot\ldots\cdot(x-x_n)$ ohne $(x-x_i)$

$$l_i(x_i) = 1 \to c = \frac{1}{(x_i-x_1)\cdot\ldots\cdot(x_i-x_n)}$$

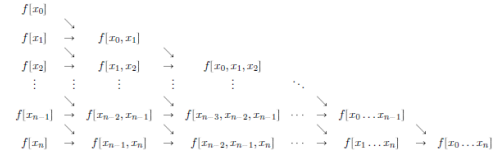
$$P_n(x) = \sum_{i=0}^{n} f_i l_i(x), \qquad l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$$

NEWTON-ALGORITHMUS

Für effizientere Auswertung der Lagrange-Interpolation
Idee: Darstellung von $P_n(x)$ durch Newton-Basisfunktionen:
$N_0(x) = 1,\ N_i(x) = \prod_{j=0}^{i-1}(x-x_j) = (x-x_o)\cdot\ldots\cdot(x-x_{i-1})$
$P_n(x) = \sum_{i=0}^{n} c_i N_i(x)$
$\quad = c_0 + c_1(x-x_0) + \cdots + c_n(x-x_0)\cdot\ldots\cdot(x-x_{n-1})$
$P(x_i) = f_i \to P\vec{c} = \vec{f}$, also:

$$\begin{bmatrix}1 & 0 & \cdots & 0\\ 1 & (x_1-x_0) & 0 & \vdots\\ \vdots & \vdots & \ddots & 0\\ 1 & (x_n-x_0) & \cdots & \prod_{i=0}^{n-1}(x_n-x_i)\end{bmatrix}\begin{bmatrix}c_0\\ \vdots\\ c_n\end{bmatrix} = \begin{bmatrix}f_o\\ \vdots\\ f_n\end{bmatrix}$$

$\vec{c}$ mit Schema der dividierten Differenzen bestimmen $\leftarrow c_i$ liegen auf der obersten Diagonalen.
$c_i := f[x_0,\ldots,x_i],\ f[x_i] := f_i$
Rekursive Definition: $f[x_i,\ldots,x_j] := \frac{f[x_{i+1},\ldots,x_j]-f[x_i,\ldots,x_{j-1}]}{x_j-x_i}$



CHEBYCHEV-STÜTZSTELLEN

Chebychev-Polynom: $T_n(x) = \cos(n\arccos(t)), -1 \le t \le 1$
Rekursion: $T_n(t) = 2tT_{n-1}(t) + T_{n-2}(t),\ T_0(T) = 1, T_1(t) = t$
Nullstellen: $t_k = \cos\left(\frac{2k-1}{2n}\pi\right),\ k = 1,\ldots,n$

Stützstellen: Für eine Interpolation mit Polynom n-ter Ord. braucht man $n+1$ Stützstellen. Für $I = [-1,1]$:

$$t_k = \cos\left(\frac{2k+1}{2(n+1)}\pi\right), \qquad k = 0,\ldots,n$$

Für allgemeine Intervalle $I = [a,b]$, ergibt sich

$$\hat{t}_k = a + \frac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right)+1\right), \qquad k = 0,\ldots,n$$

TRIGONOM. (DFT) INTERPOLATION

Fourieranalyse: Periodische Funktion $f(x) = f(x+p)$

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{k2\pi}{p}x}, \qquad c_k = \frac{1}{p}\int_0^p f(x)e^{-i\frac{k2\pi}{p}x}dx \in \mathbb{C}$$

Übergang DFT: Periode $p = 2\pi$, f gegeben durch Stützstellen $x_k = \frac{k2\pi}{N}$ und Stützwerte $f(x_k), k = 0,1,\ldots,N-1$
Fourier-Koeff.-Approx. mit Trapezregel:

$$\hat{c}_n = \frac{1}{N}\sum_{k=0}^{N-1} f(x_k)e^{-in\frac{k2\pi}{N}}$$

Matrixform: Analyse: $\hat{c} = \frac{1}{N}\mathcal{F}_N f,\ \mathcal{F}_N$: Fouriermatrix
Synthese: $f = \overline{\mathcal{F}}_N\hat{c}$ wegen $\mathcal{F}_N^{-1} = \frac{1}{N}\overline{\mathcal{F}}_N$
Interpolation: Synthese: $f_k = \sum_{n=0}^{N-1}\hat{c}_n e^{in\frac{k2\pi}{N}}$
Durch Umstellen der Reihe erhält man das trigonom. Polynom:

$$p(x) = \begin{cases}\sum_{n=-N/2+1}^{N/2-1}\hat{c}_n e^{inx} + \hat{c}_{N/2}\cos\left(\frac{N}{2}x\right), & N\ gerade\\ \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}}\hat{c}_n e^{inx}, & N\ ungerade\end{cases}$$

Es gilt immer: $p(x_k) = f(x_k) = f_k$
Fouriermatrix: N-te Einheitswurzel: $\omega = e^{-i\frac{2\pi}{N}}$

$$\mathcal{F}_N = \begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & \omega & \cdots & \omega^{N-1}\\ \vdots & \vdots & \ddots & \vdots\\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)(N-1)}\end{bmatrix}$$

serie 9 A 2 geht's weiter

QUADRATUR

Zusammengesetzt: Man kann jeweils die Funktion in zwei Bereiche aufteilen und die Bereiche einzeln Integrieren.
Quadraturformel:

$$Q_{[a,b]}^{(n)}[f] = \sum_{j=0}^{n}\alpha_j f(x_j)$$

α_j: Gewichtungsfaktoren der Quadratur
Newton-Cotes: Je nachdem ob die Stützstellen die Intervallenden enthalten, unterscheidet man zwischen offenen und geshclossenen Quadraturformeln.
- geschlossen: Stützstellenwahl $a \le x_0 < \cdots < x_n \le b$ mit $x_j = a + jh$, wobei $j = 0,\ldots,n$ und $h = \frac{b-a}{n}$
- offen: Stützstellenwahl $a < x_0 < \cdots < x_n < b$ mit $x_j = a + (j+1)h$, wobei $j = 0,\ldots,n$ und $h = \frac{b-a}{n+2}$

Berechnung der Gewichte für Newton-Cotes (hier für offen):

$$\alpha_j = h\int_a^b l_j(x)dx \xrightarrow{x\to t=\frac{x-a}{h}} h\int_0^n \prod_{\substack{i=0\\ i\neq j}}^{n}\frac{t-i}{j-i}$$ d?

Quadraturordnung: Die Ordnung einer Quadratur gibt an, welches Polynom derselben Ordnung mit der Quadraturformel nicht mehr exakt integriert wird. Definition: Eine Quadraturformel $Q_{[a,b]}$ hat Ordnung $n+1$, wenn gilt:

$$I_{[a,b]}[p_n] = Q_{[a,b]}[p_n], \qquad p_n \in \mathcal{P}_n$$

MITTELPUNKTREGEL

Offene Quadraturformel für $n = 0$

$$Q_{[a,b]}^{(0)}[f] = (b-a)f\left(\frac{a+b}{2}\right)$$

TRAPEZREGEL

geschlossene Quadraturformel für $n = 1$

$$Q_{[a,b]}^{(1)}[f] = \frac{b-a}{2}(f(a)+f(b))$$

Quadratische Konvergenz

SIMPSON-REGEL

Geschlossene Quadraturformel für $n = 2$:

$$Q_{[a,b]}^{(2)}[f] = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Quadratische Konvergenz

SUMMIERTE QUADRATUR

Das Intervall $I = [a,b]$ in N Teilintervalle aufteilen.

$$x_j = a + jh, \qquad j = 0,\ldots,n, \qquad h = \frac{b-a}{N}$$

$$Q_h^{(n)}[f] = \sum_{j=0}^{N-1} Q_{[x_j,x_{j+1}]}^{(n)}[f]$$

GAUSS-QUADRATUR

Geeignete Stützstellenwahl erhöht Ordnung. Eine $n+1$-Punkt Grauss-Quadratur hat Ordnung $2n+2$. Also wird ein Polynom der Ordnung $2n+1$ exakt integriert. Allg. Quadraturformel:

$$Q_{[a,b]}^{(n)}[f] = \Sigma_{j=0}^{n}\alpha_j f(x_j)$$

CLENSHAW-CURTIS

Variablentrafo $x = \cos(\theta)$ angewandt auf $f(x)$ führt zu:

$$\int_{-1}^{1} f(x)dx = \int_0^{\pi} f(\cos(\theta))\sin(\theta)\,d\theta = \sum_{k\ gerade}\frac{2a_k}{1-k^2}$$

wobei a_k die Fourier-Koeffizienten von $f(\cos(\theta)) \to$ FFT.
Transformation auf Interval $[-1,1]$:

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)dx$$

code.

GEWÖHNLICHE DIFFERENTIALGLEICHUNGEN

Taylor: $y(t+h) = y(t) + y'(t)h + \frac{y''(t)}{2!}h^2 + \cdots + \frac{y^{(n)}(t)}{n!}h^n$

EULER-VERFAHREN

$y(t+h)$ durch $y(t) + h\dot{y}(t) = y(t) + hf(y(t),t)$approx.
Euler explizit:
$y_{k+1} = y_k + h_k f(y_k,t_k),\ h_k = t_{k+1} - t_k,\ k = 0,1,\ldots,N-1$
Euler implizit:
$y_{k+1} = y_k + h_k f(y_{k+1},t_{k+1}), \qquad k = 0,1,\ldots,N-1$
Schrittweite: $h_k = t_{k+1} - t_k$

MITTELPUNKT-VERFAHREN

$$y_{k+1} = y_k + h_k f\left(\frac{1}{2}(y_k + y_{k+1}), \frac{1}{2}(t_k + t_{k+1})\right)$$

Schrittweite: $h_k = t_{k+1} - t_k,\ k = 0,1,\ldots,N-1$

MATRIX-EXPONENTIAL

$$e^{At} = Te^{\Lambda t}T^{-1}, \qquad e^{\Lambda t} = \begin{bmatrix}e^{\lambda_1 t} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & e^{\lambda_n t}\end{bmatrix}$$

$$e^{At} = I + At + \frac{A^2t^2}{2!} + \cdots + \frac{A^k t^k}{k!} + \cdots$$

MATRIX-EXPONENTIAL MIT KRYLOV

blablabla

RUNGE-KUTTA

Gegeben: $y'(t) = f(t,y(t)),\ y(t_0) = y_0$
Gesucht: $y(t_1) = y_0 + \int_{t_0}^{t_1} f(\tau, y(t_0+\tau))d\tau$
Runge-Kutta: Approximation des Integrals mit s-Punkt Quadratur mit Knoten c_i und Gewichten b_i mit $i = 1,\ldots,s$

$$y_1 \approx y_0 + h\sum_{j=1}^{i-1} b_i f(t_0 + c_i h, y_0 + c_i h)$$

Konvergenzordnung: s wenn expllizit, $2s$ wenn implizit
Butch-Tableau:

c_1 c_2 $\vdots$ c_s	A	$A \in \mathbb{R}^{s\times s}$: Runge-Kutta-Matrix $b \in \mathbb{R}^s$: Gewichte $c \in \mathbb{R}^s$: Knoten Es gilt $\sum_{i=1}^{s} b_i = 1$ und $c_i = \sum_{j=0}^{s} a_{ij}$
	b_1 b_2 … b_s	

Explizit: A strikt untere Dreiecksmatrix

$$k_i = f\left(t_0 + c_i h, y_0 + h\sum_{j=1}^{i-1} a_{ij}k_j\right),\ i = 1,\ldots,s$$
$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i$$

Semi-Implizit: A untere Dreiecksmatrix
Implizit: sonst
Allgemein:

$$k_i = f\left(t_0 + c_i h, y_o + h\sum_{j=1}^{s} a_{ij}k_j\right),\ i = 1,\ldots,s$$
$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i$$

Häufige Butcher-Tableaus:
klassisches RK:

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

$k_1 = f(t_0, y_0)$
$k_2 = f\left(t_0 + \frac{1}{2}h, y_0 + \frac{1}{2}hk_1\right)$
$k_3 = f\left(t_0 + \frac{1}{2}h, y_0 + \frac{1}{2}hk_2\right)$
$k_4 = f(t_0 + h, y_0 + hk_3)$

$$y_1 = y_0 + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)$$

VERSCHIEDENE SOLVER, STEIFE ODES

Steife ODE: Ein lineares DGL-System heisst steif, falls ein Eigenwert von A mit negativem Realteil existiert, so dass dieser Betrag sehr viel grösser ist als der Betrag des grössten Realteils der restl. EW. Bsp.: $\text{spec}(A) = \{-1000, -10{,}3\} \to 1000 \gg 3$
ode45: üblichster Solver, globale Fehlerord. 4, lokal 5.
ode15s: geeigneter für steife Prob., globale Fehlerord. 1, lokal 5

Polynomgrad, polyval, …, fftshift, taylor

ÜBUNG 2

THEORIE

Konvergiert, wenn: $\lim_{k\to\infty} x_k = \xi,\ I = [a,b]$
Banach-Fixpunktsatz:
1. g stetig auf I
2. g ist Selbstabbildung $g: I \to I,\ g(I) \in I$
3. g kontrahierend:
$$|g(y) - g(x)| \le L|y - x| \ \forall x, y \in I$$
3. (forts.) g kontrahierend, wenn $|g'(x)| \le L < 1$
Beispiel Selbstabb.: $x \to x^3,\ I = [0,1]$
Konvergenzordnung: Eine konverg. Seq. $x_k, k = 1,2, \dots$ in $\mathbb{R}$ konv. mit Ordnung p, wenn $\|x_{k+1} - \xi\| \le L|x_k - \xi|^p \ \forall k \in \mathbb{N}$
$e_{k+1} \approx Ce_k^p$, „Anzahl Nullen werden jeden Schritt verdoppelt“
→ **lineare Konvergenz**: $L < 1$ wenn $p = 1$
Beispiel Konv.: $e_k = 0{,}002 \dots, p = 2, \to e_{k+1} 0{,}000032 \dots$
Newton: ξ ist NS von f, dann $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},\ f'(x_k) \neq 0$
Sekantenverfahren: $x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$.
Startpunkte dazu: Ein Pt. li, einer re der NS
Relativer Fehler: $\frac{|h_k - h_{k-1}|}{|h_k|}$

TIPPS

Aufgabe 1:
A1a) 1: Pkt. angeben, 2: was darf a max. annehmen, b analog
A1c) ausprobieren, oder Fkt. die Banach nicht erfüllt
A1c) (forts.), Abstossen: $|g'(\xi)| > 1$, Anziehend: $|g'(\xi)| < 1$
Aufgabe 2:
A2b) $e_{k+1} \approx Ce_k^p,\ C = 1, \to e_{k+2} = (e_k^p)^p \to e_{k+n} = e_k^{(p^n)}$
A2c) $p = 1, C < 1$, was passiert mit C?
MATLAB-Aufgaben:
1. Kommentieren
2. Sinnvolle Variablennamen
3. help {command}
Aufgabe 3:
A3a) → Hinweis
A3b) 3 Stufen:
(1) Initialisierung ($x_0, \dots$)
(2) Iterationen `for i=bla:blu`
(3) Abbruch
`end`
Function Handles:
$$f = @(x_1, \dots, x_n)\ g(x_1, \dots, x_n), \qquad g: x \mapsto x^2$$
Hier: $f_{fix} = @(h, rho)\ iterationsfkt(h, rho)$
Fall rho geg.: $fct = @(h) f_{fix}(h, 0.4)$
Drei Function Handles: FP, Newt, Sek
`fprintf('%.4f  %.12f  %4d\n',rho,x(end),length(x)-1)`
A3c) $\rho \to 1$, Ist $\left|\frac{d}{dh}\ Iterationsfkt\right|$ kleiner/grösser/gleich 1?

ÜBUNG 3

Aufgabe 1:
A1a)
```
x=[-2:0.1:2];
plot(x,x^2,'r-,); xlabel(,x'); ylabel('f(x)'); title('..'); hold on; grid;
```
A1b) Grundgerüst Serie 2
```
f=@(x)f(x)
g=@(x)g(x)
xi=[x0 .....]; xi(end);
1e-4; 1e-10;
```
A1c) Plots anschauen, Tangente bei ξ, Nipp S.43
Aufgabe 2: Gedänpfte Newton-Iteration
-> für Pkt. die für sehr wenige x_0 konvergieren.
Ziel: Konvergenzbereich vergrössern
Slides Bsp.: 1.4.12
Korrektur beim normalen Newton: $\Delta x_k = \frac{f(x_k)}{f'(x_k)}$
Newton gedämpft: Abschwächung mit $x_{k+1} = x_k - \lambda_k \Delta x_k$
Natural Monotonicty Test: $\|\Delta x_{k+1}\| \le \left(1 - \frac{\lambda_k}{2}\right)\|\Delta x_k\|$
$$\Delta x_{k+1} = f'(x_{k+1})^{-1} f(x_{k+1}) = \|x_{k+2} - x_{k+1}\|$$
λ_k soweit reduzieren bis monotonicity test erfüllt.
⇒ Policy p10 heuristisch: $\lambda_{k+1} = \frac{\lambda_k}{2}$
A2a) $F(x) = \arctan(ax),\ x \in \mathbb{R},\ a > 0$
$a \in [0.1; 5], \qquad x_0 \in [-15,15]$
as = [0.1:0.2:5]; x0s = [-15:1:15]
Plotten:
for a = as for x0 = x0s ..... plot(x0,a) .... end end
A2b) [x conv] = dampedNewton(a,x0,maxIt,lambda_min)
Code 1.4.14, S. 96;
- LU-Zerlegung für Skalare nicht notwendig
- f und df: f = atan(a*x0); $\frac{df}{dx} = \frac{a}{1+(ax)^2}$
- Natural Monotonicty Test bei jedem Schritt machen!
- x1 = x0 – lambda deltaX
Aufgabe 3: Quasi-Newton-Verfahren
Erinnerung: $J(\vec{x})$ Jacobi-Matrix
Newton: $x_{k+1} = x_k - f(x_k)^{-1} f(x_k)$
Q-Newton: $\vec{x}_{k+1} = \vec{x}_k - J(\vec{x}_k)^{-1} \vec{f}(\vec{x}_k)$
Erinnerung: $\frac{df}{dx} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$
$\vec{J}(x) = \lim_{h\to 0}\left(\frac{1}{2h}\left(\vec{f}(\vec{x} + h\vec{e}_1), \dots, \vec{f}(\vec{x} + \vec{e}_n h)\right)\right)$, auf Blatt
A3a) nichts
A3b) i) Fkt. handles ii) → $[f_1; f_2; f_3] = [@(x)\ f_1(x_1, x_2, x_3)]$
- Einheits-Vektoren als Spalten der Einheitsmatrix
- iii) h ausreichend gleich wählen
- Berechnung von J: $J^{-1}\vec{f} = \Delta\vec{x} \to \vec{f} = J\Delta\vec{x}$ → dx = J \ f
- Abbruch: $\|\Delta x_k\| < tol$, matlab: norm
Copy apprixmation from exercise sheet onto summary

ÜBUNG 4

QR-Zerlegung: $A = QR$
- Q orthogonal: $Q^T = Q^{-1} \Rightarrow Q^T Q = I$
- $\|\vec{q}_i\|_2 = 1$
Orthonormalbasis von A mittels **Gram-Schimdt**:
$$Q = \left[\frac{\vec{a}_1}{\|\vec{a}_1\|} \quad \frac{\vec{a}_2 - \langle\vec{a}_2, \vec{a}_1\rangle \frac{1}{\|\vec{a}_1\|}}{\|\dots\|} \quad \dots\right]$$
$$R = \begin{bmatrix} \langle\vec{q}_1, \vec{a}_1\rangle & \langle\vec{q}_1, \vec{a}_2\rangle & \dots \\ 0 & \langle\vec{q}_2, \vec{a}_2\rangle & \dots \\ 0 & \vdots & \dots \end{bmatrix}$$
Kleinste Quadrate: $Ax = b$, überbestimmt
- Annahme: $f(t) = \sum_{i=1}^n c_i b_i(t)$ mit Messvektor $\vec{y}$
- Ziel: $\min \sum_{i=1}^n |f(t_i) - y_i|^2$
- Idee: LGS aufstellen: $\vec{y} = Ac, \vec{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \vec{y} = \underbrace{\begin{bmatrix} t_0^2 & 1 \\ t_1^2 & 1 \\ t_2^2 & 1 \end{bmatrix}}_{A} c$
- auflösen nach $\vec{c}$. damit ist $\vec{r} = \|\vec{y} - A\vec{c}\|_2^2$ bereits minimal.
Mit QR-Zerlegung: $z = Q^T b$, $Rx = z$, nach x auflösen
Normalengleichung: $Ac = y$
- $\|\vec{r}\|_2^2 = \|\vec{y} - A\vec{c}\|_2^2 =: \Phi(x)$
- Ziel: minimum $\Phi(\vec{c})$, grad $\Phi(\vec{c}) = 0$
$A^T\vec{y} = A^T A\vec{c}$ „**Normalengleichung**“
exp(A), cond(A)
Normalengleichung in Matlab: LR-Zerlegung
- [L,R,P]=lu(A'*A)
- $P(A^T A) = LR$
- cond($A^T A$)
Konditionszahl: $\text{cond}_2(A) = \sqrt{\frac{\lambda_{max}(A^T A)}{\lambda_{min}(A^T A)}}$
Ist die Konditionszahl $\xi = 10^D$ so ist mit einem Genauigkeitsverlust von D Dezimalstellen zu rechnen.
SVD: $A = U\Sigma V^T$
$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T \Sigma V^T$$
$$\text{cond}(\Sigma) = \frac{\sigma_{max}}{\sigma_{min}}$$

Matlab-Befehle:
- A^T: A'
- $\|A\|_2$: norm(A,2)
- Grösse A: size(A)

TIPPS

Aufgabe 1:
a) Pseudocode abtippen, $[Q\ R] = gS(A)$
- Init: R und Q präinitialisieren
- Iteration gemäss Pseudocode
$$\vec{v}_j = \vec{a}_j \leftarrow j\text{-ter Spaltenvektor}$$
b) $Z_{i,j} = \min(i,j),\ 1 \le i,j \le n$
- Matlab: $I = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix},\ J = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$, Z=min(I,J);
- → meshgrid, → repmat
c) $n = 50, Z = QR,\ e = \|QR - Z\|_2, Q^T Q = I$
Aufgabe 2:
a) $i_L(t) = \sum_{i=1}^3 c_i e^{-a_i t_1}$, a = [0.25 0.5 0.75]
LGS aufstellen: $\vec{i}_L = A\vec{c}$, c = A \ i_L
Matlab: t = 0:10'; t*a=[0*a; 1*a; ...; 10*a];
b) siehe theorie
c) was ist genauer? → cond(A)
Aufgabe 3:
a) $cond(A^T A) = ?$
b) $\|\vec{r}\|_2^2 = \left\|A\vec{x} - \vec{b}\right\|_2^2 = \|\Sigma\vec{z} - \vec{c}\|_2^2,\ \vec{z} = V^T\vec{x}$
Trafo: $U^T(A\vec{x} - \vec{b}),\ \|\vec{x}\|_2 = \|U^T\vec{x}\|_2$
- Für welche $\vec{z}^*$ wird $\|\vec{r}\|_2^2$ minimal?
- Wlche Komponenten tragen zur Minimierung bei?
$$\vec{z}^* = \text{argmin}_{\vec{z}}\|\Sigma\vec{z} - \vec{c}\|_2^2$$
$$\text{cond}(\Sigma_r) = \frac{\sigma_1}{\sigma_r}$$
c) Norm vom Residuum: $\|\vec{s}\|_2^2 = \|U^T\vec{r}\|_2^2,\ \|U\vec{x}\|_2^2 = \|\vec{r}\|_2^2$

SERIE 4

AUFGABE 1

$$A \in \mathbb{R}^{n\times n}$$
Algorithmus: $A_0 = A,\ A_k = Q_k R_k$
$$A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$$
$R = \begin{bmatrix} * & \dots & * \\ \vdots & \ddots & \vdots \\ 0 & \dots & * \end{bmatrix}$ mit EQ auf Diag. Existiert nur wenn alle Beträge der Eigenwerte verschieden sind.
Satz von Schur: $U^T A U = \tilde{R} = \begin{bmatrix} \tilde{R}_{11} & \dots & * \\ \vdots & \ddots & \vdots \\ 0 & \dots & \tilde{R}_{nn} \end{bmatrix}$
Tipps:
a) qr(A); tril(A,-1); A<tol; all(A)
b) eig(A)

AUFGABE 2

generalisiertes EW-Problem:
$$AX = \lambda Bx \Leftrightarrow B^{-1}Ax = \lambda x$$
Krylov: $\mathcal{K}_n = \text{span}\{\vec{b}, A\vec{b}, \dots, A^{n-1}\vec{b}\}$
Power Iteration: $\lambda_1,\ A \in \mathbb{R}^{n\times m}$, Startvec. $\vec{v}$
$$\mathcal{K}_n = \left[\vec{v}, A\vec{v}, \dots, \underbrace{A^{n-1}\vec{v}}_{grösster\ EV\ zum\ EW\ \lambda_n}\right]$$
Problem: Spaltenvek. lin. abh. → Instabilität
Arnoldi-Verfahren:
Krylov-Unterraum mit orthogonalisierten $\vec{v}$ aufspannen.
- Hessenbergmatrix: $H_n \in \mathbb{R}^{n\times n},\ n < m,\ H_n = V^H AV$
$\lambda_i(H_n)$: Ritz-Werte
Idee: Ritz-Werte → EW von A, p.211 im skript
Tipps: a) Pseudocode gut durchlesen (Code 4.3.3), Hinweis, Zeile 7 ändern, Zeile 12 etwas hinzufügen
b) $\hat{A} = B^{-1}A$. Plot: Fehler vs. h
Matlab: load(,file'); eigs(A,B,#EW,flags); repmat(…)
c) $B = LL^T,\ \hat{A}\vec{y} = \lambda\vec{y},\ \tilde{A} = L^{-1}AL^{-T},\ \vec{y} = L^T\vec{x}$
d) Implementierung: … $(A, B, \dots)$ …
e) Welchen Krylov-Raum haben wir benutzt?
→ Welche Matrix potentieren wir?
→ bzgl. welcher Matrix wird orthogonalisiert

AUFGABE 3

$$A(k)\vec{x} = \lambda M\vec{x}, \qquad A(k) = A_1 + ikA_2 + k^2 A_3$$

Tipps:
a) $B = A(1),\ i = 1i$ Timing: tic; ...; toc;
b) $k \in [0.01; \pi]$ -> plot

$$(k, \omega) = \left(k, \sqrt{\lambda}\right)$$

SERIE 8

A1: $F[n] = \sum_{k=0}^{N-1} f[k]\omega^{kn} = \sum_{k=0}^{N-1} e^{-i\frac{2\pi n}{N}k}$

$$\begin{bmatrix} F[0] \\ F[1] \\ \vdots \\ F[N-1] \end{bmatrix} = \begin{bmatrix} 1 & \dots & \dots & 1 \\ \vdots & \omega & \dots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \dots & \omega^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ f[N-1] \end{bmatrix}$$

$N =$ # Abtastwerte, $N \geq 2,\ \mathcal{F}_6, N = 6$

$$\Omega_3 = \begin{bmatrix} \omega^0 & . & . \\ . & \omega^1 & . \\ . & . & \omega^2 \end{bmatrix}$$

$$P_6^T \mathcal{F}_6 = \begin{bmatrix} \mathcal{F}_3 & 0 \\ 0 & \mathcal{F}_3 \end{bmatrix} \begin{bmatrix} I_3 & I_3 \\ \Omega_3 & -\Omega_3 \end{bmatrix} = \begin{bmatrix} \mathcal{F}_3 & \mathcal{F}_3 \\ \mathcal{F}_3\Omega_3 & -\mathcal{F}_3\Omega_3 \end{bmatrix}$$

Tipp: meshgrid

A2: Nipp p.67

$$P_n(x_i) = f(x_i) = f_i\delta_{ij}$$

$$l_i(x_i) = c(x - x_1)(x - x_2) \dots (x - x_n), \qquad l_i(x_i) = 1$$

$$\Rightarrow l_i(x) = \prod_{j=0}^{n} \frac{(x - x_j)}{x_i - x_j}, \qquad P_n(x) = \sum_{j=0}^{n} f_i \cdot l_i(x)$$

$$N_0(x) = 1, \qquad N_i(x) = \prod_{j=0}^{i-1} (x - x_j)$$

$$P_n(x) = \sum_{i=0}^{n} c_i N_i(x) = c_0 + c_1(x - x_0) + \cdots + c_n(x - x_0) \dots (x - x_{n-1})$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \vdots & (x_1 - x_0) & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \dots & \dots & \prod_{i=0}^{n-1}(x_n - x_i) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}$$

$$c_i = f[x_0, \dots, x_i], \qquad f[x_i] = f_i$$

$$f[x_i, \dots, x_j] = \frac{f[x_{i+1}, \dots, x_j] - f[x_i, \dots, x_{j-1}]}{x_j - x_i}$$

A3:

$$T_n(x) = T_n(\cos(\varphi)) = \cos(n\varphi)$$

a)
1. Stützstellen definieren
2. polyfit(x_stützstellen, f(x_stützst), n)
   polyval(c, x)

b)
1. Nullstellen berechnen von $\cos(n\varphi) \rightarrow \varphi$
2. Stützstellen $x_i = \cos(\varphi_i)$, $\varphi_i =$ NS aus 1

c)
mit Stützstellen von Tsch interpolieren, plotten, n variieren