Midterm Notes

$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ $\exp(-x)$ = plus minus ... => cancellation => better use $\frac{1}{e^x}$

Additionstheoreme

$\sin(x \pm y) = \sin(x)\cos(y) \pm \cos(x)\sin(y)$

$\cos(x \pm y) = \cos(x)\cos(y) \mp \sin(x)\sin(y)$

$\tan(x \pm y) = \frac{\sin(x \pm y)}{\cos(x \pm y)}$

Taylorreihe

$T_N(x, x_0) = \sum_{n=0}^{N} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$

Cancellation Examples

i) Given $f(x) := -\ln(x - \sqrt{x^2-1})$, $x > 1$

We know $(x - \sqrt{x^2-1}) = \frac{(x+\sqrt{x^2-1})(x-\sqrt{x^2-1})}{(x+\sqrt{x^2-1})} = \frac{x^2 - 2(x^2-1) + (x^2-1)}{x + \sqrt{x^2-1}} = \frac{1}{x+\sqrt{x^2-1}}$

$\ln(x - \sqrt{x^2-1}) = \ln\left(\frac{1}{x+\sqrt{x^2-1}}\right) = \ln(1) - \ln(x + \sqrt{x^2-1}) = 0 - \ln(x + \sqrt{x^2-1})$

ii) $\underbrace{\overbrace{\sqrt{x+\frac{1}{x}}}^{u} - \overbrace{\sqrt{x-\frac{1}{x}}}^{v}}_{a} = \frac{(u-v)(u+v)}{u+v} = \frac{x + \frac{1}{x} + x - \frac{1}{x}}{u+v} = \frac{\frac{2}{x}}{u+v}$

Accumulative Summation

$\begin{pmatrix} 1 & & \\ 1 & 1 & \\ 1 & 1 & 1 \end{pmatrix}(x) \Rightarrow$ $a_1 = x_1$, $a_2 = x_1 + x_2 = a_1 + x_1$, $a_3 = x_1 + x_2 + x_3 = a_2 + x_3$

Banded Matrix

Bandwidth = smallest k such that all nonzero entries are not further away from the diagonal

$(\text{Upper Triangular})^{-1}$ is also upper triangular if existent. Diag $\neq 0$ if exist.

Kronecker Product

- store $B \cdot a_{ij}$ for if the same scalar happens again
- Nur dort $(A \otimes B)^{(\text{Teil})}$ berechnen wo $x_i \neq 0$ für $(A \otimes B)\vec{x} = ?$
- $(A \otimes B)x = B \cdot \begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix} \cdot A^T$

$= \begin{bmatrix} x_1 a_1 b_1 + x_3 a_2 b_1 + x_2 a_1 b_2 + x_4 a_3 b_3 & x_1 a_3 b_1 + x_2 a_4 b_1 + x_2 a_3 b_2 + x_4 a_4 b_3 \\ x_1 a_1 b_3 + x_3 a_2 b_3 + x_2 a_1 b_4 + x_4 a_3 b_4 & x_1 a_3 b_3 + x_2 a_4 b_3 + x_2 a_3 b_4 + x_4 a_4 b_4 \end{bmatrix}$

Schur-Complement

$(n+m)\times(n+m)$ Matrix $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \Rightarrow S = D - CA^{-1}B$ das Schurkomplement

$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix} = \begin{pmatrix} I_n & -A^{-1}B \\ 0 & I_m \end{pmatrix}\begin{pmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix}\begin{pmatrix} I_n & 0 \\ -CA^{-1} & I_m \end{pmatrix}$

QR-Least Squares

$A = QR \Rightarrow \|QRx - b\|_2$ minimieren $\Rightarrow Rx = Q^{-1}b \overset{\text{ortho}}{=} Q^Tb \Rightarrow \|Rx - Q^Tb\|_2$

$\Rightarrow$ überbestimmt $\Rightarrow$ untere (Null-)Zeilen ignorieren $\Rightarrow x = R_0^{-1}b_0$

cost = cost of Householder-QR = $O(mn^2)$

has superior stability*, is "failsafe" but expensive for large Matrizes

bad at utilizing sparsity. SVD would be even more stable.

*compared to extended Normal Equations

Extended Normal Equations: if A sparse, then makes A^TA sparse

$$\begin{bmatrix} -I\alpha & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$ "useless" $r = Ax - b$ // α optional

$\Rightarrow$ way more of A but way less multiplication $\Rightarrow$ good if A sparse

$A^TAx = A^Tb$

SVD

$A = U \Sigma V^H$

sorted singular values unique $= \Sigma$ unique

U and V^H not unique

$A = U\Sigma V^H = \sum_{j=1}^{p} \sigma_j (U)_{:,j} (V)^H_{:,j}$ (3.4.8) // rank-1 Matrices

$\mathrm{Rank}(A) = \mathrm{Rank}(\Sigma) = r$

$N(A) = \mathrm{Ker}(A) =$ "rows of V higher than r"

Rank-1 Perturbations

Changing only one row $\tilde{A} = A + e_j \cdot z = A + uv^H$ // uv^H has rank 1

$$(2.6.18) \begin{bmatrix} A & u \\ v^H & -1 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \xi \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \Rightarrow A\tilde{x} = b - \frac{uv^HA^{-1}}{1 + v^HA^{-1}u} b \quad (2.6.21)$$

and generally, for rank-k modifications $\Rightarrow$ Sherman-Morrison-Woodbury

$$(A + UV^H)^{-1} = A^{-1} - A^{-1}U(I + V^HA^{-1}U)^{-1}V^HA^{-1}$$

Lyapunov

We have $AX + XA^T = I$ and $C \cdot \mathrm{vec}(X) = b$

b entspricht $I \Rightarrow b = \mathrm{vec}(I) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \Rightarrow$ calculate $AX + XA^T$ with $\mathrm{vec}(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$

find C.

SPAI – Sparse (Matrix) Approximate Inverse

$B = \underset{X \in P(A)}{\mathrm{argmin}} \|I - AX\|_F$ $\Rightarrow$ Columns of SPAI independently calculable with $b_i = \underset{x_i \in P(A_{:,i})}{\mathrm{argmin}} \|e_i - Ax_i\|_2$

Frobenius Norm

$\sqrt{\text{elements}^2}$

Norm-Constrained (3.4.31)

given $A \in \mathbb{K}^{mn}$, $m \geq n$, find $x \in \mathbb{K}^n$, $\|x\|_2 = 1$, $\|Ax\|_2 \to min$

$\|Ax\|_2^2 = \|U\Sigma V^T x\|_2^2 = \|\Sigma y\|_2^2 = \sum_\ell \sigma_\ell y_\ell^2 \to min$

U ortho => no influence in Norm

$y = V^T x$ $\quad \|y\|_2 = 1$

=> nur das kleinste sigma wählen => $y = e_n$ => $x = Ve_n$

ähnlich aber mit zusätzlicher row

neueste row einfach weglassen. Es geht in der Aufgabenstellung nicht um sie.

Low-Rank Approximation

$\|A\|_F = \sum_{j=1}^{p} \sigma_j^2$ kleinste sigma abschneiden, bis man den Rang hat, den man möchte. => beste k-Rang approx

// d.h. $\|A - F_k\|_F$ wurde minimiert

Total Least Squares: Nearest solvable LSE

Take overdetermined System and calculate best rank-m Approx of $[A \; b]$ (rank of A if lower than rank of b)

by computing the SVD: $[\hat{A} \; \hat{b}] = \sum_{j=1}^{n} \sigma_j (U)_{:,j} (V)_{:,j}^T$

Multiply both sides with $V_{:,j}$ => $\hat{A} V_{n,n+1} + \hat{b} (V)_{n+1,n+1} \stackrel{!}{=} 0$ (3.5.4) somehow because V ortho

Constrained least Squares

underdetermined System $Cx = d$ as Constraint

Idea: Max of $L(x, \lambda) = \frac{1}{2}\|Ax - b\|^2 + \lambda^T(Cx - d)$ becomes infinity if $Cx \neq d$

So we search for saddlepoint or min-in-1-direction

$\frac{\partial L}{\partial x}(x, \lambda) = A^T(Ax - b) + C^T\lambda \stackrel{!}{=} 0$

$\frac{\partial L}{\partial \lambda}(x, \lambda) = Cx - d \stackrel{!}{=} 0$ => $\begin{bmatrix} A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} A^Tb \\ d \end{bmatrix}$ is System with Constraint $Cx = d$

Impulses
input x, output y, impulse response $(0,\dots,h_0,h_1,\dots,h_{n-1},0,\dots)$

$$\begin{bmatrix} y_0 \\ \vdots \\ y_{2n-2}\end{bmatrix} = \begin{bmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & \ddots & & \vdots \\ h_{n-1} & & \ddots & h_0 \\ 0 & \ddots & & h_1 \\ \vdots & & & \vdots \\ 0 & \cdots & 0 & h_{n-1}\end{bmatrix}\begin{bmatrix} x_0 \\ \vdots \\ x_{n-1}\end{bmatrix}$$

// -1 wegen index bei 0

h: defined by one vector.

height $2n-1$ because overlap in middle when creating periodicity by expanding.

Discrete Convolution
$$y_k = \sum_{j=0}^{n-1} h_{k-j}\, x_j = H.\mathrm{row}(k)\cdot \vec{x}$$

* bzw $2n-1$, einfach

$$FT \Rightarrow c(t) = \left(\sum_{j=0}^{\mathrm{size}(H)} h_j e^{-2\pi i j t}\right)\cdot b(t)$$

ω^{tj}

```
periodicconvfft(h, x) { return fft.inv((fft.fwd(x)).cwiseProduct(fft.fwd(x)); }
```

n-periodic $\Rightarrow$ dimension of H is $n\times n$
$\Rightarrow$ circulant Matrix

All circulant have same EW
1st row = vector u
$\Rightarrow C v_k = \lambda_k v_k = \omega_n^{kj}\cdot \sum_{l=0}^{n-1} u_l\, \omega_n^{-kl}$

Toeplitz: $\begin{bmatrix} u_0 & u_1 & u_2 & \cdots \\ u_{-1} & \ddots & & \\ \vdots & & & u_0 \end{bmatrix} \Rightarrow$ circulantmatrix

$\Rightarrow \begin{bmatrix} T x \\ * \end{bmatrix} = C\begin{bmatrix} x \\ 0 \end{bmatrix}$

(4.5.9)

$\Rightarrow$ convolution $\Rightarrow$ fft

Root unity
$$\omega_n = e^{-\frac{2\pi i}{n}} \qquad \omega_n^n = 1 \qquad \omega_n^{-l} = e^{\frac{2\pi i}{l}} = \overline{\omega_n^l} \qquad \omega^{-kl} = (\overline{F}_n)_{kl}$$

Eigenvector $v_k = \left[\omega_n^{jk}\right]_{j=n}^{n-1}$ of circulant matrixes with $\lambda_k = \omega_n^{kj}$

$$\sum_{i=0}^{n-1} (\omega_n^{m+k})^i = 0$$

Fourier (4.2.13)
$$\begin{bmatrix} \omega_n^0 & \longrightarrow & & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \omega_n^2 \cdots & \omega_n^{n-1} \\ \downarrow & \cdot & \longrightarrow & \omega_n^{n-2} \\ & \downarrow & & \omega_n^{(n-1)^2}\end{bmatrix} = \left[\omega_n^{ij}\right]_{i,j=0}^{n-1} \in \mathbb{C}^{n,n}$$

$\frac{1}{\sqrt{n}} F_n$ is unitary
$F_n^{-1} = \frac{1}{n} F_n^H = \frac{1}{n}\overline{F}_n$

Cols of F_n are its eigenVectors

$v_k^H v_m = \begin{cases} n, & m=k \\ 0, & m\neq k\end{cases}$

$$CF_n = F_n D \quad \text{with} \quad D_{k,k} = \sum_{l=0}^{n-1} u_l\, \omega_n^{-kl}, \quad D = \overline{F_n}\,\vec{u}$$

Circulant
$$Cx = \frac{1}{n}\overline{F}_n\, \mathrm{diag}(d_1, d_2, \dots, d_n)\, \overline{F}_n^{-1} x = F_n^{-1}\, \mathrm{diag}\, F_n x$$

DFT-based deblurring
fft2: $p \to Y$
Y.cwiseQuotient($[\lambda_{\mu\nu}]$)
inverse fft2 (4.2.47)

2D-Fourier
$$C = F_m (F_n Y^T)^T = F_m Y F_n = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2} (F_m)_{k_1,j_1} (F_n)^T_{k_2,j_2} \quad (4.2.46)$$

Inverse $Y = F_m^{-1} C F_n^{-1} = \frac{1}{mn}\overline{F}_m C \overline{F}_n$

FFT
size(y) = 2m

$$c_k = \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} = \sum_{j=0}^{m-1} y_{2j}\, \omega_m^{jk} + \omega_n^k \sum_{j=0}^{m-1} y_{2j+1}\, \omega_m^{jk}$$

notice m vs n

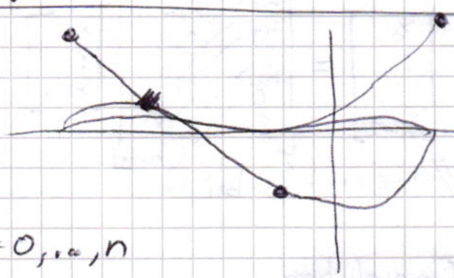
Householder QR (3.3.16) $Q = H(v) = I - 2\frac{vv^H}{v^Hv}$ with $v = \frac{1}{2}(a \pm \|a\|_2 e_1)$

Horner Scheme (5.2.6?)

(5.2.11)
Lagrange Polynomials

an punkt alle ausser einem 0

$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}} \frac{t - t_j}{t_i - t_j}, \; i = 0, \dots, n$



$$p(t) = \sum_{i=0}^{n} L_i(t)\, y_i = \prod_{j=0}^{n} (t - t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{(t - t_i)} y_i$$

$$\lambda_i = 1 / (t_i - t_0) \cdots (t_i - t_{i-1}) \cdot (t_i - t_{i+1}) \cdots (t_i - t_n)$$

$O(\overset{\text{sum} \cdot \text{Produkt}}{N n^2})$ (N: eval) oder mit precomputing $O(nN)$ $\Longrightarrow$ precomp λ_is and eval with (5.2.28) Barycentric Formula

Drawback: updating one datapoint means changing all basis

Evaluation using Neville-Aitken scheme: Good for a single Evaluation.

$k < \ell$ $\quad p_{k,\ell}$ = unique Polynomial of degree $(\ell - k)$ through the known points $(t,y)_k \dots (t,y)_\ell$

First calculate for just each (t, y) point:

$$p_{k,k}(x) = y_k$$

then

$$p_{k,\ell} = \frac{1}{t_\ell - t_k}\Big((x - t_k)\, p_{k+1,\ell}(x) - (x - t_\ell)\, p_{k,\ell-1}(x)\Big)$$

Extrapolation to zero

Lagrange: works well if function is even: $f(t) = f(-t)$ and behaves nicely around h

Given smooth function f, find approx of f'
Idea: approx using differencequotient $\Rightarrow$ Cancellation

Numerically stable Alternative:

Neville-Aitken: requires even function. Symmetric diff-quotient behaves like a polynome for a $(2(n+1))$ times continuously diff-able function.

$\Rightarrow$ Approx diff-quotient with taylor polynome

$\Rightarrow$ Neville-Aitken starting with small intervals from $(x-h)$ to $(x+h)$.

The longer it takes, the better the approximation.
Error is estimated by difference between the last two approximations

Update-friendly:

	0	1	2	3
t_0	y_0	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$
t_1	y_1	$p_{1,2}$	$p_{1,3}$	
t_2	y_2	$p_{2,3}$		
t_3	y_3			

Newton Basis (Update-Friendly) (5.2.49)

Adding Point $\Rightarrow$ Add a row to the Matrix below to compute the new coefficient a.

Instead of solving that LSE for a, we use "divided differences" recursion to compute nothing more than we need to (5.2.51). We do that with Aitken-Neville. (5.2.54).

Trigonometric Interpolation (Code 5.6.11)

Takes sine and cosine as basis and transforms it to complex polynome

$q(t) = e^{-2\pi i n t} \cdot p(e^{2\pi i t})$

no. of data points — Lagrangian barycentric interpolation in $\mathbb{C}$

Goal: periodicity same as the original function.

$\Rightarrow$ if we know its T-Periodic, use $\cos(\frac{2\pi}{T} j t)$ "potence"

Equidistant Trigonometric Interpolation => Fourier

2n equidistant nodes $t_k = \frac{k}{2n+1}$ $k = 0, \dots, 2n$

$\Rightarrow b_k = e^{\frac{2\pi i n k}{2n+1}} \cdot y_k = \sum_{j=0}^{2n} \gamma_j e^{\frac{2\pi i j k}{2n+1}}$ $c := [\gamma \text{ vector}]$

values — different values

(5.6.14) $\Rightarrow$ $\boxed{c = \frac{1}{2n+1} F_{2n+1} b}$ and then $\boxed{\begin{aligned}\alpha_j &= \frac{1}{2}(\gamma_{n-j} + \gamma_{n+j}) \\ \beta_j &= \frac{1}{2i}(\gamma_{n-j} - \gamma_{n+j})\end{aligned}}$ $j = 1, \dots, n$ $\alpha_0 = \gamma_0$

// vermutlich beziehen sich auf Koeff für sinus und cosinus

(Code 5.6.15) $\boxed{\text{Taylor: } \sum_{k=0}^{m} \frac{1}{k!} f^{(k)}(x) h^k + R_m(x,h), \quad R_m(x,h) = \frac{1}{(m+1)!} f^{(m+1)}(\xi) h^{(m+1)}}$ hoch

$\xi \in [\underbrace{t_k}_{x}, \underbrace{t_{k+1}}_{x+h}]$ // nähert an: $f(x+h)$ // $h = x - x_0$

Approximation by global polynomials

Taylor: $f \in C^{k+1}$ smoothness requirement

$f(t_0) + f'(t_0)(t-t_0) + \frac{f''(t_0)}{2}(t-t_0)^2 + \dots + \frac{f^{(k)}(t_0)}{k!}(t-t_0)^k$

Bernstein: (6.1.7)

Polynomial best: Interval $[-1, 1]$ $f \in C^r$ ← smoothness => Error converges with rate r

for $n \geq r$ $\inf_{p \in P_n} \|f - p\|_{L^\infty([-1,1])} \leq (1 + \frac{\pi^2}{2}) \frac{(n-r)!}{n!} \|f^{(r)}\|_{L^\infty([-1,1])}$

n r

→ Transformation

$\Phi : [-1, 1] \to [a, b]$ $\Phi(\hat{t}) = \frac{1}{2}(1-\hat{t})a + \frac{1}{2}(1+\hat{t})b$

$\Phi^* : [a,b] \to [-1, 1]$ $\Phi^*(f(\hat{t})) = f(\underbrace{\underbrace{\Phi(\hat{t})}_{\in [a,b]}}_{\in [-1,1]})$

$\boxed{\text{... Of Norms see (6.1.21)}}$

Lagrangian Interpolation Error: (6.1.50), (6.1.38)

Ableiten Multidimensional: S.598 (8.4.9), (8.5.9)

Hermite Interpolation: (5.4.5)

transform to $[a,b]$: $p(x) = H_{00}(t) p_k + H_{10}(t)(x_{k+1} - x_k) m_k + H_{01}(t) p_{k+1} + H_{11}(t)(x_{k+1} - x_k) m_{k+1}$

with $t = (x - x_k)/(x_{k+1} - x_k)$

Chebychev Lagrange Interpolation

nth Chebychev Polynomial $T_n(t) = \cos(n \cdot \cos^{-1}(t)) \quad -1 \leq t \leq 1$

Nodes fix ~~...~~ Endpoints better $\Rightarrow$ no weird oscillations

Chebynodes in $[a,b]$: $t_k = a + \frac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right)+1\right), \; k = 0,\dots,n$

//bound with Lebesgue constant λ_T (6.1.91)

Chebynodes are roots of Cheby poly!

Can also be applied locally (6.5.5)

Error estimate Lagrange (6.1.50) $\Rightarrow$ larger Mesh means larger Error

poor smoothness $\rightarrow$ still algebraic convergence, if analytic in Interval then even expon.

Quadratur

Int dt $\Rightarrow$ t wird c_j und t in Funktion

(7.1.2): $\int_a^b f(t)\,dt \approx Q_n(f) = \sum_{j=1}^{n} w_j^n f(c_j^n)$ // Ignore superscript n

$\mathbb{R} \ni$ weights, nodes $\in [a,b]$

wikip.: $Q_n = \int_a^b p_n\,dx = (b-a)\sum_{i=0}^{n} w_i f(x_i)$

$w_j = \frac{1}{b-a}\int_a^b L_{in}(x)\,dx$

$L_{in} = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots}$ (7.15)

Verschiebung: $\Phi[-1,1] \to [a,b]$: $\Phi(\tau) = a + \frac{1}{2}(b-a)(\tau+1)$

Gegeben Q_n in $[-1,1]$ gesucht in $[a,b]$

$$\int_a^b f(t)\,dt = \int_{-1}^{1} f(\Phi(\tau))\frac{d\Phi}{d\tau}\,d\tau = \frac{1}{2}(b-a)\int_{-1}^{1} f(\Phi(\tau))\,d\tau$$

$$\approx Q_n(f(\tau)) \longrightarrow \approx Q_n(f(\Phi(\tau))) \cdot \frac{1}{2}(b-a) =: Q_n^{[a,b]}$$

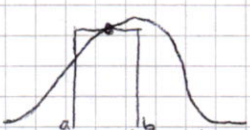
$\Rightarrow$ generic Quadrature formula has different weights

$\hat{f}(t) := f(\Phi(t))$, $\hat{w}_j \cdot \frac{1}{2}(b-a) = w_j$

$c_j = \frac{1}{2}(1-\hat{c}_j)a + \frac{1}{2}(1+\hat{c}_j)b$ // $\hat{f}, \hat{w}, \hat{c} \in [-1,1]$

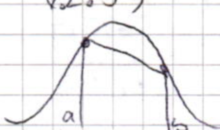
$$\int_a^b f(t)\,dt \approx \frac{1}{2}(b-a)\sum_{j=1}^{n}\hat{w}_j \hat{f}(\hat{c}_j) = \sum_{j=1}^{n} w_j f(c_j)$$

(7.2.5)



midpoint rule

$f(\frac{1}{2}(b+a)) \cdot (b-a)$



trapez

$Q_{trp}(f) = \frac{1}{2}(f(0)+f(-1))$

$Q_{trp} = \frac{b-a}{2}(f(a)+f(b))$

other rules p.526

Order(Q_n) = [max degree polynomial for which Q_n is exact] + 1

order is invariant under affine transformation

Error approx estimates: (7.3.39), (7.3.42)

An n-point Quad. rule has order $\geq n$ if and only if

$\forall w_j$: $w_j = \int_a^b L_{j-1}(t)\,dt$

L_{j-1} = the $(j-1)$-th Lagrange polynomial (5.2.11) with the ordered node set $\{t_1,\dots,t_n\}$

Smoothing by Transformation

$$\int_a^b \underbrace{\Phi(t)}_{\notin C^\infty} \underbrace{f(t)}_{\in C^\infty} dt$$ only algebraic error convergence

$\cancel{\frac{d\Phi}{dt} = \Phi'(t) \Rightarrow \frac{d\Phi}{ds}}$

$s = \Phi(t) \Rightarrow t = \Phi^{-1}(s)$

$$\Rightarrow \int_a^b s\, f(\Phi^{-1}(s))\, dt = \int_{\Phi(a)}^{\Phi(b)} s f(\Phi^{-1}(s))(\Phi^{-1}(s))' ds \qquad \frac{dt}{ds} = (\Phi^{-1}(s))' \Rightarrow dt = (\Phi^{-1}(s))' ds$$

$\Rightarrow s \in C^\infty \Rightarrow$ error cvg exponential

Explicit Euler: Err onestep $= y(t_k) + h y'(t_k) - y(t_{k+1})$
$\hookrightarrow$ Taylor $\Rightarrow$ Err onestep $= \frac{1}{2} y^{(2)}(\xi) h^2$

Composite Simpson/Trapezoidal: (7.4.5)

Detecting order of convergence

$\varepsilon_k = \|x^{(k)} - x^*\|$ assume $\varepsilon_{k+1} \approx C\varepsilon_k^p \Rightarrow \log(\varepsilon_{k+1}) \approx \log(C) + p \cdot \log(\varepsilon_k)$

this for two steps $\Rightarrow p = \dfrac{\log(\varepsilon_{k+1}) - \log(\varepsilon_k)}{\log(\varepsilon_k) - \log(\varepsilon_{k-1})}$

OR: for RK start with $y' = \lambda y \Rightarrow k_1, k_2, k_3$ (depending on y_j) $\Rightarrow y_{j+1} \Rightarrow$ get rid of y_j to see factor of a step $\Rightarrow$ its higest order of λ is the order.
$\Rightarrow$ then check $|F(\mu)| < 1$ to find stability interval
$F(h\lambda)$ = was dazukam

Splines

continuous: Left derivative = right side derivative

We can go cubic and require also $S''_{[x_i, x_{i+1}]} = S''_{[x_{i+1}, x_{i+2}]}$

- Complete splines: $S(t_0) = c_0,\ S(t_n) = c_n$
- Natural splines: $S''(t_0) = S''(t_n) = 0$
- Periodic spline: $S'(t_0) = s'(t_n),\ s''(t_0) = s''(t_n)$

(S. 421 oben)

Termination (8.2.21) bzw (8.2.23)

Iteration efficiency: (8.3.38) Inhaltsverzeichnis S. 560

Secant Method: (8.3.23)

Solve LSE with Newton (8.4.1) given $F(x^*) = 0$

Newton's Method converges locally quadratically in 1D.

$F(x+h) \approx F(x) + dF(x)h$ $\boxed{x_{k+1} = x_k - DF(x_k)^{-1} \cdot F(x_k)}$ (8.4.1)

Product Rule (8.4.9)

Matrix Inverse: $\boxed{D\, inv(x) H = -X^{-1} H X^{-1}}$

$$\frac{DX^{-1}}{Dx} = -X^{-1} \frac{DX}{Dx} X^{-1}$$

big X, small x

$$0 = \frac{d}{dw} A^{-1}(w) A(w) = \frac{dA^{-1}}{dw} \cdot A + A^{-1} \frac{dA}{dw}$$

$$\Rightarrow \frac{dA^{-1}}{dw} = -A^{-1} \frac{dA}{dw} A^{-1}$$

$\Rightarrow$ if small x is big X
$\boxed{DX^{-1} = -X^{-1} \cdot 1 \cdot X^{-1}}$?

$\Rightarrow$ Newton Correction $S = X^{(k)} A X^{(k)} - X^{(k)}$ (8.4.34)

$\Rightarrow$ Iteration $X^{(k+1)} = X^{(k)}(2I - AX^{(k)})$ (8.4.35) to calculate Matrix inverse approximately

Damping Newton

$x^{(k+1)} = x^{(k)} - \lambda^{(k)} DF(x^{(k)})^{-1} F(x^{(k)})$ (8.4.47) $0 < \lambda < 1$

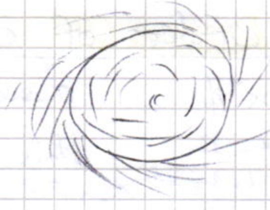
To find λ => Affine invariant damping strategy (8.4.49)

reduce damping if passed <=> increase lambda
if failed, increase by factor of 2 <=> decrease by factor 2

Stiff initial value Problems

Example with rotation Matrix ODE: $y' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} y + \lambda(1 - \|y\|^2) y$

If $\|y_0\| = 1$ => $\|y(t)\| = 1$ because it's a rotation



Explicit Euler (11.2.7)

$y_{k+1} = y_k + h_k f(t_k, y_k)$

$\Rightarrow \Psi^h y = y + hf(y)$

Implicit Euler (11.2.13)

$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$

$\Rightarrow \Psi^h y = z \mid z = y + hf(z)$

Equidistant Konvergence:

Explicit Euler $err \in O(h^\alpha)$ $\alpha = 1$
Implicit Euler order 1
Midpoint implicit order 2

"All SSM converge algebraically with some order $p \in \mathbb{N}$ $err = O(h^p)$ "

SSM: $y_{k+1} = \Psi^{h_{k+1}} y_k$, $h_{k+1} = t_{k+1} - t_k$

Goal: $y_k \approx y(t_k)$

=> Calculate y_k's and interpolate them => yields function

// Cost = $\frac{s}{h}$ ≈ no. of equidistant timesteps to achieve error reduction by factor ρ $h_{new} = \rho^{-\frac{1}{p}} h_{old}$ "

"The larger p, the less effort for same Error reduction //

Diagonalization Euler

$y' = My = V D V^{-1} \underbrace{y}_{z(t)}$ => $z'(t) = V^{-1} y'(t) = V^{-1} V D \underbrace{V^{-1} y(t)}_{z(t)} = Dz \Leftrightarrow z_1' = \lambda_1 z_1$, $z_2' = \lambda_2 z_2$, ...

generalized => $y(t) = \exp(Mt) y_0$ (12.1.34)

// $e^{Diag} = \begin{bmatrix} e^{a_1} & & \\ & e^{a_2} & \\ & & e^{a_3} \end{bmatrix}$

Decoupling Euler: (12.1.42)

Runge-Kutta Blow-up (12.1.16)

blowup for $k \to \infty$ if $S(z)$ = Stabilityfunction $= 1 + z b^T (I - zA)^{-1} 1 = \det(I - zA + z1b^T)$ is > 1 in absolute value.

$z = \lambda h$ with ODE $y' = \lambda h$ // Vars from Butcher $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$

Explicit Euler $\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$ $S(z) = 1 + z$

Explicit Trapezoidal $\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$ $S(z) = 1 + z + \frac{1}{2} z^2$

Classical RK4 method $\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$ $S(z) = 1 + z + \frac{1}{2} z^2 + \frac{1}{6} z^3 + \frac{1}{24} z^4$

E.E: avoid blowup: $h < \frac{2}{|\lambda_i|}$

Diagonalisation Runge-Kutta

$y' = My = VDV^{-1} y$ $\qquad k_i = M(y_0 + h \sum_{j=1}^{i-1} a_{ij} k_j)$

$z_k = V^{-1} y_k$, $k = 0, 1$ $\qquad \hat{k}_i = V^{-1} k_i = D(z_0 + h \sum_{j=1}^{i-1} a_{ij} \hat{k}_j)$

$z_1 = z_0 + h \sum_{i=1}^{s} b_i \hat{k}_i$

$$\begin{array}{ccc} y' = My & \xrightarrow{\text{Diagonalisation}} & z'_e = \lambda_e z_e \\ & & e = 1, \dots, d \\ \downarrow \text{RK-SSM} & & \downarrow \text{RK-SSM} \\ \varphi_h & \xrightarrow{\text{Diagonalization}} & \varphi_{e,h} \end{array}$$

$y_1 = \varphi_h y_0$

RK-SSM discrete evolution for $z'_e = \lambda_e z_e$

No blow-up of $(y_k) \Leftrightarrow$ No blow-up for $(z_{e,k})_k$

$\varphi_{\lambda,h} y = S(z) y = S(h\lambda) y$, $y \in \mathbb{C}$ (12.1.49)

$\Rightarrow$ region of absolute stability given by $S_\varphi = \{z \in \mathbb{C} \mid |S(z)| < 1\} \subset \mathbb{C}$

$y_k = S(z)^k y_0 \Rightarrow |y_k| = |S(z)|^k |y_0|$

Linearisation:
$y' = f(y_0) + Df(y_0)(y - y_0)$
$= My + b$

$$\begin{array}{ccc} y' = f(y) & \xrightarrow{\text{Lin}} & y' = Df(y_0) y + b \\ \downarrow \text{RK-SSM} & & \downarrow \text{RK-SSM} \\ \varphi_h & \xrightarrow[\text{of the linearized eq. (by Taylor expansion)}]{\text{Lin}} & \varphi_h \end{array}$$

for small timestep the behavior of explicit RK-SSM applied to $y' = f(y)$ close to y_0 is determined by the EValues of $Df(y_0)$. If EW mit large modulus $\Rightarrow$ timestep constraint to avoid blowup

If $Re(\lambda) > 0 \Rightarrow y(t)$ grows exponentially $\Rightarrow$ blow-up of numerical solution o.k.

General (implicit) Runge Kutta

$k_i := f(t_0 + c_i h, y_0 + h \sum_{j=1}^{s} a_{ij} k_j)$, $i = 1, \dots, s$ $\qquad y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i$

↳ non linear system of equations with $s \cdot d$ unknowns. $\quad k_i \in \mathbb{R}^d$

Increment eq. are usually solved by simplified Newton method. (12.3.25)

$$x^{(k+1)} = x^{(k)} - DF(x^{(0)})^{-1} F(x^{(k)}) \quad \text{for } F(x) = 0$$

General RK-SSM

$k_i = \lambda (y_0 + h \sum_{j=1}^{s} a_{ij} k_j)$

$z = \lambda h$

$y_1 = y_0 + h \sum b_i k_i$

$$\Leftrightarrow \begin{bmatrix} I - zA & 0 \\ -h b^T & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ k_s \\ y_1 \end{bmatrix} = \begin{bmatrix} \lambda y_0 \\ \vdots \\ \lambda y_0 \\ y_0 \end{bmatrix} \Rightarrow y_1 = y_0 + h b^T (I - zA)^{-1} \lambda y_0 \mathbf{1} = S(z) y_0$$

▷ $S(z)$ = a rational function $\frac{p(z)}{q(z)}$

Quadrature by Transformation

$$\int_0^2 \sqrt{\frac{2-t}{t}}\, f(t)\, dt$$

$s = \sqrt{\frac{2-t}{t}} \iff \cancel{t=s}$

$2 - t = s^2 t$

$s^2 t + t = 2$

$t = \frac{2}{s^2+1}$

$$\frac{dt}{ds} = \frac{2}{s^2+1} \cdot 2s = \frac{4s}{s^2+1} \Rightarrow dt = \frac{4s}{s^2+1}\, ds$$

$$\int_{\sqrt{\frac{2-t}{t}}=\infty}^{\sqrt{\frac{2-t}{t}}=0} \frac{4s^2}{s^2+1}\, \cancel{ds}\, f\left(\frac{2}{s^2+1}\right) ds$$

$s = \tan(q) \iff q = \arctan(s)$

$$\frac{ds}{dq} = \cancel{\frac{\cos(q)+\sin(q)}{\sin^2(q)}}\; 1 + \tan^2(q)$$

$ds = 1 + \tan^2(q)\; dq$

$\Rightarrow \cancel{q}\quad q = \arctan\left(\sqrt{\frac{2-t}{t}}\right)$

$$\int_{\arctan(\infty)}^{\arctan(0)} \frac{4\tan^2(q)}{\tan^2(q)+1}\, f\left(\frac{2}{\tan^2(q)+1}\right)(1+\tan^2(q))\, dq$$

$$= \int_{\frac{\pi}{2}}^{0} 4\tan^2(q)\, f\left(\frac{2}{\tan^2(q)+1}\right) dq$$

$$= -\int_0^{\frac{\pi}{2}} 4\tan^2(q)\, f\left(\frac{2}{\tan^2(q)+1}\right) dq$$

$$q = \underset{[-1,1]\to[a,b]}{\Phi(\tau)} = a + \frac{1}{2}(b-a)(\tau+1)$$

$$= 0 + \frac{1}{2}\frac{\pi}{2}(\tau+1) = \frac{\pi}{4}\tau + \frac{\pi}{4}$$

$$\Rightarrow \tau = \frac{4}{\pi}\left(q - \frac{\pi}{4}\right)$$

$$\frac{dq}{d\tau} = \frac{\pi}{4} \Rightarrow dq = \frac{\pi}{4}\, \cancel{dq}\, d\tau$$

$$= \int_{\frac{4}{\pi}(0-\frac{\pi}{4})=-1}^{\frac{4}{\pi}(\frac{\pi}{2}-\frac{\pi}{4})=1} 4\tan^2\left(\frac{\pi}{4}\tau+\frac{\pi}{4}\right) f\left(\frac{2}{\tan^2\left(\frac{\pi}{4}\tau+\frac{\pi}{4}\right)+1}\right) \cdot \frac{\pi}{4}\, d\tau$$

$$\approx \sum_{j=1}^{n} \underbrace{\omega_j^n}_{\text{Gauss Weights}}\, \pi \tan^2\left(\frac{\pi}{4}\underbrace{c_j^n}_{\text{Gauss Nodes in } [-1,1]} + \frac{\pi}{4}\right) f\left(\frac{2}{\tan^2\left(\frac{\pi}{4}c_j^n+\frac{\pi}{4}\right)+1}\right)$$

Upper, Lower, StrictlyUpper ... : MatrixXd B = A.triangularView<Eigen::Lower>();

Diagonal : MatrixXd D = A.diagonal() ~~%~~.asDiagonal();
vektor Matrix

alle verwendeten als ref

& = alle verwendeten als Kopien

$\underbrace{\text{std::function<int (VectorXd)>}}_{\text{auto}}$ f = [&a, b](VectorXd v) -> int {... return i;};

return param

as referenz copy

return (optional)

Block : B = A.block(i,j, p,q) — start upper left point; size

oder Eigen::Block<double>(m.derived, i,j, p,q);

vector.segment(i_from, n_amount)

Eigen: vec.data() unsure

Matrix from std::vector : MatrixXd A = MatrixXd::Map(&myvec[0], rows, cols);

oder A = Map<MatrixXd>(myvec);

std::vector vec(12); ← initial size

vec.push_back(); a=vec.at(0); ← bounds check b=vec[1]; ← no check

vec.erase(vec.begin() + i);

vec = VectorXd::Constant(n, const)
Zero
Ones
Random
↳ m, falls Matrix

static_cast<MatrixXd>(A) ← if you're sure

dynamic_cast<MatrixXd(B) returns Nullpointer if failed type match, or error if it is a reference

ColMajor/RowMayor Mapping /dox/group-TutorialMapClass.html

Eigen::Map<Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor>>(p, rows, cols)

or ColMajor (default)

or MatrixXd

e.g. std::vector<double> a => double* p = a.data();

pointer to first elem

e.g. 3, 3

is the major of the vector

std => Eigen : Eigen::Vector3d ~~#~~(vec.data());

using Vector = Eigen::VectorXd;

using namespace std;

#include <iostream> #include <vector>

#include <cmath>

#include <Eigen/Dense>

DiagonalMatrix<double, 3> m(1,2,3);

VectorXd::LinSpaced(steps = , min, max)

⚠ aliasing m = m.transposeInPlace();

or m = m.transpose().eval();

Matrix.setFromTriplets(triplets.begin(), triplets.end());

Dense -> Sparse : denseM.sparseView()

Sparse -> Dense : denseM = MatrixXd(sparseM)

A.coeffref(0,0) = 2 ← sparse

pow(x, a) = x^a std::exp

Identity(a,b)

SparseLU: /dox-devel/classEigen_1_1SparseLU.html

```
A.makeCompressed();
Eigen::SparseLU<MatrixXd> splu;
splu.analyzePattern();
splu.factorize()
x= splu.solve(b)
```

oder compute

DenseLU: `x = A.lu().solve(b)`

```
HouseholderQR<MatrixXd> qr(A.rows(), A.cols());
qr.compute(A); MatrixXd Q = qr.householderQ();
```

Cholesky: $A = LDL^T$

Untere Dreiecksmatrix

$\xrightarrow{\text{Normalengl.}}$ `solution = (A.transpose()*A).ldlt().solve(A*b);`

mit mehreren b: `MatrixXd X = A.ldlt().solve(B)`

SVD: `Eigen::JacobiSVD<MatrixXd> svd (A, Eigen::ComputeFullU | Eigen::ComputeFullV);`

```
MatrixXd U = svd.matrixU(); MatrixXd V = svd.matrixV();
VectorXd sv = svd.singularValues();
MatrixXd Sigma = sv.asDiagonal();
```

⚠ sv will never be 0, only < Epsilon

```
mgl::Figure fig;
fig.title("b_0(t)");
fig.xlabel("t");
fig.ylabel("y");
fig.plot(t, y-shift, "r").label("b_0(t)");
fig.save("b0_n");
```

```
#include<Eigen/Core>
Eigen::MatrixXd m;
m.unaryExpr(&functionname);
//apply to each Element in Matrix
```

```
//elementwise apply to foo
std::transform(foo.begin(), foo.end(), bar.begin(), myfunc);
//with another parameter to myfunc   arg
```

```
fig.plot(x, y, "+r").label("Sample points");
fig.plot(u, v, "b").label("Function");
fig.legend(); fig.save("plot.eps");
fig.fplot("sin(3*x)"); //analytical plot
fig.setlog(bool, bool);
```

```
std::sort(myTriplets.begin(), myTriplets.end(),
    [s](const Triplet &a, const Triplet &b){
        return ((a.row()<<s)+ a.col()) < ((b.row()<<s)
            +b.col());}
    ); // myTriplets is now sorted by row, then col

unsigned getNumberOfDigits (unsigned int i){
    return i>0 ? (int) log2((double)i)+1 : 1;
```