

I. Computing with Matrices and Vectors

Skript: 1.2.12 Initialization of dense matrices

- using `index_t = typename dynMat_t::Index;`
- using `entry_t = typename dynMat_t::Scalar;`

Skript: 1.2.14 Demonstration code for access to matrix blocks in EIGEN.

Def.: (Row/Column major) $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \in \mathbb{K}^{m \times n}$

• R.M.: $A_{arr} \ 1, 2, \dots, 8, 9$. (C-arrays, Python)

~ Mapping: $(i, j) \mapsto m(i-1) + (j-1)$

• C.M.: $A_{arr} \ 1, 4, 7, 2, 5, 8, 3, 6, 9$. (EIGEN/Matlab)

~ Mapping: $(i, j) \mapsto n(j-1) + (i-1)$

Skript: 1.3.2 BLAS - Basic LinAlg Subprograms

Bem. 1: (1.4.2 Cost of basic operation)

$x, y \in \mathbb{K}^n; a \in \mathbb{K}^m; b \in \mathbb{K}^n; A \in \mathbb{K}^{m \times n}; B \in \mathbb{K}^{n \times k}$

description	#mult/div	#add/sub	complexity
$x^T y$	n	$n-1$	$O(n)$
$a b^T$	nm	0	$O(nm)$
AB	mnk	$m(k(n-1))$	$O(mnk)$
Matlab eig of $A \in \mathbb{K}^{n \times n}$			$O(n^3)$

Bem. 2: (Safe check for $= 0$)

if $(nq < 1e-3 * \text{norm}(A(:, j)))$ % Matlab

Bem. 3: Orthogonalisation (Algorithm)

• QR $\Rightarrow$ "good" • Gram-S. $\Rightarrow$ "bad"

Def.: (1.5.3 Roundoff errors) Max. rel. roundoff er:

$EPS := \max_{x \in \mathbb{R}^n} |x - rd(x)| / |x| \stackrel{\text{machine precision}}{=}$

Assumption (1.5.31 "Axiom" of roundoff analysis)

[...] $x * y = (x * y)(1 + \delta), f(x) = f(x)(1 + \delta), \forall x, y, u$ with $|\delta| \leq EPS$.

Bem. 4: (Cancellation) $\hat{=}$ subtraction of almost equal numbers ($\Rightarrow$ extreme amplifications of relative errors)

Bem. 5: (Avoiding Cancellation)

- avoid subtraction of two almost equal sign.
- Ex. 1.5.51 (stable discriminant formula)
- Ex. 1.5.53 (trigonometric identities)
- Ex. 1.5.58 (Summation of exponential series)

Ex. 1.5.60 ("rather approximate than suffer cancellation")

$\rightarrow$ Taylor expansion: (Analysis) Sei $f: [a, b] \rightarrow \mathbb{R}$ genügend oft diff'bar. Reihenentwicklung um x_0 :

$$T_k(x; x_0) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!} \frac{(x-x_0)^k}{k!} \quad \xi \in [x_0, x]$$

Restglieddarstellung: $R_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!} \frac{(x-x_0)^{k+1}}{(k+1)!}$

Def.: (1.5.72 Stable Algorithm)

[...] $\exists C \geq 1: \forall x \in X: \exists \tilde{x} \in X: \|x - \tilde{x}\|_2 \leq C \omega(x) \text{eps} \|x\|_2$

$\omega(x) \hat{=}$ #elementary ops. in algo. $\wedge F(x) = F(\tilde{x})$.

Bem. 6: In principle stability of an algorithm for computing Ax is confirmed, if $\forall x \in X$ the computed result $y = y(x)$ satisfies:

$$\|y - Ax\|_2 \leq C_{min} EPS \|x\|_2 \|A\|_2$$

with a small const. $C > 0$ independent of data and problem size.

Bem. 7: Sought: $Ax = b, A \in \mathbb{K}^{m \times n}, x, b \in \mathbb{K}^n$.

• residual: $r := b - Ax$.

• stable if: $\|r\| \leq C_{res} \cdot EPS \cdot \|b\|$.

Bem. 8: (cost of solving a general LSE)

$\Rightarrow O(n^3)$ [e.g. Gaussian elimination, LU]

• special matrices: $\rightarrow$ diagonal $\rightarrow O(n)$

$\rightarrow$ unitary $\Rightarrow O(n^2) \rightarrow$ triangular $\Rightarrow O(n^2)$

Skript: 1.6.86 Direct solution of dense linear

e.g.: system in EIGEN

• `MatrixXd mat = MatrixXd::Random(n,n);`

• `MatrixXd X = mat.lu().solve(b);`

$\rightarrow$ Elimination solve $\Rightarrow$ setup phase $O(n^3) +$ phase $O(n^2)$

Skript: 1.6.91 Efficient implementation of inverse

power method in EIGEN.

Bem. 3: (rank-1 modification) $\tilde{A} = A + uv^T$

$\tilde{A} \tilde{x} = b \Rightarrow$ Block elimination

$$\Rightarrow x = A^{-1}b - \frac{A^{-1}u(v^T(A^{-1}b))}{1 + v^T(A^{-1}u)}$$

Skript: Lem. 1.6.113 Sherman-Morrison-Woodbury

formula, generalisation of the above.

Bem. 10: (S2A1) Sparse linear system

for general $A, B: \bullet nnz(A \otimes B) = nnz(A)nnz(B)$

• $nnz(A+B) \leq nnz(A) + nnz(B)$.

Def.: (Sparse matrix storage format)

• triplet format (CCO): (S2A3)

```
struct Triplet {
    // template <class scalar>
    Triplet(void) {}
    Triplet(int i, int j, const scalar& v) {}
    Triplet(size_t i, size_t j, double v) {}
    int i, j, v;
    size_t i, j; // row & column
    scalar v; // value at row, col
};
```

```
struct TripletMatrix {
    size_t rows, cols; // sizes: rows & cols
    vector<Triplet<double>> triplets; // dynamic size
    Matrix<double, -1, -1> densify() const; // densify()
    Matrix<double, -1, -1> M = Matrix<double, -1, -1>::Zero(rows, cols);
    for (auto it = triplets.begin(); it != triplets.end(); ++it) {
        M(it->i, it->j) += it->v;
    }
    return M;
};
```

• compressed row format (CRS): (S2A3)

```
struct ColValPair {
    ColValPair(size_t col, double v) {}
    int col, v;
};

bool operators<const ColValPair & other> const {
    return this->col < other.col;
}

size_t col; // col-index
double v; // value at col
```

```
struct CRSMatrix {
    size_t rows, cols; // size of matrix rows, cols
    vector<vector<ColValPair<double>>> row_ptr; // for each row, all
    // vector of (col, val) pairs
    Matrix<double, -1, -1> densify() const {
        Matrix<double, -1, -1> M = Matrix<double, -1, -1>::Zero(rows, cols);
        size_t i = 0;
        for (auto it = row_ptr.begin(); it != row_ptr.end(); ++it) {
            for (auto it2 = it->begin(); it2 != it->end(); ++it2) {
                M(i, it2->col) += it2->v;
            }
            ++i;
        }
        return M;
    }
};
```

Bem. 11: Matrix assembled in triplet format.

```
vector<Triplet<double>> triplets;
SparseMatrix<double, RowMajor> spMat(rows, cols); // fill the vector
spMat.setFromTriplets(triplets.begin(), triplets.end());
spMat.makeCompressed();
TripletMatrix<double> T; T.triplets.reserve(#triplets);
(S2A3) for (auto it = triplets.begin(); it != triplets.end(); ++it) {
    T.triplets.push_back(*it);
}
```

Bem. 12: (S2A3) Solve $Ax = y$ for banded Matrix

```
using SparseLU;
template <class Vector>
void solveEigen(const Vector& ka, &b, &y, Vector& x) {
    typedef typename Vector::Scalar TScalar; // Dimensions
    unsigned int n = y.size();
    SparseMatrix<Scalar> A(n, n);
    A.reserve(3);
    for (unsigned int i = 0; i < n; ++i) {
        A.insert(i, i) = 2; // diag; A.coeffRef(i, i) = 2;
        if (i < n-1) A.insert(i, i+1) = a(i);
        if (i > 2) A.insert(i, i-2) = b(i-2);
    }
    A.makeCompressed();
```

```
SparseLUK SparseMatrix<Scalar>> solver; // call SparseLU
solver.analyzePattern(A); solver.compute(A);
x = solver.solve(y);
```

LU 1. Var

II Iterative Methods for Non-LSFs

Bem. 13: (Ex. 1.6.3, 1.8.1, 2.0.1) Nodal analysis.

Kirchhoffs law: • Schleife: $\sum V_i = 0$

• Knoten: $\sum I_{in} = \sum I_{out}$

$\rightarrow I_{k,j} \hat{=}$ current node $k \rightarrow$ node j : $I_{k,j} = -I_{j,k}$

Def.: (Iteration Error) • $e_k := x_k - x^*$

• $E_k := \|x_k - x^*\|$

Skript: (Def.:) 2.1.8. Local & Global convergence

Def.: 2.1.9 Linear convergence. A sequence

$x_k, k=0,1,\dots$ in $\mathbb{R}^n$ converges linearly to $x^* \in \mathbb{R}^n$

$\exists L < 1: \|x_{k+1} - x^*\| \leq L \|x_k - x^*\| \quad \forall k \in \mathbb{N}_0$

Bem. 14: The smallest possible L is called rate of (linear) convergence.

Bem. 15: How to tell lin. conv. in numerical test

(x^* known). $E_{k+1} \approx L E_k \approx L^2 E_{k-1} \dots$

$\Rightarrow E_k \approx L^k E_0 \Rightarrow \log(E_k) \approx k \log(L) + \log(E_0)$

$\rightarrow (k, \log(E_k))$ on a straight line with slope $\log(L) \leq 0$ (semilog plot).

Bem. 16: From tabulated values: check if $E_{k+1} \approx L E_k$

Def.: 2.1.17 Order of convergence. A convergent

sequence $x_k, k=0,1,\dots$ in $\mathbb{R}^n$ converges with

order p to $x^* \in \mathbb{R}^n$ if

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

Bem. 17: Identifying conv. of order p from

measured E_k : $E_{k+1} \approx C E_k^p$

$\Rightarrow \log(E_{k+1}) \approx \log(C) + p \log(E_k)$

$\Rightarrow \log(E_{k+1}) - \log(E_k) \approx p(\log(E_k) - \log(E_{k-1}))$

$\Rightarrow p = (\log(E_{k+1}) - \log(E_k)) / (\log(E_k) - \log(E_{k-1}))$

(SFA) convergence rate.

err.push_back(abs(x_k - x_exact));
for(auto it = err.begin(); it != err.end(); ++it) {
rate.push_back((log(*it) - log(*it-1)) / (log(*it-1) - log(*it-2)));
}

Bem. 18: Number of correct in every step.

$x_k = x^*(1 + \delta_k)$, $\delta_k \hat{=}$ rel. error, $\delta_k \leq 10^{-l}$

$\Rightarrow x_k$ has l correct digits.

$\Rightarrow |x_{k+1} - x^*| \approx C |x_k - x^*|^2$

$\Rightarrow |\delta_{k+1}| \approx C |\delta_k|^2, \delta_k \neq 0 \Rightarrow |\delta_{k+1}| \approx C |\delta_k|^2$

$\Rightarrow C |\delta_k| \approx 1 \wedge \delta_k = 10^{-l} \Rightarrow \delta_{k+1} \approx 10^{-2l}$

Def.: • abs. tol.: $\|x_k - x^*\| \leq \text{tol}_{abs}$

• rel. tol.: $\|x_k - x^*\| \leq \text{tol}_{rel} \cdot \|x^*\|$

Def.: Correction based termination.

• abs.: $\|x_{k+1} - x_k\| \leq \text{tol}_{abs}$

• rel.: $\|x_{k+1} - x_k\| \leq \text{tol}_{rel} \cdot \|x_{k+1}\|$

Bem. 19: Linearly conv. iteration with known rate $L < 1$.

$\rightarrow \|x_k - x^*\| \leq \frac{1}{1-L} \|x_k - x_{k+1}\|$ { can replace $\|x_k - x^*\|$ in termination criteria

$\rightarrow \|x_k - x^*\| \leq \frac{1}{1-L} \|x_k - x_{k+1}\|$ { a priori termination

Def.: Fixed point iterations. (FPI)

$x_{k+1} = \Phi_F(x_k)$, with iteration fct $\Phi_F: U \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$

$x^* = \lim_{k \rightarrow \infty} x_k$ & continuous $\Rightarrow x^* = \Phi_F(x^*)$

Bem. 20: FPI is consistent: $\Phi_F(x) = x \Leftrightarrow F(x) = 0$.

Bem. 21: • $|\Phi'(x^*)| < 1 \Rightarrow$ convergence

• $|\Phi'(x^*)| > 1 \vee 1 < |\Phi'(x^*)| \Rightarrow$ divergence

Lem.: (2.2.10) If $\Phi: U \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$, $\Phi(x^*) = x^*$, Φ diff'bar in

x^* , and $\|D\Phi(x^*)\| < 1$, then conv locally at least

linearly (rate $\approx \Phi'(x^*)$). $D\Phi(x^*) \hat{=}$ Jacobian

Bem. 22: If $D\Phi(x^*) = 0 \Rightarrow$ local quadratic.

Lem.: 2.2.11 If $\Phi \in C^{m+1}(U, \mathbb{R})$, $\Phi(x^*) = x^*$ for some x^* in the

interior of U , and $\Phi^{(l)}(x^*) = 0$ for $l=1, \dots, m$, $m \geq 1$, then

the FPI conv locally to x^* with order $m+1$.

Lem.: 2.2.12 Sufficient condition for linear

convergence of fixed point iteration (Skript).

Bem. 23: Bisection Code 2.3.2; "kind of lin. conv."

Def.: Newton's method. Tangent in x_k :

$F(x) := F(x_k) + F'(x_k)(x - x_k) \Rightarrow \tilde{F}(x) = 0$

$\Rightarrow x_{k+1} = x_k - F(x_k) / F'(x_k), \Delta F(x_k) \neq 0$

Bem. 24: Newton's method $\Rightarrow$ local quadratic conv.

since $\Phi'(x^*) = 0$ & Lem. 2.2.11.

Def.: Secant method. Approx durch Sekante,

NST der Sekante.

$S(x) = F(x_k) + (x - x_k) \frac{F(x_k) - F(x_{k-1})}{x_k - x_{k-1}} \Rightarrow s(x) = 0$

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

Bem. 25: Find the maximal guaranteed order of conv

of secant method through analytical considerations.

$|e_k| = C |e_{k-1}|^p$ (i)

$|e_{k+1}| = C |e_k|^p \Rightarrow |e_{k+1}| = C |C |e_{k-1}|^p|^p = C^{p+1} |e_{k-1}|^{p^2}$ (II)

given by problem: $|e_{k+1}| = K |e_k| |e_{k-1}|$

$\stackrel{(i)}{\Rightarrow} |e_{k+1}| = K |C |e_{k-1}|^p| |e_{k-1}| = K C |e_{k-1}|^{p+1}$ (I)

$\stackrel{(II)}{\Rightarrow} C^{p+1} |e_{k-1}|^{p^2} = K C |e_{k-1}|^{p+1} \Rightarrow K C^{p+1} |e_{k-1}|^{p^2-p-1} = 1$

$\Rightarrow p^2 - p - 1 = 0$

$\Rightarrow p_{1/2} = \frac{1}{2} (1 \pm \sqrt{5}) \Rightarrow p = \frac{1}{2} (1 + \sqrt{5}) \approx 1.6$

Def.: Asymptotic efficiency. $\text{eff} = \frac{\text{gain extra correct digits}}{\text{work}}$

• work $\hat{=}$ # F-eval + # F'-eval = W (per step)

• $|e_k| \leq p |e_{k-1}|$, digit gained $-\log_{10}(p)$, $p < 1$

• efficiency = $\frac{\log(p)}{W \cdot k(p)}$, $k(p) \hat{=}$ min. # of it. for accuracy

• $k(p) = \log |\log(p)| / \log(p)$

• Asymp. eff. = $\frac{\log(p)}{W \cdot \log |\log(p)|}$

• $\frac{\log(p)}{W} \hat{=}$ indicator for eff. of an iterative method.

Def.: Newton's method (in $\mathbb{R}^n$). as before.

$\tilde{F}_k(x) = F(x_k) + DF(x_k)(x - x_k)$, $DF(x_k) \in \mathbb{R}^{n \times n} \hat{=}$ Jacobian

$\tilde{F}_k(x) = 0 \Rightarrow x_{k+1} = x_k - DF(x_k)^{-1} F(x_k)$

Skript: Code 2.4.2: Newton's method in C++

Skript: Ex. 2.4.13 (A special quasi-linear system of equations)

Def.: • Product rule: $h \in \mathbb{R}^n \hat{=}$ perturbation vector

$D \tilde{F} x \rightarrow F(x) \cdot G(x) \tilde{F} h = DF(x)h \cdot G(x) + F(x) \cdot DG(x)h$

• Chain rule: $D \tilde{F} x \rightarrow F(G(x)) \tilde{F} h = DF(G(x)) DG(x)h$

Bsp.: (lectures notes) For $F(x) = A(x)x - b$ find $DF(x)$.

$\rightarrow A(x) = T x + x \|x\|_2$

• derive: $x \rightarrow x \|x\|_2$

(i) prod. rule: $D \tilde{F} x \rightarrow x \|x\|_2 \tilde{F} h = x \cdot h \cdot \|x\|_2 + x D \tilde{F} x \rightarrow x \|x\|_2 \tilde{F} h$

(ii) $\|x\|_2 = \sqrt{x^T x}$ (chain rule: $F \mapsto \tilde{F}, G \mapsto x^T x$)

$D \tilde{F} x \rightarrow x \|x\|_2 \tilde{F} h = \frac{1}{\|x\|_2} (h^T x + x^T h) = \frac{1}{\|x\|_2} x^T h$

$\Rightarrow D \tilde{F} x \rightarrow x \|x\|_2 \tilde{F} h = \|x\|_2 h + x \cdot \frac{1}{\|x\|_2} x^T h = (\|x\|_2 I + \frac{x x^T}{\|x\|_2^2}) h$

$\Rightarrow DF(x) = \frac{A(x)}{\|x\|_2} + \frac{x x^T}{\|x\|_2^2} \hat{=}$ rank-1 mod. of $A(x)$

Def.: simplified Newton correction.

$\Delta x_k := DF(x_{k-1})^{-1} F(x_k)$, $DF(x_{k-1})^{-1} \hat{=}$ LU-dec available

Bem. 26: Termination criteria.

$\|\Delta x_k\| \leq \text{tol}_{rel} \|x_k\|$ or $\|\Delta x_k\| \leq \text{tol}_{abs}$

Def.: Affine invariant damping strategy Skript: (2.4.49)

Skript: Code 2.4.50 Generic damped Newton method based on natural monotonicity test.

II. Data Interpolation in 1D

Def.: Interpolation conditions. Given:

data points $(t_i, y_i) \in \mathbb{R} \times \mathbb{R}, i=0, \dots, n$ with $t_i \neq t_j, y_i \neq y_j$

Sought: interpolant $f: I \subset \mathbb{R} \rightarrow \mathbb{R}, f(t_i) = y_i, i=0, \dots, n$

our setting: interpolant f can be represented

as a linear combination of basis fct $b_j: I \rightarrow \mathbb{R}$:

$$f(t) = \sum_{j=0}^n c_j b_j(t), c_j \hat{=} \text{coeff./parameter}$$

Def.: piecewise linear interpolation, "tent functions"

$$f(t) = \sum_{j=0}^n y_j b_j(t) \quad f(t_i) = y_i, b_j(t_i) = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases}$$

$\sim b_j(t_i) \hat{=} \text{cardinal basis for } \xi \in t_i$

Bem.27: Basis rep. + interpolation conditions:

$$\sum_{j=0}^n c_j b_j(t_i) = y_i, \quad \begin{pmatrix} b_0(t_0) & \dots & b_n(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \dots & b_n(t_n) \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix} = y$$

$\Rightarrow A c = y$

Bem.28: f exists & unique $\Leftrightarrow$ LSE $\uparrow$ has unique

solution. Necessary: $m=n$.

Bem.29: (SGA2) p.w. lin. interpolation formula:

$$s(x) = [(t_{i+1} - x) \cdot y_i + (x - t_i) \cdot y_{i+1}] / [t_{i+1} - t_i]$$

(SGA3) Horner for p.d.f. in matlab convention

```
template <typename coeffVec,
pair<double, double> evalp(const coeffVec &c,
const double &x) {
pair<double, double> p; double px, dpx;
int s=c.size(); px=c[s];
for(int i=s; i>0; --i) { px=x*px+c[i]; } //Horner O(n)
dpx=(s-1)*c[0];
for(int i=s; i>0; --i) { dpx=x*dpx+(s-i)*c[i]; }
p.first=px; p.second=dpx;
return p; }
```

Def.: Barycentric formula (interpolation eval)

$\hookrightarrow$ Skript: (3.2.27), Code 3.2.28

Def.: Aitken-Neville, Skript: (3.2.29) A.N. scheme

(5746) A.N. vector<double> t={1/2, 1/3, 1/4}; //O(n^2)

vector<double> U={2/3, 3/2, 3/4};

int n=t.size();

for(int i=0; i<n; ++i) { //replace j with n and get A.N. algo.

for(int k=i-1; k>0; --k) {

U[k]=U[k+1]+(U[k+1]-U[k])*(x-t[i])/(t[i]-t[k]);

res=U[0]; //A

Def.: Newton Basis. $N_0(t)=1, N_k(t)=(t-t_0)$

$$N_k(t) = \prod_{i=0}^{k-1} (t-t_i), t_i \hat{=} \text{Stützstellen}$$

Interpolationspolynom: $p_n(t) = \sum_{k=0}^n y_k N_k(t)$

mit divided differences (3.2.48)

Bsp.: $t_0=0, y(t_0)=1, \frac{3-1}{2-0}=2, \frac{8-2}{2-0}=3$

$t_1=1, y(t_1)=3, \frac{4-3}{2-1}=1, \frac{8-3}{2-1}=5$

$t_2=2, y(t_2)=8, \frac{11-5}{2-1}=6, \frac{11-8}{2-1}=3$

$\Rightarrow p_2(t) = 1 + 2(t-0) + 3(t-0)(t-1)$

Skript: Code 3.2.51: Divided diff., recursive implement.

Skript: Code 3.2.54: — " — eval by modified Horner

Bem.30: A.N. vs. div. diff.: • Gefahr der Auslöschung

bei der dividierten Differenzen.

• A.N. ist gut geeignet, wenn ein Interpolationspoly.

nur an ganz wenigen Stellen ausgewertet werden

soll, dabei ist er weniger anfällig gegen Auslöschung.

Lem.: 3.2.67 Abs. conditioning of poly. interpolation

$$\|I_s\|_{\infty \rightarrow \infty} = \left\| \sum_{i=0}^s |L_i| \right\|_{\infty} = \text{Lebesgue const } \Lambda_s \text{ (of } s)$$

Def.: Lagrange polynomials. $L_i(t) = \prod_{j \neq i} \frac{t-t_j}{t_i-t_j}, i=0, \dots, n$

$\sim L_i$ polynomials satisfy $L_i \in \mathbb{P}_n$ and $L_i(t_j) = \delta_{ij}$.

Def.: (3.2.68) Norms on spaces of functions

• supremum norm: $\|f\|_{\infty} := \sup_{t \in I} |f(t)|$

• L^2 -norm: $\|f\|_{2, I} := \sqrt{\int_I |f(t)|^2 dt}$

• L^1 -norm: $\|f\|_{1, I} := \int_I |f(t)| dt$

Bem.31: 3.2.70 Lebesgue const. for equidistant nodes

$$\Lambda_s \geq C e^{n/2}$$

Thm.: 3.3.3. Local shape preservation by p.w. lin. interp.

$\hat{=}$ change of y_i affect interpolant only on $[t_i, t_{i+1}]$

Bem.32: Locality of p.w. lin. interpolation

Def.: 3.4.1 Cubic Hermite polynomial interpolation.

Given data points $(t_j, y_j) \in \mathbb{R}^2, j=0, \dots, n$ with pairwise

distinct ordered nodes t_j , and slopes $c_j \in \mathbb{R}$, the p.w.

cubic Hermite interpolant $s: [t_0, t_n] \rightarrow \mathbb{R}$ is def. by

$$s|_{[t_i, t_{i+1}]} \in \mathbb{P}_3, i=0, \dots, n, s(t_i) = y_i, s'(t_i) = c_i, i=0, \dots, n.$$

Bem.33: local monotonicity preserving p.w. cubic

Hermite intp. $\Rightarrow$ Use slope limiting w/ (3.4.12)/(3.4.14)

Def.: 3.5.1 Spline space. Given an interval $I := [a, b] \subset \mathbb{R}$

and a knot set/mesh $M := \{a = t_0 < \dots < t_n = b\}$, the vector

space $S_{d, M}$ of the spline functions of degree d

(order $d+1$) is defined by

$$S_{d, M} = \{s \in C^{d-1}(I): s_j := s|_{[t_{j-1}, t_j]} \in \mathbb{P}_d \forall j=1, \dots, n\}$$

Bsp.: Endterm: $\dim(\{s \in C^k([a, b]): f|_{[t_{j-1}, t_j]} \in \mathbb{P}_k, j=1, \dots, m\})?$

Log: $(k+1) \cdot m - (m-1)(k+1) = \dots = k+1$

Def.: (3.5.4) Computing cubic spline interpolants.

$\sim$ LSE $((n-1) \times (n+1))$ eq. (3.5.3)

Bem.34: (3.5.10) Types of cubic spline interpolants.

(1) Complete cubic spline interpolation } see Skript

(2) Natural cubic spline " } for details!

(3) Periodic cubic spline " }

Thm.: 3.4.17. Property of lin., monotonicity preserving

interpolation into C^1 :

$I: \mathbb{R}^n \rightarrow C^1(I)$, linear & monotonicity preserving

$\Rightarrow I(\varphi)'(t_j) = 0 \quad \forall t_j \in \mathbb{R}^n$.

Bem.35: Cubic spline interpolation is "almost

local": cardinal interpolants decay exponentially.

($\Rightarrow$ so called weak locality)

Def.: (3.4.3) Local representation of p.w. cubic Hermite interpolant:

lin. combination of generalized cardinal basis fct:

$$s(t) = y_{i-1} H_1(t) + y_i H_2(t) + c_{i-1} H_3(t) + c_i H_4(t), t \in [t_{i-1}, t_i] \quad (3.4.4)$$

(5104) Legendre polynomial & Gauss points

void legende(const VectorXd &x, MatrixXd &Lx, &DLx) { //eval Legendre poly

int n=Lx.cols()-1; int N=x.size(); //and derivative, 3-term recur.

for(int i=0; i<N; ++i) { Lx(i,0)=x(i); DLx(i,0)=0; DLx(i,1)=1;

for(int k=2; k<N; ++k) {

Lx(i,k)=(2*i+1-k)/k * Lx(i,k-1) + (2*i-k+1)/k * Lx(i,k-2);

DLx(i,k)=(2*i-k+1)/k * DLx(i,k-1) + (2*i-k+1)/k * DLx(i,k-2);

DLx(i,k-1)=(2*i-k)/k * Lx(i,k-1) + (2*i-k)/k * Lx(i,k-2);

return Lx(n);

MatrixXd gaussPts(int n, double r0=1e-10, atol=1e-12) { //Gauss

MatrixXd zeros(n,n); double x0,x1,p0,p1,s; //points w/ secant.

for(int i=0; i<n; ++i) {

for(int j=i+1; j<n; ++j) { //initializes initial guesses

if(j==1) x0=-1; x1=1; //initializes initial guesses

else x0=zeros(j-2,k-2);

if(j==k) x1=1; x1=zeros(j-1,k-2);

else x1=zeros(j-1,k-2);

p0=Pxx(x0,k); //secant method

for(int i=0; i<n; ++i) {

p1=Pxx(x1,k); s=(p1-k*x0)/(p1-p0);

if(Pxx(x1-k*s,k) < 0) { //regula falsi

x0=x1; p0=p1; } else {

x1=x1-k*s; if(fabs(s) < max(atol, r0*(min(fabs(x0), fabs(x1)))) {

zeros(i-1,k-1)=x1; break; } } } //end of secant.

return zeros;

(51043) Numerical integration of improper integral

void gaussWells(int n, VectorXd &w, &w0) { // 5.3.35 Gauss-Wells

w.resize(n); w0.resize(n); //computes weights & nodes of n-point

if(n==0) {

w0=0; w0=2; }

else {

Vector b(n-1); MatrixXd J=MatrixXd::Zero(n,n);

for(int i=1; i<n; ++i) {

double d=(i)/sqrt(4*i*i-1);

J(i,i-1)=d; J(i-1,i)=d;

EigenSolver<MatrixXd> eig(J);

x=eig.eigenvalues().real();

w=2*eig.eigenvectors().real().topRows<1>().cwiseProduct(

eig.eigenvectors().real().topRows<1>()); }

usage of eigenvalues

IV. Approximation of functions in 1D

Thm. 4.1.6 Uniform approx. by polynomials.

Thm. 4.1.11 L^∞ polynomial best approx. estimate.

If $f \in C^r([-1, 1])$, $r \in \mathbb{N}$, then:
 $\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1, 1])} \leq (1 + \frac{\pi}{2})^r \frac{(n-r)!}{n!} \|f^{(r)}\|_{L^\infty([-1, 1])}$

Bem. 36: Asymptotics for degree $n \rightarrow \infty$: $O(n^{-r})$.

$\Rightarrow$ Algebraic conv. with rate r in the polynomial degree.

Def.: Affine transformation:

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

Def.: Pullback: $\{C^0([a, b]) \rightarrow C^0([-1, 1])\}$

$$\Phi^*: \{f \mapsto \hat{f} \mapsto f(\Phi(\hat{t}))\}$$

Bem. 37: $\Phi^*: \mathcal{P}_n \rightarrow \mathcal{P}_n$ bij., i.e., $\Phi^* p = p$

$$\| \Phi^* f \|_{L^\infty([a, b])} = \| f \|_{L^\infty([-1, 1])}$$

$$\| (\Phi^* f)^{(r)} \|_{L^\infty([-1, 1])} = (\frac{1}{2}(b-a))^r \| f^{(r)} \|_{L^\infty([a, b])}$$

Bem. 38: VL of Thm 4.1.11.

$$\inf_{p \in \mathcal{P}_n} \| \Phi^* f - p \|_{L^\infty([-1, 1])} \leq C n^{-r} \| (\Phi^* f)^{(r)} \|_{L^\infty([-1, 1])}$$

Def.: 4.1.31. Types of asymptotic conv. of approx.

schemes: $\exists p > 0$: $T(n) \in n^{-p}$: alg. conv. (log log)

$\exists 0 < q < 1$: $T(n) \leq q^n$: exp. conv. (semilog) $\forall n \in \mathbb{N}$

Thm. 4.1.37 Representation of interpol. error:

$$[-1] \quad f(t) - L_J(f)(t) = f^{(n+1)}(\xi_t) / (n+1)! \cdot \prod_{j=0}^n (t - t_j) = C \cdot \omega_J(t)$$

with $C := f^{(n+1)}(\xi_t) / (n+1)!$, $\omega_J(t) := \prod_{j=0}^n (t - t_j)$ ["nodal poly."]

Def.: (4.1.43) Estimate:

$$\| f - L_J f \|_{L^\infty(I)} \leq \| f^{(n+1)} \|_{L^\infty(I)} / (n+1)! \cdot \| \omega_J(t) \|_{L^\infty(I)}$$

Thm. 4.1.56 Residue thm.

Bem. 39: Estimate of a contour integral

$$\int_{\gamma} f(z) dz = \int_0^1 f(\gamma(t)) \gamma'(t) dt$$

$$\Rightarrow \left| \int_{\gamma} f(z) dz \right| \leq | \gamma | \cdot \max_{t \in [0, 1]} | f(\gamma(t)) | \cdot | \gamma' | \leq \text{length}$$

Leu. 4.1.58 Residual formula for quotients

$$[-1] \quad \text{res}_p(\frac{p}{q}) = \frac{p'(p)}{q'(p)}, \quad p \in \mathbb{C}. \quad (\text{Thm. 4.1.37})$$

Def.: (4.1.63) Estimate an alternative to Leu. 4.1.41

$$| f(t) - L_J f(t) | \leq | \omega(t) / (2\pi i) \int_{\gamma} \frac{f(z)}{(z-t)\omega(z)} dz | \cdot \max_{z \in \gamma} | f(z) |$$

$$\leq | p | / (2\pi) \cdot \max_{z \in \gamma} | \omega(z) | / \min_{z \in \gamma} | \omega(z) | \cdot \text{dist}([a, b], \gamma)$$

Def.: 4.1.67 Chebyshev polynomials: $|t| \leq 1, n \in \mathbb{N}$

The n^{th} Cheb. poly. is $T_n(t) := \cos(n \arccos(t))$.

Bem. 40: $|T_n(t)| \leq 1$.

Thm. 4.1.68 3-term recursion for Cheb. poly.

$$[-1] \quad T_{n+1}(t) = 2t T_n(t) - T_{n-1}(t), \quad T_0 = 1, T_1(t) = t, \quad n \in \mathbb{N}.$$

Bem. 41: $T_n \in \mathcal{P}_n$ with leading coefficient 2^{n-1} .

Def.: Chebyshev points = $\xi_j^{(n)} \stackrel{!}{=} \text{optimal nodes?}$

$$\xi_j^{(n)} = \cos(\frac{2j-1}{2n} \cdot \pi), \quad j = 0, \dots, n-1$$

Thm. 4.1.73 Minimax property of cheb. poly.

Bem. 42: Cheb. interpolation on the interval $[-1, 1]$ is

characterized by:

• "optimal" nodes $J = \{ \xi \cos(\frac{2k+1}{2(n+1)} \cdot \pi) \mid k = 0, \dots, n \}$.

• $\omega(t) = 2^{-n} T_{n+1}(t)$, $\| \omega(t) \|_{L^\infty([-1, 1])} = 2^{-n}$, leading coeff. 1

• $f \in C^{n+1}([-1, 1])$: $\| f - I_J f \|_{L^\infty([-1, 1])} \leq \frac{(b-a)!}{2^{n+1} n!} \| f^{(n+1)} \|_{L^\infty([-1, 1])}$.

• $f \in C^{n+1}([a, b])$: $\| f - I_J f \|_{L^\infty([a, b])} \leq \frac{(b-a)!}{2^{n+1} n!} (\frac{b-a}{2})^{n+1} \| f^{(n+1)} \|_{L^\infty([a, b])} \sim \text{study asymptotics of } T \text{ for } h_k \rightarrow 0$ [i.e. h-conv.]

• Cheb. nodes in the interval $I = [a, b]$ are:

$$t_k := a + \frac{1}{2}(b-a)(\cos(\frac{2k+1}{2(n+1)} \cdot \pi) + 1), \quad k = 0, \dots, n$$

Bem. 43: Sensitivity of cheb. interp.: Lebesgue

$$\text{const. } \Lambda_J: \lambda_{\text{cheb}} \leq \frac{1}{\pi} \log(\Lambda(n)) + 1$$

Bem. 44: Code 4.1.70 $\xrightarrow{\text{C++}}$ (S3A3)

vector<double> chebpolmult(const int d, const double x) {

vector<double> V = {1, x};

for (int k=1; k<d; k++) {

V.push_back(2*x*V[k]-V[k-1]);

return V;

Bem. 45: (4.1.63) Estimate for cheb.

$$\| f - L_J f \|_{L^\infty([-1, 1])} \leq \frac{4|p|}{\pi} \cdot 1 / [(p^{n+1} - 1)(p^{p-1} - 2)] \cdot \max_{z \in \gamma} | f(z) |, \text{ p.l.}$$

Def.: Linalg: (Normalgleichung)

$$Ax = \underbrace{A(A^T A)^{-1} A^T}_{\text{projektion } P_A} y \xrightarrow{\text{kor. (A) } \rightarrow \text{20\%}} x = (A^T A)^{-1} A^T y \Rightarrow A x = A^T y.$$

Thm. 4.2.7 Mean square best approx. through

Cor.: 4.2.12 Best approximant by orthog. projection.

If q is the best " of in $V \Rightarrow f - q$ is " to $V \perp V$:

$$(f - q, p) = 0 \quad \forall p \in V \Leftrightarrow f - q \perp V.$$

Bem. 46: $f - q \hat{=}$ best approx. error.

Bem. 47: Algo. for computing q :

(i) Gram-S. $\rightarrow$ GNB = $\{ b_0, \dots, b_n \}$

(ii) $q = \sum_{j=0}^n (f, b_j) x b_j$

Thm. 4.2.31 3-term recursion for orthog. poly.

Def.: discrete L^2 -inner products

$$\text{Given: } J = \{ \xi_j \}_{j=0}^m: (f, g)_J := \sum_{j=0}^m f(\xi_j) g(\xi_j).$$

$$\cdot \text{Ex.: } \| f - p \|_J^2 = \sum_{j=0}^m ((f - p)(\xi_j))^2.$$

Thm. 4.3.2 Cheb. alternation thm.

Bem. (4.5.5) General local Lagrange interpolation

on a mesh.

Cor.: 4.5.6 p.w. poly. Lagrange interpolation operator:

The mapping $f \mapsto s$ defines a linear operator

$$I_M: C^0([a, b]) \rightarrow C_{M, \text{pw}}^0([a, b]).$$

Bem. 48: (Focus) Asymptotic conv. in maximum norm:

h-convergence: $n_j = n \quad \forall j$ (fixed local degree).

Consider sequence of meshes M_k with

$$\text{meshwidth } h_k := \max |x_j^{(k)} - x_{j-1}^{(k)}|.$$

• find bound: $\| f - I_{M_k} f \|_{L^\infty([a, b])} \leq C T(h_k)$

$\sim$ study asymptotics of T for $h_k \rightarrow 0$ [i.e. h-conv.]

Bem. 49: (Ex.) $f(t) = \arctan(t)$, on $[-5, 5]$.

• $M_k = \{ -5 + j h_k \mid j = 0, \dots, 2^k \}$, $h_k = 2^{-k}$ ("equidistant")

• Alg. conv.: $\| \text{err} \|_{\infty} = O(h^\alpha)$ $\alpha \hat{=}$ rate

• Empiric: $\alpha \approx n+1$, $n \hat{=}$ local pol. degree

Bem. 50: Estimate derived from (4.1.43)

$$\Rightarrow \| f - s \|_{L^\infty([x_0, x_m])} \leq \frac{h_{m+1}}{(m+1)!} \| f^{(m+1)} \|_{L^\infty([x_0, x_m])}$$

$$\text{with } h := h_m = \max |x_j - x_{j-1}|: j = 1, \dots, m$$

Bem. 51: p-convergence: M fixed, raise local degree

Def.: 4.5.14 p.w. cubic Hermite interpolant:

$$[-1] \quad s: [a, b] \rightarrow \mathbb{R}: \mathcal{P}_3 \Rightarrow O(h^4)$$

$$s[x_j, x_j] \in \mathcal{P}_3, \quad j = 1, \dots, m, \quad s(x_j) = f(x_j), \quad s'(x_j) = f'(x_j), \quad j = 0, \dots, m$$

Thm. 4.5.17 Cvg. of approx. by cubic Hermite interpol.

[Estimate] $[-1] \quad f \in C^4([a, b])$:

$$\| f - s \|_{L^\infty([a, b])} \leq \frac{1}{4!} h_m^4 \| f^{(4)} \|_{L^\infty([a, b])}.$$

Bem. 52: Rule of thumb: p.w. approx. with polynomials

of degree p , h-conv.: algebraic, maximal

rate p+1 (smoothness of f required!)

Def.: discrete L^2 -inner products (4.2.21)

$$\text{Given: } J = \{ \xi_j \}_{j=0}^m: (f, g)_J := \sum_{j=0}^m f(\xi_j) g(\xi_j).$$

Bem.: für Quadraturen immer auf die folgende Form transformieren $\int_a^b f(t) dt = \int_{-1}^1 f(\dots) ds$ (SMA1)
Bem.: Chebyshev Q.R. Clenshaw-Curtis rules $\rightarrow$ positive weights throughout

I. Numerical Quadrature

Bem. 53: Task: Compute $\int_a^b f(t) dt$. possible
• f given in procedural form (only point eval.)
• $\# \{f\text{-evals}\} \hat{=}$ measure of cost

Def.: 5.1.1 Quadrature rule on $[a, b]$.
[...]
 $\int_a^b f(t) dt \approx Q_n(f) := \sum_{j=1}^n w_j f(c_j)$.
 $\rightarrow w_j \in \mathbb{R} \hat{=}$ quadrature weights
 $\rightarrow c_j \in [a, b] \hat{=}$ nodes

Def.: linear Transformationen von Intervallen
 $\int_a^b f(t) dt = \int_{d-c}^{b-a} f(\frac{b-a}{d-c} \frac{t-d}{b-a} + c) \frac{b-a}{d-c} dt$
 $= \int_{d-c}^{b-a} \tilde{w}_j f(\tilde{c}_j) \frac{b-a}{d-c} dt$

Bem. 54: (Herleitung)
 $\Phi(t) = \alpha + \beta t \Leftrightarrow t = \frac{\Phi(t) - \alpha}{\beta} = \frac{\Phi(t)(b-a)}{d-c} - \frac{bc-ad}{d-c}$
 $\Phi(b) = d, \Phi(a) = c \mid \Phi(b) = \alpha b + \beta = d$
 $\Phi(a) = \alpha a + \beta = c \mid \Rightarrow$ solve
 $\Rightarrow \alpha = \frac{d-c}{b-a}, \beta = \frac{bc-ad}{b-a}$

Def.: Interpolation operator (Ann. linear)
 $I: \mathbb{R}^n \times \mathbb{R}^n \rightarrow V, ([c_j], [y_j]) \mapsto I_J[y]$

Bem. 55: $\int_a^b f(t) dt \approx \int_a^b I_J(f)(t) dt = \int_a^b I_J(I_J(c_j))(t) dt$
with $w_j = \int_a^b I_J(c_j)(t) dt \rightarrow$ unabh. von t .

Def.: Quadrature error
 $E_n(f) := |\int_a^b f(t) dt - Q_n(f)| \leq \frac{1}{2} (b-a) \|f - Af\|_{C([a,b])}$
with $Q_n(f) = \int_a^b (Af)(t) dt, Af = I_J[f(c_j)]$

Bem. 56: Polynomial Quadrature (5.2.1), (5.2.2)

Def.: 5.3.1 Order of quad. rule.
[...]
order(Q_n) := $\max \{q \in \mathbb{N}_0: Q_n(p) = \int_a^b p(t) dt \forall p \in \mathbb{P}_q\}$

Bem. 57: P_n are invariant under affine pullback $\rightarrow$ order of Q.R. isn't affected by trafo.

Thm.: 5.3.5 Sufficient order conditions for Q.R.

Thm.: 5.3.12 Maximal order of n -point Q.R.
The max. order of an n -point Q.R. is $2n$.

Thm.: 5.3.21. Existence of n -point quadrature formulas of order $2n$.

Def.: 5.3.26 Legendre polynomials on $[-1, 1]$.
The n -th Legendre polynomial P_n is def. by
• $P_n \in \mathbb{P}_n$ • $\int_{-1}^1 P_n(t) q(t) dt = 0 \forall q \in \mathbb{P}_{n-1}$
• $P_n(1) = 1$

Lemma: 5.3.27 Zeros of Legendre polynomials: P_n has n distinct zeros in $]-1, 1[$.
Gauss points $= \hat{=} \xi_j^n, j=1, \dots, n \hat{=}$ zeros of P_n .

Lemma: 5.3.28 Positivity of Gauss-Legendre quadrature weights.

Def.: 3-recursion for Legendre polynomials:
 $P_{n+1}(t) := \frac{2n+1}{n+1} t P_n(t) - \frac{n}{n+1} P_{n-1}(t), P_0 := 1, P_1(t) := t$

Bem. 58: $\tilde{P}_n := \frac{1}{\sqrt{n!}} P_n$ satisfy $\tilde{P}_1 := 0$.
(5.3.36) $t \tilde{P}_n(t) = \frac{n}{2n+1} \tilde{P}_{n-1}(t) + \frac{n+2}{2n+3} \tilde{P}_{n+1}(t)$

Thm.: 5.3.35 Quadrature error estimate for Q.R. with positive weights: [...]
 $E_n(f) = |\int_a^b f(t) dt - Q_n(f)| \leq \frac{1}{2} (b-a) \inf_{p \in \mathbb{P}_{n-1}} \|f - p\|_{C([a,b])}$

Def.: $f \in C^r$ Thm. 5.3.41 $|E_n(f)| = O(n^{-r})$, alg. conv.
• f has analytic extension to $\mathbb{C}$ -neighbourhood of $[a, b]$ $\xrightarrow{\text{Cheb. int. ext.}} |E_n(f)| = O(q^{-n}), 0 < q < 1$, exp. conv.

Bem. 59: smoothness of integrand is crucial for fast alg.
Ex.: smoothing: $\int_a^b f(t) dt = \int_a^b s f(s) ds \rightarrow$ now analytic

Bem. 60: The message of asymptotic convergence: (i) a priori [...]

We aim for error reduction by a factor of $p > 1$. (ii) a posteriori [...]
• Alg. conv.: $\frac{err_n}{err_0} \approx \frac{1}{p} \Rightarrow \frac{n_1}{n_0} = \frac{1}{p}$
• $\hat{v}_p \hat{=}$ measure for additional effort.

• Exp. conv.: $\frac{err_n}{err_0} \approx \frac{1}{p} \Rightarrow q^{n_1 - n_0} = \frac{1}{p} \Rightarrow n_1 = n_0 - \log(q)$
 $\rightarrow$ Fixed additional # of quad. nodes gain factor p in accuracy! $\otimes q \approx \frac{err_0}{err_n}$

Def.: Composite quadrature
mesh: $M = \{a = x_0 < x_1 < \dots < x_m = b\}$
 $\Rightarrow \int_a^b f(t) dt = \sum_{j=1}^m \int_{x_{j-1}}^{x_j} f(t) dt$ Q.R. here! (5.4.1)

Def.: Quadrature error estimates. $f \in C^r([a, b])$
 $|\int_{x_{j-1}}^{x_j} f(t) dt - Q_{n_j}(f)| \leq C |x_j - x_{j-1}|^{\min(r, q_j)} \|f^{(q_j)}\|_{C([x_{j-1}, x_j])}$

with $q_j \hat{=}$ local order, $Q_{n_j} \hat{=}$ local q.r. in $[x_{j-1}, x_j]$.
• If $q_j = q$ for all j :
 $\Rightarrow |\int_a^b f(t) dt - Q_n(f)| \leq C h_M^{\min(r, q)} \|f^{(q)}\|_{C([a, b])}$

with $h_M = \max_j |x_j - x_{j-1}|$.
 $\Rightarrow$ alg. conv. with rate $\min(r, q)$ (as $h_M \rightarrow 0$).
Letting h_M & fixed local q.r. n -convergence.

Bem. 61: $n_j \hat{=}$ # local nodes, for all j .
 $E_n(f) = O(n^{-\min(r, q)}) = O(h_M^{\min(r, q)})$
for $n \rightarrow \infty$ ($n \hat{=}$ # f -evals).
 $\Rightarrow$ alg. conv. with rate $\min(r, q)$ (as $h_M \rightarrow 0$).
Letting h_M & fixed local q.r. n -convergence.

Bem. 62: Composite quad. vs. G.L. quad.:
fixed local q.r., order q , equidistant mesh, $f \in C^r$:

$E_n^{CR}(f) \leq C n^{-\min(r, q)}$ for large n .
 $E_n^{GL}(f) \leq C n^{-r}$

$\Rightarrow$ In terms of rate: G.L. is at least as good as C.R., G.L. will "auto-detect" best possible rate!
And if f analytic then exp. conv. of G.L. $\leftarrow$ the clear winner

Bem. 63: concerning (5.5.3), Goal: $B_k = \frac{1}{2} \|f\|_{C([x_{k-1}, x_k])}$
Minimize $E(f) \leq \sum_k B_k h_k^3$ under constraint $\sum h_k = b-a$.
Focus on two cells:

minimize their combined error contribution
 $B_k(h_k + \delta)^3 + B_{k+1}(h_{k+1} - \delta)^3 \rightarrow \min$
Minimizer s^* satisfies: $B_k(h_k + s^*)^3 = B_{k+1}(h_{k+1} - s^*)^3$

Bem. 64:
At optimal cell size distribution all cells make the same contribution to the error bound.

Policy: Adaptive numeric quadrature (beneath 5.5.....)
(i) a priori [...]
(ii) a posteriori [...]
 $\Rightarrow$ iterative improvement of mesh $\Rightarrow$ either "move nodes" or "mesh refinement" (add node).

Bem. 65: Adaption loop for numerical quad. (alg.):
(1) - (4) [...] Skript. right after (5.5.7).

Def.: (5.2.3) Error estimates for poly. quad.
[...]
 $f \in C^r([a, b])$
 $\Rightarrow |\int_a^b f(t) dt - Q_n(f)| \leq \frac{1}{n!} (b-a)^{n+r} \|f^{(n+r)}\|_{C([a, b])}$

Lemma: 5.3.41 Quad. error estimates for C^r -integrands
[...]
 $q \geq r: E_n(f) \leq C q^{-r} (b-a)^{r+1} \|f^{(r)}\|_{C([a, b])}$
 $q < r: E_n(f) \leq \frac{1}{q!} (b-a)^{q+1} \|f^{(q)}\|_{C([a, b])}$

Bem. (zu Bem. 60): derivation of alg. conv.: Set $N_k = 2^k$
 $E_{N_k} / E_{N_{k-1}} \approx (C 2^{-(k-1)r}) / (C 2^{-kr}) = 2^r$
 $\Rightarrow r \approx \log(\frac{err_0}{err_n}) / \log(2)$

$\triangle$

Bem.: (S1B1) Orthogonality is preserved only by the implicit midpoint rule!
 EE&E don't preserve orthogonality.

XI. Numerical Integration

Def.: Numerical solution of ODE's:

$$\dot{y} = f(t) \Rightarrow y(t) = \int_0^t f(s) ds.$$

Def.: M.1.3 Solution of an ordinary diff. eq.

Lem.: M.1.4 Smoothness of solutions of ODE's

[...] If $f: I \times D \rightarrow \mathbb{R}^d$, $f(t, y(t)) \in C^r(I \times D, \mathbb{R}^d)$
 $\Rightarrow$ the trajectory $t \mapsto y(t) \in C^{r+1}(I)$.

Def.: M.1.5 Autonomous ODE

Def.: (M.1.20) Initial value problem

• Need to fix unique solution: specify $\downarrow$ value.

• IVP: $\dot{y} = f(t, y)$, $y(t_0) = y_0$.

Def.: (M.1.22) Autonomization. $\dot{y} = f(t, y)$:

$z(t) = \begin{bmatrix} y(t) \\ t \end{bmatrix}$ solves $\dot{z} = \begin{bmatrix} f(z_1, \dots, z_n, z_{n+1}) \\ 1 \end{bmatrix}$

Def.: (M.1.23) Conversion to 1st order ODE:

$\dot{y} = f(y, \dot{y})$: $z(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix}$ solves $\dot{z} = \begin{bmatrix} f(z_1, \dots, z_n, z_{n+1}) \\ z_{n+1} \end{bmatrix}$

Def.: M.1.26 Local Lipschitz continuity.

[...] $\exists L > 0$: $\|f(t, u) - f(t, v)\| \leq L \|u - v\|$.

Thm.: M.1.32 Thm. of Peano & Picard-Lindelöf.

[...] f locally Lipschitz $\Rightarrow \forall (t_0, y_0)$ initial conditions setting:
 the IVP has a unique solution y .

Bem. 66: Autonomous ODE's: always use $t_0 = 0$.

Def.: M.1.33 Evolution map: [...]

$\Phi: \mathbb{R} \times D \rightarrow D$, $(t, y_0) \mapsto y(t, y_0)$

Bem. 67: ODE $\Leftrightarrow$ Evolution map:

• $\forall t \in \mathbb{R}, y \in D$: $\frac{\partial \Phi}{\partial t}(t, y) = f(\Phi(t, y))$

• $\forall z \in D$: $f(z) = \frac{\partial \Phi}{\partial t}(t, (\Phi^t)^{-1}(z))$.

Bem. 68: M.2 Polygonal approximation methods

IVP: $\dot{y} = f(t, y)$, $y(t_0) = y_0$

Goal: (i) $y(T)$ for some "final time" $T > t_0$

(ii) approximate trajectory $t \mapsto y_h(t) \approx y(t)$.

Def.: (M.2.3) Temporal mesh

$M = \{t_0 \leq t_1 \leq \dots \leq t_{n-1} < t_n = T\} \subset [t_0, T]$.

Def.: M.2.1 Explicit Euler method

$\dot{y} = f(t, y) \Rightarrow f(t, y)$ gives slopes in t .

Recursion formula: $y_{k+1} = y_k + (t_{k+1} - t_k) f(t_k, y_k)$

$\frac{y_{k+1} - y_k}{t_{k+1} - t_k} = f(t_k, y_k) = h_k \cdot \text{time}$

$\Rightarrow \dot{y} = f(t, y)$

Def.: M.2.2 Implicit Euler method

$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$, with $h_k := t_{k+1} - t_k$

$(y_{k+1} - y_k) / (t_{k+1} - t_k) = f(t_{k+1}, y_{k+1})$, Diff. quotient.

Thm.: M.2.15 Implicit function thm.

Bem. 69: $\Rightarrow$ for small $|h|$: $G(h, y) = 0$ defines

a function $h \mapsto y(h)$: $G(h, y(h)) = 0$, i.e.,

$\Rightarrow y_{k+1}$ exists for sufficiently small h .

Def.: M.2.3 Implicit midpoint method

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

$(y_{k+1} - y_k) / (t_{k+1} - t_k) = f(\frac{1}{2}(t_k + t_{k+1}), \frac{1}{2}(y_k + y_{k+1}))$.

Bem. 70: timestepping: sequence $y_0, y_1, \dots, y_n$

Def.: M.3 General single step methods

recursion: $y_{k+1} = \Phi(h_k, y_k)$, $\Phi \hat{=}$ discrete evolution

$y(t_{k+1}) = \Phi(h, y(t_k))$, $\Phi \hat{=}$ evolution map

Goal: $\Phi \approx \Phi$

Def.: M.3.5 Single step method.

Bem. 71: (M.3.7) Consistent discrete evolution

Def.: M.3.2 Convergence.

setting: $\dot{y} = f(y)$, $y(0) = y_0$

Assume: $\|f(z) - f(w)\| \leq L \|z - w\| \quad \forall z, w \in D$

Error measure: $\max \|y_j - y(t_j)\|$

with $y_j \hat{=}$ sequence produced by SSM, $y(t_j) \hat{=}$ exact sol.

family of meshes: $(M_\ell)_\ell$: $h_\ell := \max |t_{j+1}^\ell - t_j^\ell| \rightarrow 0, \ell \rightarrow \infty$

Bem. 72: Algebraic conv. of SSM. (cf. Skript)

In general (for smooth trajectories) consistent

SSM converge algebraically for meshwidth

$h \rightarrow 0$: error $\in C(h^p)$, $p \in \mathbb{N}$, $p \hat{=}$ order of SSM.

Def.: M.3.2.1 Order of SSM.

Bem. 73: Convergence analysis for explicit euler m.

Bem. 74: For SSM: One-step-err $O(h^{p+1}) \rightarrow$ Order p .

M.4 Explicit Runge-Kutta SSM.

Bem. 75: Rationale for high order SSM: M.4.1

discretization error $\approx Ch^p$.

Goal: error reduction by factor $p > 1$ by h -refinement

$\frac{Ch_p}{Ch_{h_0}} = p \Rightarrow h_1 = p^{-1/p} \cdot h_0$

Effort $\sim \# \text{ timesteps} \sim h^{-1}$: $W_1 = p^{-1/p} W_0$

effort after refinement

effort before refinement

Def.: M.4.3 Explicit Runge-Kutta method

[...] $c_i := \sum_{j=1}^s a_{ij}$, s -stage explicit RK-SSM

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

$y_1 = y_0 + h \sum_{i=1}^s b_i k_i$, with $\Psi = \sum_{i=1}^s b_i k_i$, $k_i \hat{=}$ increments.

Bem. 76: Effort for one step: s f -eval.

Bem. 77: (M.4.11) Butcher scheme ... for explicit RK-SSM.

Bsp.: $\begin{bmatrix} 0 & 0 \\ a & 0 \end{bmatrix}$ applied to ODE $\dot{y} = f(y)$.

$k_1 = f(y_0)$, $k_2 = f(y_0 + h a k_1) = f(y_0 + h a f(y_0))$

$k_3 = f(y_0 + h a k_2) = f(y_0 + h a f(y_0 + h a f(y_0)))$

$\Rightarrow y_1 = y_0 + h(p k_1 + p k_2 + p k_3)$

$= y_0 + h p [f(y_0) + f(y_0 + h a f(y_0)) + f(y_0 + h a f(y_0 + h a f(y_0)))]$.

Bem. 78: RK-SSM consistent, if $\sum_{i=1}^s b_i = 1$.

Bem. 79: M.4.15 "Butcher barriers" for expl. RK-SSM

order $p \leq \#$ of stages of RK-SSM

Bem. 80: • EE, order = 1 • expl. trapezoidal rule, order 2

• expl. midpoint rule, order = 2 • Classical 4th order RK-SSM, ord. = 4

• Kutta's 3/3-rule ord. = 4.

M.5 Adaptive Stepsize Control

Bem. 81: Stepsize adaptation for SSM (M.5.8) $\rightarrow$ above!

[...] Choose h_k based on y_{k-1} and h_{k-1} alone.

(I) ESTIMATE of one-step error (M.5.8) [...]

$\Phi^h y(t_k) - \Phi^h y(t_k) \approx \text{EST}_k := \Phi^h y(t_k) - \Phi^h y(t_k)$

one-step error, Order(Φ^h) > Order(Φ).

(II) REFINE: $\text{EST}_k < \max(\text{atol}, \text{rtol} \cdot \|y_k\|)$?

(i) Accept step, goto next step with $h_{k+1} = \alpha h_k$, $\alpha > 1$

(ii) Reject step, repeat with $h_k \leftarrow \frac{1}{2} h_k$

Bem. 82: EST_k has next to nothing to do with

$y(t_k) - y_k$! It's only $\approx$ one-step error.

Bem. 83: No global err control through local-in-time adapt. timestepping

Bem. 84: sensitive dependence of step

position on $y(t)$ for small times, since we got

significant error with the initial large steps.

We can't recover because we're looking at the (usually)

local error only... (RK3 (usually) ord. = 3, resp. RK4 $\uparrow$ ord. = 4)

Bem.: (S1A2) We might see no relevant improvement

in the avg. orders of RK3 & RK4. This is due to the

fact that the r.h.s. of the IVP isn't continuously

diff'bar: the avg. theory breaks down!

$$M = \begin{pmatrix} 1 & -\lambda h a_{21} & -\lambda h a_{22} \\ 0 & 1 & -\lambda h a_{12} \\ 0 & -\lambda h a_{31} & -\lambda h a_{32} \end{pmatrix}$$

XII. Single step methods for stiff IVPs

Ex.: 12.1.1 Blow-up of explicit Euler method.
 $y_{k+1} = y_k + h \lambda y_k \Rightarrow y_k = (1 + h \lambda)^k y_0, k=1, \dots, N$
 blow-up $\Leftrightarrow |1 + h \lambda| > 1 \Rightarrow S(u)$

For $\lambda < 0$ we want $y_k \rightarrow 0 \Leftrightarrow |1 + h \lambda| < 1$.
 If $\lambda < 0, |h \lambda| > 2 = \text{blow-up}$ $h < \frac{2}{|\lambda|}$
 Necessary is $|h \lambda| \leq 2 \Rightarrow \text{timestep constraint}$

Bem. 85: Model problem analysis for general expl. RK-SSM.

(i) $b_i, k_i = \lambda(y_0 + h \sum_{j=1}^s a_{ij} k_j) \Rightarrow \sum_{j=1}^s m_{ij} k_j = \lambda y_0$
 with $m_{ij} = \sum_{l=1}^s \lambda h a_{il} a_{lj}$, if $j \leq i$. Let's do it for $i=3$:

$k_3 = \lambda(y_0 + h(a_{31} k_1 + a_{32} k_2))$, $(-\lambda h a_{31} - \lambda h a_{32})/k_1$
 $\Leftrightarrow (-\lambda h a_{31}) k_1 + (-\lambda h a_{32}) k_2 = \lambda y_0$

$\Rightarrow M k = \lambda y_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, with $M = (m_{ij})$ like above.
 $\Rightarrow M = I - \lambda h A \Rightarrow (I - \lambda h A) k = \lambda y_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix} (*)$

(ii) $y_1 = y_0 + h \sum_{j=1}^s b_j k_j \Rightarrow y_1 - h(b_1 k_1 + b_2 k_2 + \dots) = y_0$
 $\rightarrow -h(b_1, \dots, b_n)(k_1, \dots, k_n)^T = y_0 - y_1$
 $\Rightarrow (-\lambda h b_1 \quad \lambda)(k_1, y_1)^T = \lambda y_0 (**)$

$\Rightarrow \begin{pmatrix} -\lambda h b_1 & \lambda \\ -z b^T & \lambda \end{pmatrix} \begin{pmatrix} k_1 \\ y_1 \end{pmatrix} = \lambda y_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. $z = \lambda h (***)$

(iii) derive stability function:
 $\Leftrightarrow (I - zA) k = \lambda y_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Leftrightarrow k = (I - zA)^{-1} \lambda y_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$-z b^T k + \lambda y_1 = \lambda y_0 \Leftrightarrow y_1 = y_0 + h b^T k$ (II)
 $\Rightarrow y_1 = y_0 + h b^T (I - zA)^{-1} \lambda \begin{pmatrix} 1 \\ 1 \end{pmatrix} y_0$
 $\Leftrightarrow S(z) = (1 + z b^T (I - zA)^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}) y_0$

Thm.: 12.1.15 Stability function of expl. RK-SSM.

[...] $\psi_\lambda^h = S(\lambda h) \Leftrightarrow y_1 = S(\lambda h) y_0, z := \lambda h$
 stability function: $S(z) := 1 + z b^T (I - zA)^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = [\dots]$

Bem. 86: RK-sequences (uniform timestep $h > 0$): $y_k = S(z)^k y_0$.

Bem. 87: all stability functions are truncated

taylor expansion of $\exp$: $e^z = 1 + z + \frac{1}{2} z^2 + \frac{1}{6} z^3 + \frac{1}{24} z^4 + \dots$

Cor.: 12.1.18 Poly. stability fct of expl. RK-SSM consistent.

[...] $\Rightarrow |S(z)| \rightarrow \infty$ as $|z| \rightarrow \infty$

Def.: (12.1.20) Systems of ODEs.

$\dot{y} = M y, M \in \mathbb{R}^{d \times d}$. Solved by diagonalization:
 $V^{-1} M V = D = \text{diag}(\lambda_1, \dots, \lambda_d), z = V^{-1} y \Rightarrow \dot{z} = D z (**)$

(*) $\hat{=}$ decoupled scalar linear ODEs

Def.: (12.1.42) $f(y) = M y$, transformed increments:

$\hat{k}_i = V^{-1} k_i, \hat{z}_i = V^{-1} z_i$ decoupled
 $\hat{k}_i = D(z_0 + h \sum_{j=1}^s a_{ij} \hat{k}_j), \hat{z}_i = z_0 + h \sum_{j=1}^s b_j \hat{k}_j$ (12.1.44)
 $\Rightarrow (\hat{k}_i)_i = \lambda_i [(y_0)_i + h \sum_{j=1}^s a_{ij} (\hat{k}_j)_j], (\hat{z}_i)_i = (z_0)_i + h \sum_{j=1}^s b_j (\hat{k}_j)_j$
 for $i=1, \dots, d \Rightarrow y_k = V \text{diag}(y_k^{(1)}, \dots, y_k^{(d)}) V^{-1}$

Thm.: 12.1.46 (Abs) stability of expl. RK-SSM for linear systems of ODEs. $\Rightarrow$ **Def.: 12.1.43**

Region of (abs.) stability
 $[...] S_\eta := \{ z \in \mathbb{C} : |S(z)| < 1 \} \subset \mathbb{C}$. values

Bem. 88: Why $\mathbb{C}$? $\rightarrow$ because of complex EV of M .

Bem. 89: If S_η is bounded $\Rightarrow$ Timestep constraint

to avoid blow-up.

Bem. 90: $0(\text{zero}) \in \partial S_\eta$, because $S(z) = 1 + z + O(|z|^2)$

for consistent SSM. $|e^{\lambda t}| = e^{\text{Re}(\lambda)t}$

Note: $\text{Re}(\lambda)$ in $y(t) = e^{\lambda t}$ governs decay/growth:

Discussion: linear system: $\dot{y} = M y$, EVs $= \lambda_1, \dots, \lambda_d$:

• There is a $\lambda_j: \text{Re}(\lambda_j) > 0 \rightarrow$ blow-up of exact sol.

$\rightarrow$ Blow-up of (y_k) is even desirable, since $\rightarrow$

• $\text{Re}(\lambda_j) \leq 0 \forall j$ and $|\text{Re}(\lambda_2)| > 1$:

$\rightarrow$ Exact solution remains bounded.

$\rightarrow$ Blow-up of numerical solution disastrous!

$\rightarrow$ Avoid blow-up! $\Rightarrow$ time step constraint!

Notion: 12.2.3 Stiff IVP $\triangleq$ predicting

Bem. 91: Heuristic considerations for stiffness:

• Linear ODE: $\dot{y} = M y$: $\checkmark$

• general ODE: $\dot{y} = f(y)$, stiffness state dependent

stiff at state $y^* \in y(t)$? $\checkmark$

Tool: Linearization $f(y) \approx f(y^*) + Df(y^*)(y - y^*)$

Bem. 92: "Close" to y^* solutions of $\dot{y} = f(y)$ will

behave like solutions of the linearized ODE:

$\dot{z} = f(y^*) + Df(y^*)(z - y^*) \hat{=}$ affine linear ODE. (*)

Def.: (12.2.3) Linearization of expl. RK-SSM

$k_i := f(y^*) + Df(y^*) h \sum_{j=1}^s a_{ij} k_j$

$\hat{=}$ increment equation for RK-SSM applied to (*)

Bem. 93: for small timestep the behaviour of an

expl. RK-SSM applied to $\dot{y} = f(y)$ close to the state

y^* is determined by the EV of the Jacobian $Df(y^*)$.

Def.: How to distinguish stiff IVP. $\dot{y} = f(y)$

• $\min \text{Re}(\lambda): \lambda \in \sigma(Df(y(t))) \leq -\epsilon < 0$ and

• $\max \text{Re}(\lambda): \lambda \in \sigma(Df(y(t))) \leq 0 \Rightarrow$ probably stiff.

12.3 Implicit RK-SSM.

Ex: Euler methods for stiff decay IVP

Impl. Euler: $y_{k+1} = y_k + h f(y_{k+1})$

Lin. model/problem analysis: apply to $\dot{y} = \lambda y$

$\rightarrow y_{k+1} = \frac{1}{1 - h \lambda} y_k$ (uniform timestep h)

$\Rightarrow y_k = \left(\frac{1}{1 - h \lambda} \right)^k y_0$

If $\lambda < 0 \Rightarrow \left| \frac{1}{1 - h \lambda} \right| < 1 \Rightarrow \lim_{k \rightarrow \infty} y_k = 0$

$\Rightarrow$ No stability induced timestep constraint for all $h > 0$!

Bem. 94: For any timestep, the impl. Euler method

generates exponentially decaying solution

sequences $(y_k)_{k=0}^\infty$ for $\dot{y} = M y$ with diag'bar

matrix $M \in \mathbb{R}^{d \times d}$ with EV $\lambda_1, \dots, \lambda_d$, if $\text{Re}(\lambda_i) < 0 \forall i=1, \dots, d$.

Def.: (12.3.7) Collocation principle

$\hookrightarrow$ Skript: Abstract collocation idea.

First step of SSM for IVP: $\dot{y} = f(y), y(0) = y_0$

$\rightarrow y_1$ stepsize h .

(i) On $[0, h]$ approx. $y(t) \approx y_h(t) \in V, V \hat{=}$ finite-dim.

space of fct: $[0, h] \rightarrow \mathbb{R}^d$. std-choice: $V = (\mathbb{P}_s)^d, \dim(V) = s \cdot d$

(ii) Selection of y_h through collocation conditions

in equation $\begin{cases} y_h(0) = y_0, \\ y_h(s_j) = f(y_h(s_j)) \end{cases}$

for collocation points $0 \leq s_1 \leq \dots \leq s_s \leq h$

(iii) $y_1 = y_h(h)$

Bem. 95: Collocation points from reference interval $[0, 1]$:

$s_j = c_j h, 0 \leq c_1 \leq \dots \leq c_s \leq 1$.

Def.: (12.3.11) s-stage collocation single step method.

Bem. 96: Collocation SSM $\hat{=}$ a poly. quadrature formulas

Exp.: 12.3.15 Cvg of collocation SSM

(i) Equidistant collocation points $c_j = \frac{j-1}{s-1}, j=1, \dots, s$

$\Rightarrow$ alg. cvg. with the empiric rate $p \approx s$.

(ii) Gauss points in $[0, 1]$ as normalized collocation

points $c_j = \tau_j, s$

$\Rightarrow$ alg. cvg. with the empiric rate $p \approx 2s$.

$\Rightarrow$ Gauss collocation SSM.

$$A = \begin{pmatrix} p(x) & 1 & 0 & -q \\ 1 & p(x) & 1 & -q \\ 0 & 1 & p(x) & 1 \\ 0 & 0 & 1 & p(x) \end{pmatrix}, p(x) = 3x \|x\|_2$$

$$A = \begin{pmatrix} 2/h_1 & 1/h_1 & 0 & \dots & 0 \\ 0 & a_1 & b_1 & \dots & 0 \\ 0 & 0 & b_1 & a_2 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & b_{n-2} & a_{n-1} & b_{n-1} & 1/h_n \end{pmatrix}, a_i = \frac{2}{h_i} + \frac{2}{h_{i+1}}, b_i = \frac{1}{h_{i+1}}, i=1, \dots, n-1$$

(S2A4) reuse the LU factorisation:

```

fullPivLU<MatrixXd> LU = A.fullPivLU();
for (int l=0; l<n; ++l)
    X.col(l) = LU.solve(b); //with MatrixXd X, VectorXd b
    shift(b);
}

```

LU 2. Var

(S2A2) Partitioned Matrix - Block Matrices:

```

auto triR = R.template triangularView<Upper>();
Eigen: double s = 1./u.dot(triR.solve(v)); //s = 1/(uA^T v)
MatrixXd Ainv(n,n); A<R> v, u.transpose(), 0;
LU: VectorXd x = A.partialPivLU().solve(b); // b = (r, p)^T

```

3. Var

(S5A1) Lambda function: $F(x) = Ax + (c_1 e^{x_1}, \dots, c_n e^{x_n})^T$

```

auto f = [4, 4, 3] (const VectorXd& x) {
    VectorXd tmp = (A*x) + c.cwiseProduct(x.array().exp().matrix());
    return tmp;
};
using pair = pair<double, double>;
auto Order = [ ] (const pair &P, const pair &Q) -> bool {
    return P.first < Q.first;
};

```

return type

(S5A3) Newton (3D) (nodal analysis)

```

auto F = [4, 4, 3] (VectorXd u, double alpha, beta, Uin) {
    VectorXd f(3);
    f << 3*u(0)-u(1)-u(2), -u(0)+3*u(1)-u(2)-Uin,
        -u(0)-u(1)+3*u(2)+alpha*exp(beta*(u(2)-Uin)/(U1-1));
    return f;
};
auto DF = [4, 4, 3, 4, 4, 4] (VectorXd u) {
    MatrixXd dF(3,3);
    dF << 3-1, -1, 3-1, -1, -1, 3*alpha*beta*exp(beta*(u(2)-Uin)/(U1));
    return dF;
};

```

Newton:

```

srand((unsigned)int) time(0);
VectorXd U0 = VectorXd::Zero(3);
VectorXd U1 = VectorXd::Random(3);
double err = (U1-U0).cwiseAbs().norm();
for (int i=0; i<maxit; ++i) {
    f = F(Ui, alpha, beta, Uin); //with VectorXd f(3);
    dF = DF(Ui); //with MatrixXd dF(3,3);
    U1 = U0 - dF.partialPivLU().solve(f);
    err = (U1-U0).cwiseAbs().norm();
}
cout << "Error: " << F(U1, alpha, beta, Uin).norm() << endl;

```

LU 4. Var

(S5A2) sparse LU:

```

SparseLU<SparseMatrixDouble> Ax.lu;
Ax.lu.analyzePattern(Ax); Ax.lu.factorize(Ax);
VectorXd Ainv.u = Ax.lu.solve(b);
VectorXd Ainv.u = Ax.lu.solve(u);

```

Def: Stepsize prediction (Chapter 11)

$EST_k := \| \Phi^h y_k - \Phi^{h_k} y_k \|, h = h_k$

$\rightarrow Order(\Psi) = p \text{ \& } Order(\tilde{\Psi}) > p$

One-step errors:

$\| \Psi^h y_k - \Phi^h y_k \| = O(h^{p+1}) + O(h^{p+2})$

$\| \tilde{\Psi}^h y_k - \Phi^h y_k \| = O(h^{p+2})$ with $\Phi^h y_k \hat{=}$ exact sol.

for small timestep h_k , we can ignore $O(h_k^{p+2})$

$\Rightarrow EST_k = ch_k^{p+1}$ (i)

Goal (efficiency!): $EST_k = TOL$ (ii)

(i) $h_{new} = h_k \cdot \sqrt[p+1]{\frac{TOL}{EST}}$

(S5A2) solve $A(x)x = b \Leftrightarrow F(x) = A(x)x - b = 0$.

DF(x) = $A(x) + \frac{x x^T}{\|x\|_2}$ (quasi-linear system)

```

template<class Real, class Vector>
void newton_step(Func f, Real k, const Vector& b, Real x, Vector& x_new) {
    Vector x = x.normalized();
    SparseMatrix<double> Ax = A(x);
    DF = Ax + k.dot(x).normalized();
    SparseLU<SparseMatrixDouble> Ax.lu; //LU
    Ax.lu.analyzePattern(Ax); Ax.lu.factorize(Ax);
    VectorXd Ainv.u = Ax.lu.solve(b); //solve
    " Ainv.u = Ax.lu.solve(b);
    x_new = 2*x - Ainv.u + Ainv.u/(1 + x.dot(Ainv.u));
    " (x.dot(Ainv.u) - x.dot(b));
}

```

int main() {

int n=5; //dim=5;

int maxit=1000; double tol=1e-8;

VectorXd b = Vector::Random(n); x(n); x(2)=0(n);

Vector<double> err; VectorXd x0=b;

err.push_back((x-x0).cwiseAbs().norm());

for (int i=0; i<maxit; ++i) {

if (err.back() > tol; ++i) break; //naive

x = x0;

newton_step(A, b, x, x);

err.push_back((x-x0).cwiseAbs().norm());

for (int i=0; i<maxit; ++i) {

if (err.back() > tol; ++i) break; //naive

if (i==1) T.insert(1,1)=1;

if (i==1) T.insert(1,1)=1;

auto A = [4, 4, 3] (const VectorXd& x) -> SparseMatrix<double> {

double norm = x.norm();

for (int i=0; i<n; ++i) T.coeffRef(i,i) = 3+norm;

return T;

}

(S6A1) Atkinson-Deville.

```

double d.polval(VectorXd t, y, double x) {
    int n = y.size();
    VectorXd dy = VectorXd::Zero(n);
    for (int i=0; i<n; ++i) {
        for (int k=i-1; k>=0; --k) {
            double quot = (x-t[i])*(t[i]-t[k]);
            dy(k) = dy(k) + t[i]*(t[i]-t[k])*quot*(dy(k)-dy(i));
            y(k) = y(k) + t[i]*(t[i]-t[k])*quot;
        }
    }
    return dy(0);
}

```

(S6A2) (code 3.1.18) p.w. linear interpolation

```

class LinearInterpolant {
public:
    LinearInterpolant(Vector<pair<double, double>> points) {
        data_points = points;
        assert( "size() > 1 & k < min. two points" );
        auto Order = [ ] (const pair<double, double> &P, &Q) -> bool {
            return P.first < Q.first;
        };
        sort(data_points.begin(), data_points.end(), Order);
        double operator() (double x) {
            auto Compare = [ ] (const pair<double, double> &P, double val) -> bool {
                return P.first < val;
            };
            if (it == data_points.begin() || it->first > x) return first;
            if (it == data_points.end()) return second;
            double res = (x-it->first)*(it->second-(it->first)) / (it->second-(it->first));
            return res;
        }
    }
};

```

(S6A1) Natural cubic Splines

```

private:
    vector<pair<double, double>> data_points;
int main() { LinearInterpolant I = LinearInterpolant( points );
}
class Batsci {
public:
    pair<const vector<double> &t, &y> VectorXd b(n);
    t=t; y=y; n=t.size();
}

```

```

b(0) = 3*(y[1]-y[0])/((t[1]-t[0])*(t[1]-t[0]));
for (int i=1; i<n-1; ++i) {
    b(i) = 3*(y[i+1]-y[i])/((t[i+1]-t[i])*(t[i+1]-t[i])) +
        3*(y[i]-y[i-1])/((t[i]-t[i-1])*(t[i]-t[i-1]));
}
b(n-1) = 3*(y[n]-y[n-1])/((t[n]-t[n-1])*(t[n]-t[n-1]));
vector<double> acoeff, bcoeff; //compute a_i, b_i
for (int i=0; i<n; ++i) {
    bcoeff.push_back( (t[i+1]-t[i]) );
    acoeff.push_back( (t[i+1]-t[i])*(t[i+1]-t[i]) );
}
SparseMatrix<double> A(n,n); A.reserve(2);
A.insert(0,0) = 2*bcoeff[0]; //compute A
A.insert(0,1) = bcoeff[0];
for (int i=1; i<n-1; ++i) {
    A.insert(i,i-1) = bcoeff[i-1];
    A.insert(i,i) = 2*bcoeff[i];
    A.insert(i,i+1) = bcoeff[i];
}
A.insert(n-1,n-2) = bcoeff[n-2];
A.insert(n-1,n-1) = 2*bcoeff[n-1];
A.makeCompressed();
SparseLU<SparseMatrixDouble> solver; //compute c
solver.analyzePattern(A); solver.compute(A);
c = solver.solve(bcoeff);
double operator() (double x) const { //eval. s(x) [see Batsci]
    assert( (t[0] <= x <= t[n-1]) && "not in [t0, t_{n-1}]" );
    auto j = lower_bound(t.begin(), t.end(), x) - t.begin();
    if (j==0) return t[0];
    double h_j = t[j]-t[j-1];
    " tau = (x-t[j-1])/h_j; tau2 = tau*tau; tau3 = tau*tau*tau;
    " s(x) = y[j-1]*(1-tau)^3 + 3*tau*(y[j]-y[j-1])*(1-tau)^2 +
        3*tau^2*(y[j]-y[j-1])*(1-tau) + y[j];
    return s(x);
}

```

(S6A3) discrete Legendre inner product

```

int main() {
    double pi = atan(1)*4; // how to do pi in c++
    auto ch2.zeros = [4, pi] (const int& k, int const& j) {
        return cos((2*j+1)*(2*pi*(k+1)*pi));
    };
    int n=5; d=2;
    vector<double> V; MatrixXd scal(d,d);
    for (int j=0; j<d; ++j) {
        V = ch2.polmult(d, ch2.zeros(d,j));
        for (int k=0; k<d; ++k) scal(j,k) = V[k];
    }
    for (int k=0; k<d; ++k) {
        for (int l=0; l<d; ++l) {
            double res = 0;
            for (int i=0; i<n; ++i) {
                res += V[i]*scal(i,k)*scal(i,l);
            }
            cout << scal.col(k).dot(scal.col(l)) << endl;
        }
    }
}

```

(S6A4) p.w. cubic Hermite interpolation

```

class PCH {
public:
    PCH(const VectorXd& t, &y): t(t), y(y), c(t.size()) {
        n=t.size(); h=t[1]-t[0];
        c(0) = (y[1]-y[0])/h;
        for (int i=1; i<n-1; ++i) {
            c(i) = (y[i+1]-y[i])/h;
            c(n-1) = (y[n]-y[n-1])/h;
        }
        VectorXd operator() (VectorXd x) const { //eval interpolant at x
            VectorXd ret(x.size());
            double tmp, t1, y1, y2, c1, c2, a1, a2, a3;
            for (int i=0; i<n; ++i) {
                if (t[i]-x) < 0 { //Matlab code 3.4.6
                    t1 = t[i-1]; y1 = y[i-1];
                    y2 = y[i]; c1 = c[i-1]; c2 = c[i];
                    a1 = y2-y1; a2 = a1-h*c1; a3 = h*c2-a1-a2;
                    tmp = (x-t1)/h;
                    ret(i) = y1 + (a1*(a2+a3+tmp)*(tmp-1) + tmp)*tmp;
                }
            }
            return ret;
        }
    }
};

```


Thm. 12.3.17 Order of collocation SSM.

Bem. 37: Many of the techniques and much more of the theory for expl. RK-SSMs carry over to general (impl.) RK-SSMs:

- Sufficient conditions for consistence (Cor. 11.4.12).
- Alg. cvg. for meshwidth $h \rightarrow 0$ and related concept of order (Def. 11.3.21).
- Embedded methods and algorithms for adaptive stepsize control from Section 11.5.

Def. 12.3.18 General RK-SSM

[..] $k_i = f(t_i + c_i h, y_0 + h \sum_{j=1}^s a_{ij} k_j)$, $i=1, \dots, s$, $y_1 = y_0 + h \sum_{i=1}^s b_i k_i$.

Def. 12.3.20 General butcher scheme

Def. 12.3.21 Stage form eq. for increments

[..] instead of increments we consider stages.

- Recap:
- Stiffness
 - impl. Euler methods is unconditionally stable for $\dot{y} = \lambda y$, $\lambda < 0$.
 - Collocation RK-SSM (incl. impl. Euler: $s=1, c_j=1$)

Thm. 12.3.27 Stability function

[..] $S(z) = 1 + z b^T (I - zA)^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = [\dots]$, $z := \lambda h$

Thm. 12.3.35 Region of stability [..]

[..] s -stage Gauss collocation SSM stability domain: $S_q = C^- \hat{=} A$ -stable

Bem. 38: A-stable = unconditionally stable for decay eq.

Def. 12.3.32 A-stability of a RK-SSM

[..] $C^- := \{z \in \mathbb{C} : \operatorname{Re}(z) < 0\}$, S_q = stability region.

Bem. 33: A-stable RK-SSM won't be affected by stability induced timestep constraints when applied to stiff IVP.

Def. 12.3.38 L-stable RK-method

[..] (i) $\operatorname{Re}(z) < 0 \Rightarrow |S(z)| < 1$

(ii) $\lim_{\operatorname{Re}(z) \rightarrow -\infty} S(z) = 0$.

Bem. 100: L-stability = A-stability + ($S(\infty) = 0$).

Bem. 101: Construction of L-stable methods:

$$S(z) = 1 + z b^T (I - zA)^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1 + b^T (\frac{1}{z} I - A)^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\Rightarrow S(\infty) = 1 - b^T A^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \hat{=} 0$$

$$\Rightarrow b = (A)_{ji} \quad (j\text{-th row of } A)$$

$$\Rightarrow b^T A^{-1} = e_i^T \Rightarrow b^T A^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1.$$

Bem. 102: 12.3.4.1 Necessary condition for

L-stability of RK-methods. $(A)_{ij} = \delta_{ij}$
 $\Rightarrow$ new Butcher scheme, see the difference!

12.4 Semi-implicit RK-SSM

Idea: Fixed small number of Newton steps to compute increments

$\Rightarrow$ Linearize increment equation

Ex.: Impl. Euler for $\dot{y} = f(y)$: $y_1 = y_0 + h f(y)$; Ord. 1

Linearization: $y_1 = y_0 + h [f(y_0) + Df(y_0)(y_1 - y_0)]$

$\Rightarrow y_1 = y_0 + \frac{h}{1 - h Df(y_0)} f(y_0) \hat{=} \text{semi-impl. Euler SSM}$
 invertible for sufficiently small h

• Semi-implicit midpoint method: Order 2

$$y_1 = y_0 + h f(\frac{1}{2}(y_0 + y_1))$$

$$\Rightarrow y_1 = y_0 + h [f(y_0) + \frac{1}{2} Df(y_0)(y_1 - y_0)]$$

Bem.: RADAU vs. Semi-impl. RADAU

Loss of order due to linearization!

Bem.: How to explore L-stability of a ROW method?

(i) Apply SSM to $\dot{y} = \lambda y$ [$f(y) = \lambda y$] (plug it into (12.4.6))

(ii) We get $y_1 = S(\lambda h) y_0$ (More general

$Q(\lambda h) y_1 = P(\lambda h) y_0$, P, Q polynomials)

(iii) verify $\lim_{z \rightarrow \infty} S(z) = 0$

(iv) show $\operatorname{Re}(z) < 0 \Rightarrow |S(z)| < 1$, most challenging step.

Summary: • Stiff IVP • A- & L-stability

• Stability induced timestep constraint

• Methods: (semi-) implicit RK-SSM usually in embedded form.

Bsp.: (SIRBA4) $y_1 = y_0 e^{\lambda h}$: $z := \lambda h$; stability plot

$$S(z) = \sum_{k=0}^{\infty} \frac{z^k}{k!} = e^z \quad \operatorname{Re}(z) < 0 \Rightarrow |e^z| < 1$$

$$= \sum_{k=0}^{\infty} \frac{z^k}{k!} \quad \operatorname{Re}(z) < 0 \Rightarrow |S(z)| < 1$$

Bsp.: Conversion to 1st order ODE (SIRBA4)

$$\dot{u}_i = v_i, \quad i=1, \dots, n$$

$$2\dot{v}_1 - \dot{v}_2 = u_1(u_2 + u_1), \quad i=1$$

$$-\dot{v}_{i-1} + 2\dot{v}_i - \dot{v}_{i+1} = u_i(u_{i-1} + u_{i+1}), \quad i=2, \dots, n-1$$

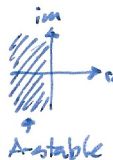
$$2\dot{v}_n - \dot{v}_{n-1} = u_n(u_n + u_{n-1}), \quad i=n$$

$$\Rightarrow C \dot{v} = C \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} u_1(u_2 + u_1) \\ u_2(u_{i-1} + u_{i+1}) \\ \vdots \\ u_n(u_n + u_{n-1}) \end{pmatrix} = g(u)$$

$$\text{mit } C = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & \dots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \Rightarrow \dot{y} = f(y) = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} y_{n+1} \\ y_{2n} \end{pmatrix}$$

```
(SIRBA4): int main() {
    double T=1; int n=5; VectorXd y0(2*n);
    for(int i=0; i<n; ++i) {
        y0(i) = (i+1)/n; y0(i+n) = -1;
    }
    auto f = [&n] (VectorXd y) {
        VectorXd f(2*n); g(n); g(0) = y(0)*(y(n)+y(0));
        g(n) = y(n)*(y(0)+y(n));
        for(int i=1; i<n-1; ++i) {
            f(i) = y(i)*(y(i-1)+y(i+1));
            f(i+n) = -2*y(i);
        }
        for(int i=0; i<n; ++i) {
            C.insert(i,i) = 2;
            if(i<n-1) C.insert(i,i+1) = -1;
            if(i>0) C.insert(i,i-1) = -1;
        }
        C.makeCompressed();
        f.head(n) = y.head(n);
        SparseLU<SparseMatrix<double>> solver;
        solver.analyzePattern(C); solver.compute(C);
        f.tail(n) = solver.solve(g);
        return f;
    };
    // use RK45C solver now...
    (SIRBA4) #include "ode45.hpp"
    MatrixXd matode(const MatrixXd &A, double T) {
        auto F = [&A] (const MatrixXd &h) { // solves IVP y' = -A*y, y(0) = y0
            return -(A - A.transpose()) * h;
        };
        ode45<MatrixXd> O(F);
        O.options().set("RelTol", 1e-8);
        return O.solve(y0, T).back().first(); // solution at (T)
    };
    bool checkInvariant(const MatrixXd &A, double T) { // check if invariant!
        MatrixXd h(3,3); h = matode(A, T);
        if(h.transpose() * A * h.transpose() != A).norm() < 1e-10 {
            return true; // invariant!
        } else {
            return false; // not invariant!
        }
    };
    MatrixXd invmat(const MatrixXd &A, double h) {
        return (MatrixXd::Identity(3,3) - h*A).partialPivLU().solve(y0 + h*A*y0);
    };
}
```

C++/Eigen:
 $B = \text{MatrixXd::Map}(A.data(), m, n);$
 $B = \text{Map}(MatrixXd(A.data(), m, n));$ $\leftarrow B = \text{reshape}(A, m, n)$ // Matlab
 $\lceil \log_2(k) \rceil \Rightarrow \text{ceil}(\log_2(k))$ // iterate throughout
 $\text{vector<double> bin} = \{2, \dots, 3\}$ // iterate throughout




```

(S1043) Fortsetzungen... // quadrature on [a,b], but
template <class func> // weights brokes were on [1,1]
double gaussquad(func &f, const VectorXd &w, double a, b) {
    double I = 0;
    QuadRule Q = QuadRule(2); // w() = Q.weights
    for(int i=0; i<w.size(); ++i) {
        I += f((x(i)+a)*(b-a)/2+a)*w(i); // no gauss.hpp
    }
    return I*(b-a)/2;
}
template <class func> // compute S00 using subot. x=colld
double quadint(int n, func &f) { // n = # Gauss points
    VectorXd w, x;
    golubwelsch(n, w, x);
    auto ftilde = [f] (double x) {
        return f((x-a)*(b-a)/2+a);
    };
    return quad(ftilde, w, x, 0, 1);
}
(S1044) Nested numerical Quadrature (double integral)
gauleg.hpp {
    struct QuadRule {
        VectorXd weights, nodes;
    };
    QuadRule gauleg(unsigned int n); // this is Golub-Welsch
    #include "gauleg.hpp"
    template <class func>
    double gaussquadtriangle(func &f, unsigned int k) {
        QuadRule Q = gauleg(k);
        auto g = [f, Q] (double x) {
            return evalgaussquad(f, Q, x);
        };
        return evalgaussquad(g, Q, 0, 1);
    }
}
(S1045) RK-method
template <class State> // e.g. VectorXd
class RKIntegrator {
public:
    RKIntegrator(const MatrixXd &A, const VectorXd &b) {
        m(A), b(b), s(b.size());
    }
    template <class Function>
    vector<State> solve(const Function &f, double T,
        const State &y0, unsigned int N) const {
        double h = T/N;
        vector<State> res; res.reserve(N);
        res.push_back(y0);
        state ytemp = y0; ytemp2 = y0;
        for(int k=0; k<N; ++k) { // loop over fixed steps
            step(f, h, ytemp, ytemp2); res.push_back(ytemp);
            swap(ytemp, ytemp2);
        }
        return res;
    }
private:
    template <class Function>
    void step(const Function &f, double h, const State &y,
        State &ytemp, State &ytemp2) const {
        ytemp = y;
        for(int i=0; i<s; ++i) { // loop over RK
            state inc = y0; // compute increments
            for(int j=0; j<s; ++j) inc += h*A(j,i)*k.at(j);
            k.push_back(f(inc));
            ytemp += h*k.at(i);
        }
        ytemp2 = ytemp;
    }
    const MatrixXd A;
    const VectorXd b; unsigned int s;
}
int main() { // avg study
    double errtol; RKIntegrator RK(h, b);
    for(int i=0; i<N; ++i) {
        auto res = RK.solve(f, T, y0, N);
        double err = (res.back()-fexact).norm();
        if(i>0) {
            cout << log(errtol/err)/log(2) << endl;
            errtol = err; // 1/2 r estimat
        }
    }
}
(S1046) QR decomposition
Householder QR(MatrixXd &A) { q = Householder(A);
    q.compute(A);
    MatrixXd Q = q.householder(Q);
}
Adaptive Quadrature:
template <class func>
double adaptquad(func &f, VectorXd &w, double atol, rtol) {
    int n = w.size();

```

```

    VectorXd h = (b-a).array().asMatrix();
    " mp = "
    " traploc(n) simploc(n-1);
    for(int i=0; i<n-1; ++i) {
        traploc(i) = (h(i)+h(i+1))/2*(f(h(i))+f(h(i+1))+2*f(h(i+1/2)));
        simploc(i) = (h(i)+h(i+1))/3*(f(h(i))+f(h(i+1))+4*f(h(i+1/2)));
        VectorXd estloc = (simploc.array()-traploc.array()).abs().matrix();
        double esttot = estloc.sum(); // Is = simploc.sum();
        if(esttot < atol) return I;
        VectorXd Mn(2*n-1); Mn.head(n) = h.head(n);
        int count = 0; double avgloc = 0; double esttot2 = (n-1);
        for(int i=0; i<n-1; ++i) {
            if(estloc(i) > avgloc) Mn(count++) = mp(i); count++;
            cout << "new steps added: " << count << endl;
            Mn = Mn.head(n+count);
            sort(Mn.data(), Mn.data()+Mn.size());
            return adaptquad(f, Mn, atol, rtol);
        }
    }
}
Mean square best approx.
VectorXd eval_bj(const VectorXd &h, double b, int j) {
    int n = h.size(); // 3-term rec: 4.2.3.1
    VectorXd b0(n); b0.setOnes();
    if(j==0) return b0;
    VectorXd b1(n); b1 = (b.array()-a(0)).matrix();
    if(j==1) return b1;
    VectorXd b2(n);
    for(int i=0; i<n; ++i) {
        b2(i) = (b1.array()-a(i)).matrix().cwiseProduct(b1).array()
            - (b1(i-1)*b1(i)).matrix();
        b0 = b1; b1 = b2;
    }
    return b2;
}
template <class func> // eval f
VectorXd eval_f(func &f, const VectorXd &h) {
    int n = h.size(); VectorXd fx(n);
    for(int i=0; i<n; ++i) fx(i) = f(x(i));
    return fx; // Thm 4.2.3.1
}
void coefforth(const VectorXd &T, double b, int k) {
    int m = T.size(); a(0) = T.sum()/m;
    VectorXd b0(m); b0 = b.setOnes();
    " b1(m); b1 = b0.setOnes();
    " b2(m); b2 = (T.array()-a(0)).matrix();
    for(int i=0; i<k; ++i) {
        b0 = b1; b1 = b2;
        a(i) = b1.dot(b0.cwiseProduct(T))/pow(b1.norm(), 2);
        b1 = (b0.array()-a(i)).matrix().cwiseProduct(b0).array();
        b2 = (b1.array()-a(i)).matrix().cwiseProduct(b1).array();
        b1 = (b1-i*b2).array(); // matrix(); return;
    }
}
template <class func> // eval coeff f at x
VectorXd coeff_c(func &f, VectorXd &T, double b, int k) {
    VectorXd coeff(k); f_i = eval_f(f, T);
    for(int j=0; j<k; ++j) {
        VectorXd b_j = eval_bj(a, b, T, j);
        coeff(j) = f_i.dot(b_j)/b_j.dot(b_j);
    }
    return coeff;
}
template <class func> // eval coeff f at x
VectorXd bestapprox(func &f, const VectorXd &T,
    const int k, VectorXd &h) {
    VectorXd a(k), b(k-1);
    coefforth(a, b, k);
    VectorXd c(k); c = coeff_c(f, T, a, b, k);
    int numEvals = x.size(); res.setZero();
    VectorXd res(numEvals);
    for(int i=0; i<numEvals; ++i) res += c(i)*eval_bj(a, b, x, i);
    return res;
}
int main() {
    fx = bestapprox(f, T, k, x);
}

```