

Polynomiale Interpolation (PI)

Ziel: Gegeben sind $n+1$ paarweise verschiedene Stützstellen $x_0, x_1, \dots, x_n$ und zugehörige Stützwerte $y_0, y_1, \dots, y_n$.

Finde das Interpolationspolynom (IP) n -ten Grades,

$$p_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n \in \mathbb{P}_n \quad n = \text{Grad des IP}$$

welches die Interpolationsbedingungen (IB) erfüllt

$$p_n(x_j) = y_j, \quad x_j$$

L1: Löse das LGS

$$\begin{matrix} \text{I:} & \begin{bmatrix} 1 & x_0^n & \dots & x_0^0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^n & \dots & x_n^0 \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} \\ \text{II:} & a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n = y_0 \\ \vdots & \\ \text{III:} & a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n = y_1 \\ \vdots & \end{matrix}$$

Indem man einfach "Gauß". (Vektor mit a_i bleibt unverändert)

L2: Lagrange'schen Interpolationsformel (LI)

$$p_n(x) = \sum_{j=0}^n y_j L_j^n(x)$$

$$\text{wobei } L_j^n(x) = \prod_{\substack{i=0 \\ i \neq j}}^n \frac{x - x_i}{x_j - x_i} \quad (j=0,1,\dots,n)$$

die Lagrange Polynome (LP) sind LP Eigenschaften:

(LP1) $L_j^n(x_i)$ sind Polynome n -ten Grades

$$(LP2) L_j^n(x_k) = \delta_{jk} = \begin{cases} 1, & j=k \\ 0, & j \neq k \end{cases}$$

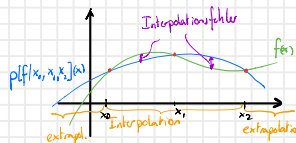
Interpolationsfehler

Für ein f $(n+1)$ -mal diffbar.

für jedes $x \in I \exists \xi(x)$ mit:

$$e(x) = f(x) - p[f](x_0, \dots, x_n)(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^n (x - x_j) \quad (\text{Fehlerfunktion})$$

$$\|e\|_{\infty} = \max_{x \in I} |e(x)| \leq \frac{\|f^{(n+1)}\|_{\infty}}{(n+1)!} (b-a)^{n+1} \quad (\text{grösste Fehler})$$



Numerische Integration = Quadratur

Ziel: Approximation von $I[f] = \int_a^b f(x) dx \Rightarrow$ Lsg. Integriere IP.

$$\text{Quadraturregel (QR): } Q[f] = \sum_{j=0}^n w_j f(x_j) \quad \begin{matrix} x_j: \text{Quadraturknoten} \\ w_j: \text{Quadraturgewichte} \end{matrix}$$

$$\text{Herleitung: } \int_a^b f(x) dx \approx \int_a^b p[f](x_0, \dots, x_n)(x) dx = \int_a^b \sum_{j=0}^n f(x_j) L_j^n(x) dx = \sum_{j=0}^n f(x_j) \int_a^b L_j^n(x) dx = \sum_{j=0}^n f(x_j) \cdot w_j = Q_n[f]$$

$$w_j = \int_a^b L_j^n(x) dx : \text{unabhängig von } f, \text{ d.h. für bekannte Knoten kann man sie ein für alle mal berechnen}$$

Newton-Cotes auf $I=[a,b]$

Mittelpunktsregel ($n=0$) (MR)

$$x_0 = \frac{a+b}{2}$$

$$L_0^n(x) = 1$$

$$w_0 = \int_a^b L_0^n(x) dx = b-a$$

Trapezregel ($n=1$) (TR)

$$x_0 = a, x_1 = b$$

$$L_0^n(x) = \frac{x-b}{a-b}, L_1^n(x) = \frac{x-a}{b-a}$$

$$w_0 = \int_a^b L_0^n(x) dx = \frac{b-a}{2}$$

$$w_1 = \frac{b-a}{2}$$

Simpsonregel ($n=2$) (SR)

$$x_0 = a, x_1 = \frac{a+b}{2}, x_2 = b$$

$$L_0^n(x) = \frac{x - \frac{a+b}{2}}{a - \frac{a+b}{2}} \cdot \frac{x-b}{\frac{a+b}{2} - b} = \frac{x-b}{a-b} \cdot \frac{x - \frac{a+b}{2}}{\frac{a-b}{2}} = \frac{2(x-b)(x - \frac{a+b}{2})}{(a-b)^2}$$

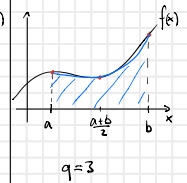
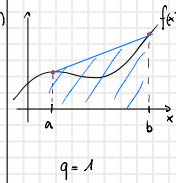
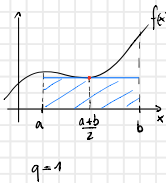
$$L_1^n(x) = \frac{x-a}{\frac{a+b}{2} - a} \cdot \frac{x-b}{\frac{a+b}{2} - b} = \frac{x-b}{\frac{a+b}{2} - a} \cdot \frac{x-a}{\frac{a+b}{2} - b} = \frac{4(x-a)(x-b)}{(a-b)^2}$$

$$w_0 = w_2 = \frac{b-a}{6}, w_1 = \frac{4(b-a)}{6}$$

$$Q_0[f] = (b-a) f\left(\frac{a+b}{2}\right)$$

$$Q_1[f] = \frac{(b-a)}{2} (f(a) + f(b))$$

$$Q_2[f] = \frac{b-a}{6} \left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$$



Quadraturfehler (QF)

$$\text{Quadraturfehler: } |E[f]| = |Q[f] - I[f]|$$

Genauigkeitsgrad (q): Höchster Grad q der monom den man exakt integrieren kann. $Q[x^q] = I[x^q], Q[x^{q+1}] \neq I[x^{q+1}]$

- Dank Variablentransformation darf man einfach auf RI $[-1,1]$ überprüfen

Auf RI gilt: $I[1] = 2, I[x] = 0, I[x^2] = \frac{2}{3}, I[x^3] = 0, I[x^4] = \frac{2}{5}$

- Je grösser der GG desto genauer ist eine QR vorausgesetzt!

Für den QF: $E[f] \leq \frac{\|f^{(q+1)}\|_{\infty}}{(q+1)!} (b-a)^{q+1} \quad (QF.1)$

- Für NC, GG ist mindestens $n/2$ falls n ungerade/gerade

Ordnung s einer QR: $s = q+1$

Manchmal nennt man es Konsistenzordnung

- Mit einer GG von q kann man nur Polynome bis Grad q exakt integrieren. Variable substitution - Intervall transformation (aber keine Wertordnung)

Ziel: beliebige Intervall mit Hilfe einer QR der auf ein Referenzintervall (RI) $I=[-1,1]$ basiert ist, integrieren. i. Transformation finden: $x = \frac{b-a}{2}t + \frac{a+b}{2} \quad t \in [-1,1]$

$$\frac{dx}{dt} = \frac{(b-a)}{2}$$

$$\begin{aligned} \text{ii. In Integral einsetzen: } I_{[a,b]}[f] &= \int_a^b f(x) dx = \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) \frac{b-a}{2} dt \\ \tilde{f}(t) &= \frac{b-a}{2} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) \\ I_{[a,b]}[f] &= \int_{-1}^1 \tilde{f}(t) dt = I_{[-1,1]}[\tilde{f}] = Q_{[-1,1]}[\tilde{f}] \end{aligned}$$

In das Intervall $I=[0,1]$

$$\text{i. } x = a + (b-a)t, \quad dx = (b-a) dt$$

$$\text{ii. } I_{[a,b]}[f] = \int_a^b f(x) dx = \int_0^1 f(a + (b-a)t) (b-a) dt = \int_0^1 \tilde{f}(t) dt = Q_{[0,1]}[\tilde{f}]$$

$$\tilde{f}(t) = (b-a) \cdot f(a + (b-a)t)$$

Summierte Quadraturregel (SQR)

Das Intervall $I=[a,b]$ wird in N Teil-Intervalle $I_j = [x_{j-1}, x_j]$ ($j=1, \dots, N$) zerlegt mit: $x_j = a + j \cdot h, (j=0,1,\dots,N) \quad h = \frac{b-a}{N} = x_j - x_{j-1}$

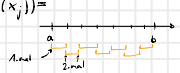
Wendet man die QR auf die Teilintervalle an und summiert

Summierte MR (SMR):

$$Q_0^N[f] = \sum_{j=1}^N Q_0[f \text{ auf } I_j] = \sum_{j=1}^N h \cdot f\left(\frac{x_{j-1} + x_j}{2}\right)$$

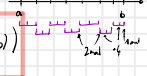
Summierte TR (STR):

$$\begin{aligned} Q_1^N[f] &= \sum_{j=1}^N Q_1[f \text{ auf } I_j] = \sum_{j=1}^N \frac{h}{2} (f(x_{j-1}) + f(x_j)) \\ &= \frac{h}{2} (f(a) + 2 \sum_{j=1}^{N-1} f(x_j) + f(b)) \end{aligned}$$



Summierte SR (SSR):

$$\begin{aligned} Q_2^N[f] &= \sum_{j=1}^N Q_2[f \text{ auf } I_j] = \sum_{j=1}^N \frac{h}{6} (f(x_{j-1}) + 4f\left(\frac{x_{j-1} + x_j}{2}\right) + f(x_j)) \\ &= \frac{h}{6} (f(a) + 2 \sum_{j=1}^{N-1} f(x_j) + 4 \sum_{j=1}^{N-1} f\left(\frac{x_{j-1} + x_j}{2}\right) + f(b)) \end{aligned}$$



Summierte 2-Punkt GLQ:

$$SG_N[f] = \sum_{j=1}^N G_j [f \text{ auf } I_j] = \sum_{j=1}^N \frac{h}{2} \left[f\left(\frac{1+\sqrt{3}}{2}x_j + \frac{b-a}{2}\right) + f\left(\frac{1-\sqrt{3}}{2}x_j + \frac{b-a}{2}\right) \right]$$

QF von SQR:

$$E^N[f] = |I[f] - Q_N[f]| = \left| \sum_{j=1}^N I[f \text{ auf } I_j] - Q_N[f \text{ auf } I_j] \right| \leq \sum_{j=1}^N E[f \text{ auf } I_j] \leq \frac{\|f^{(q+1)}\|_{\infty}}{(q+1)!} h^{q+1} (b-a) = \frac{\|f^{(q+1)}\|_{\infty}}{s!} (b-a) h^s$$

Landau-Notation: $e = O(h^s)$ falls $1 \leq s \leq C h^p$ für $C, p \in \mathbb{R}_+$
 $E^N[f] = O(h^{q+1}) = O(h^s)$

Die Ordnung s kann man dann experimentell bestimmen, indem man die Steigung des Fehlers gegen # Teilintervalle auf einer log-log plot abliest.

$$E^N[f] = O(h^s) \quad \log(E^N[f]) = s \cdot \log(h) \quad \leftarrow \text{Verify}$$

Konvergenzordnung einer Quadratur

quadratur = @sum simpsonregel; % holt quadratur aus anderen dts
 $a = 0;$
 $b = 1;$
 integrand = @(x) x.^5; % definiere integrationsfunktion
 reftol = 1e-6; % bestimme exakte Lösung
 $N = 2 \cdot 2^{(1:b)};$ % Vektor mit # Teilintervalle
 approxsol = zeros(1, length(N)); % alloziere Speicherplätze
 for ii = 1: length(N);
 approxsol(ii) = quadratur(integrand, a, b, N(ii)); % Approx. Log. mit N(ii) Subintervalle
 end
 err = abs(approxsol - reftol); % Absoluten Fehler
 fig = loglog(N, err, 'x');
 $p = \text{polyfit}(\log(N), \log(err), 1);$ % bestimme Konvergenzrate
 fprintf('Die Konvergenzordnung lautet %1.2f \ln', -p(1));

Summierte Quadraturformel (2-Punkt GLQ)

function y = sum2punktgauss(f, a, b, N)

$y = 0;$
 $x = \text{linspace}(a, b, N+1);$
 $h = (b-a)/N;$
 for ii = 1:N
 $y = y + h/2 * (f(-h/\sqrt{3}) + 0.5 * (x(ii) + x(ii+1))) +$

$f(h/\sqrt{3}) + 0.5 * (x(ii) + x(ii+1)));$

end

Bem.: integrand muss genug kompiliert sein damit es nicht exakt von QR berechnet werden können.

- Falls die zu integrierende Funktion nicht glatt ist, dann ist die Konvergenz beschränkt.

Gauss Quadratur

Bei NC QR sind die Knoten äquidistant, jetzt will man die n Knoten wählen so, dass die grösstmögliche GG erreicht wird.

Grösstmögliche GG ist $q = 2n+1$.

Diese GG kann man mithilfe der Orthogonalität erreichen.

Dafür verwendet man die Legendre-Polynome, die eine orthogonale Basis bilden (d.h. $\langle P_n(x), P_j(x) \rangle = 0$ für $j \neq n$).

$$P_0(x) = 1, P_1(x) = x, P_2(x) = \frac{3}{2}x^2 - \frac{1}{2}, P_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x, \dots$$

$$P_{j+1}(x) = \frac{2j+1}{j+1} x \cdot P_j(x) - \frac{j}{j+1} P_{j-1}(x) \quad j \geq 1$$

Und wir wählen die Knoten x_i als die Nullstellen von $P_{n+1}(x)$.

Gauss (-Legendre) Quadratur

Die $(n+1)$ -Punkte Gauss-Legendre Quadratur (GLQ) auf dem RI $[-1, 1]$ ist gegeben durch

$$G_n[f] = \sum_{j=0}^n w_j \cdot f(x_j)$$

wobei die Gauss-Punkte x_j die Nullstellen des $(n+1)$ -ten Legendre-Polynoms $P_{n+1}(x)$ und die Gewichte sind:

$$w_j = \frac{2(1-x_j^2)}{((n+1)P_n(x_j))^2}, \quad j = 0, 1, \dots, n$$

Sie hat die grösstmögliche GG $q = 2n+1$ und damit Ordnung $s = 2n+2$

Bem.: - Gewichte sind positiv

- Hohe Ordnung bedeutet nicht zwingend hohe Genauigkeit! (nur wenn glatt)

Bsp: 2-Punkt GLQ ($n=1$)

$$G_1[f] = \sum_{j=0}^1 w_j f(x_j) = w_0 f(x_0) + w_1 f(x_1)$$

$$P_{n+1} = P_2 = \frac{3}{2}x^2 - \frac{1}{2} \quad \text{hat NS} \quad x_{0,1} = \pm \frac{1}{\sqrt{3}}$$

$$w_0 = \frac{2(1 - \frac{1}{3})}{(2 \cdot (\frac{1}{\sqrt{3}}))^2} = \frac{4/3}{(2/\sqrt{3})^2} = 1 = w_1 \quad (\text{da es wird jeweils } 2)$$

$$G_1 = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right), \quad \text{nun wollen wir es auf einem beliebigen I.}$$

$$I_{[a,b]}[f] = I_{[-1,1]}[f] \approx G_1[f] = \frac{b-a}{2} \left[f\left(\frac{a+b}{2\sqrt{3}}\right) + f\left(\frac{b-a}{2\sqrt{3}}\right) + f\left(\frac{b-a}{2}\right) \right]$$

3-Punkt GLQ ($n=2$)

$$G_2 = \frac{b-a}{2} \left[\frac{5}{9} f\left(\frac{a+b}{2} - \frac{\sqrt{3}}{2} \frac{b-a}{2}\right) + \frac{8}{9} f\left(\frac{a+b}{2}\right) + \frac{5}{9} f\left(\frac{a+b}{2} + \frac{\sqrt{3}}{2} \frac{b-a}{2}\right) \right]$$

$$G_{2,1,1,1} = \frac{5}{9} f\left(-\frac{1}{\sqrt{3}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{1}{\sqrt{3}}\right)$$

Skalarprodukt in $CI[-1,1]$: $\langle f, g \rangle = \int_{-1}^1 f(x)g(x)dx$

Konvergenz

algebraischer Konvergenz: $E(N) = CN^{-\alpha}$

exponentieller Konvergenz: $E(N) = Cq^N$

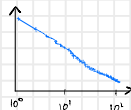
für N : Methodenparameter # Teilintervalle
 (Ordnung einer QR) gross genug, $C, \alpha \in \mathbb{R}_+, q \in (0, 1)$

Man bestimmt α und q experimentell.

algebraisch: auf einem log-log plot

$$\log(E) = \log(CN^{-\alpha}) = \log(C) - \alpha \log(N)$$

$\Rightarrow \log(E)$ hat die Steigung $-\alpha$

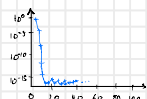


exponentiell: auf einem semi-log plot

$$E(N) = Cq^N$$

$$\log(E) = \log(C) + \log(q) \cdot N$$

$\Rightarrow \log(E)$ hat die Steigung $\log(q)$



Bsp: SSA2 (Gauss-Legendre Quadratur, zu integrierende Funktion: $f(x) = \cosh(x)$)

$I = \sinh(1) - \sinh(0);$
 $N = 1:100;$ % exakte Lösung
 $E = \text{zeros}(1, \text{length}(N));$
 for $i = 1:N$
 $[x_{in}, w_{in}] = \text{gausslog}(N, -1, 1, 'in');$ % G-L Knoten & Gewichte
 $E(i) = \text{abs}(\sum (\cosh(x_{in,i}) * w_{in,i}) - I);$
 end
 semilogy(N, E, 'b+');
 xlabel('N');
 ylabel('E');
 $\text{max} = \sum (\log(E) > -14);$ % bestimme grösste n, sodass Rundungsfehler nicht ins Spiel kommen
 $p1 = \text{polyfit}(\log(1:\text{max}), \log(E(1:\text{max})), 1);$
 $\alpha = -p1(1);$
 $q = \exp(p1(1));$
 $\text{fprintf}('q = %f \ln', q);$
 $\text{fprintf}('f = %f \ln', \alpha);$

In Analogie für algebraische Konvergenz

Integration verbessern

Gewichtsfunktion

Integrale einer bestimmten Form, z.B. $I[f] = \int_{-1}^1 \frac{f(x)}{\sqrt{1-x^2}} dx$ mittels einer geeigneten QR.

$\rightarrow$ Wir bauen eine Gauss QR mit Gewichtsfunktion $w(x) = \frac{1}{\sqrt{1-x^2}}$, d.h.:

Orthogonalbasis bzgl. dem gewichteten Skalarprodukt: $\langle f, g \rangle = \int_{-1}^1 \frac{f(x)g(x)}{\sqrt{1-x^2}} dx$
 $\rightarrow$ benutze Gram-Schmidt Verfahren auf monobasis $y_0(x) = x^j$: $p_0(x) = \frac{1}{\sqrt{2}}$, $p_1(x) = \frac{\sqrt{3}}{2}x$, $p_2(x) = \frac{3}{2\sqrt{5}}x^2 - \frac{1}{\sqrt{5}}$
 x_j sind die Nullstellen des berechneten Orthogonalpolynoms.

$$w_j = \int_{-1}^1 w(x) L_j'(x) dx, \quad \text{wo } L_j' \text{ die Lagrange-Polynome zu den Knoten } x_j$$

Substitution

$I = \int_0^{\pi/2} y \cos(y) dy = \dots$
 $y = \arcsin(x)$ exponentielle Konvergenz.
 algebraische Konvergenz

Mehrdimensionale Quadraturen

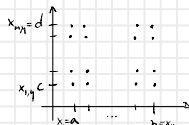
Ziel: approx. Integration von $I[f] = \int_a^b \int_c^d f(x,y) dx dy$ über ein Gebiet $[a,b] \times [c,d]$

Herleitung: 1. $\int_a^b f(x,y) dx \approx \sum_{i=1}^n \omega_{i,y} f(x_i,y)$ $\Rightarrow I[f] \approx \int_c^d \sum_{i=1}^n \omega_{i,y} f(x_i,y) dy$

$$2. I[f] \approx \sum_{i=1}^n \omega_{i,y} \int_c^d f(x_i,y) dy$$

$$3. \int_c^d f(x_i,y) dy \approx \sum_{j=1}^m \omega_{j,x} f(x_i,y_j)$$

$$4. I[f] \approx \sum_{i=1}^n \sum_{j=1}^m \omega_{i,y} \omega_{j,x} f(x_i,y_j)$$



Bsp: S4A2: Dreieck K mit Ecken $(0,0), (a,0), (0,b)$: $I[f] = \iint_K f(x,y) dx dy$

Wir transformieren das K in das Referenzquadrat $\hat{S} = [-1,1] \times [-1,1]$ durch $x = a \cdot \frac{1+\xi}{2}, y = b \cdot \frac{1+\eta}{2}$ $(\xi,\eta) \in \hat{S}$

$$J(\xi,\eta) = \frac{ab}{4} \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} = ab \cdot (1-\eta)/8$$

$$\Rightarrow I[f] = \int_K f(x,y) dK = \int_{\hat{S}} f(x(\xi,\eta), y(\xi,\eta)) \cdot \det(J) d\hat{S} = \frac{ab}{8} \int_{-1}^1 \int_{-1}^1 f(\xi,\eta) (1-\eta) d\xi d\eta$$

Die Gewichte und die Knoten der 2D-Gauss-Legendre-Quadratur mit $n=2$ auf dem Dreieck K mit $a=b=1$ ist:

$$I[f] \approx \frac{1}{8} \sum_{i,j=0}^1 \omega_{ij} f(x_i, y_j) \quad \left(\omega_{ij} = \frac{1}{4} \left(1 - \frac{1}{2} \right) \left(1 + \frac{1}{2} \right) \right)$$

Wobei $y_0 = 1/\sqrt{3}$, $y_1 = -1/\sqrt{3}$, $\omega_{00} = \omega_{01} = \omega_{10} = \omega_{11} = \frac{1}{4}$

Adaptive Quadratur

Ziel: Berechne $I[f] = \int_a^b f(x) dx$ bis auf eine Toleranz $tol < |Q[f] - I[f]|$

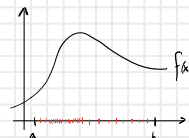
so effizient (möglichst wenig Fkt. Auswertungen) wie möglich.

Idee: Anstatt gleich verteilte Teil-Intervalle, verwenden feinere oder gröbere

Teil-Intervalle je nachdem wie stark die Funktion $f(x)$ variiert, (z.B. wo der QF gross ist).

Dazu benötigen wir eine Schätzung des Fehlers.

Um eine Schätzung zu bekommen:



① A posteriori Fehler Schätzer:

$$E[f] \leq \frac{1}{5} \|f^{(5)}\|_{\infty} (b-a)^{5/4} = K(b-a)^{5/4}$$

Nur brauchbar, wenn $\|f^{(5)}\|_{\infty}$ bekannt ist. (Was fast) immer nicht der Fall ist)

A posteriori Fehler Schätzer

① Vergleiche das Resultat einer Methode, mit dem Resultat einer genaueren Methode (mit höherer Ordnung).

② Vergleiche das Resultat einer Methode, mit dem Resultat des selben QR, aber mit halbiertem Intervall Länge.

$$E^1[f] \approx K(b-a)^{5/4}$$

$$E^2[f] \approx K \left(\frac{b-a}{2} \right)^{5/4} = 2^{-5/4} E^1[f]$$

$$E^1[f] = |Q^1[f] - Q^2[f]| + |Q^2[f] - I[f]| \leq |Q^1[f] - Q^2[f]| + |Q^2[f] - I[f]|$$

$$E^1[f] \left(1 - \frac{1}{2^{5/4}} \right) \approx |Q^1[f] - Q^2[f]|$$

$$\Rightarrow \begin{cases} E^1[f] \approx \frac{2^5}{2^5 - 1} |Q^1[f] - Q^2[f]| \\ E^2[f] \approx \frac{1}{2^5 - 1} |Q^1[f] - Q^2[f]| \end{cases}$$

$$E^2[f] \approx \frac{E^1[f]}{2^{5/4}}$$

Adaptive Quadratur MATLAB CODE

function Q = adapt_simpson_simple(f,a,b,tol,maxlevel)

% Berechne Fehler schätzer

m = QS*(a+b);

$$Q = (b-a)/6 * (f(a) + 4 * f(m) + f(b));$$

$$QL = (m-a)/6 * (f(a) + 4 * f(0.5*(a+m)) + f(m)); \quad \% \text{ linke Teil-Intervall}$$

$$QR = (b-m)/6 * (f(m) + 4 * f(0.5*(m+b)) + f(b)); \quad \% \text{ rechte Teil-Intervall}$$

$$E2 = \text{abs}(Q - (QL + QR)) / 5;$$

if (E2 < tol || maxlevel == 0)

$$Q = QL + QR;$$

else

$$QL = \text{adapt_simpson_simple}(f,a,m,tol, \text{maxlevel}-1);$$

$$QR = \text{adapt_simpson_simple}(f,m,b,tol, \text{maxlevel}-1);$$

$$Q = QL + QR$$

end

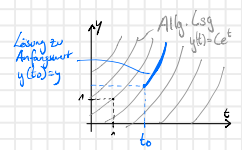
Explizite Einschrittverfahren

Ziel: Anfangswertproblem (gewöhnliche DGL + Anfangsbedingung) untersuchen.

- Lösbarkeit - Genauigkeit - explizite Einschrittverfahren.

Skalare DGL: $y'(t) = y(t) = f(t, y(t))$

Eine DGL lässt sich graphisch mit einem Richtungsfeld darstellen.



explizites Euler Verfahren ($q=1$)

1. Diskretisiere Zeitintervall $I = [t_0, T]$

$$t_k = t_0 + k \cdot h, \quad k = 0, 1, \dots, N \quad \text{mit } h = \frac{T - t_0}{N}$$

2. Setze Anfangswert $y_0 = y(t_0)$

$$\text{BT: } \begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

3. Berechne für $k=0, 1, \dots, N-1$

$$y_{k+1} = y_k + h \cdot f(t_k, y_k), \quad t_{k+1} = t_k + h = t_0 + (k+1)h$$

Anfangswertproblem (AWP) erster Ordnung

Finde $y(t), y(t_0), \dots, y(t_N)$

$$\begin{cases} \dot{y}(t) = f(t, y(t), \dots, y(t_N)) \\ y(t_0) = y_0 \end{cases} \quad \text{auf } I = [t_0, T] \quad \text{und} \quad \begin{cases} y(t_0) = y_0 \\ y(t_N) = y_0 \end{cases}$$

$$\text{Kürzer: } \vec{y}(t) = f(t, \vec{y}) \quad \text{mit } \vec{y}(t) = \begin{pmatrix} y(t) \\ y'(t) \\ \vdots \\ y^{(n)}(t) \end{pmatrix}, \quad \vec{y}_0 = \begin{pmatrix} y_0 \\ y'_0 \\ \vdots \\ y^{(n)}_0 \end{pmatrix}, \quad f(t, \vec{y}) = \begin{pmatrix} f_0(t, y(t), \dots, y^{(n)}(t)) \\ f_1(t, y(t), \dots, y^{(n)}(t)) \\ \vdots \\ f_{n-1}(t, y(t), \dots, y^{(n)}(t)) \end{pmatrix}$$

$$\text{Bsp: Lineares AWP: } \begin{cases} \dot{\vec{y}}(t) = A \vec{y}(t) \\ \vec{y}(t_0) = \vec{y}_0 \end{cases} \rightarrow \vec{y}(t) = \vec{y}_0 \cdot e^{A(t-t_0)} = \vec{y}_0 \sum_{k=0}^{\infty} \frac{A^k (t-t_0)^k}{k!}$$

Reduktion auf System 1. Ordnung

AWP n-te Ordnung kann auf ein System 1. Ordnung reduziert werden:

0) Gegeben ein gew. DGL n-ter Ordnung.

$$y^{(n)}(t) = f(t, y, y', \dots, y^{(n-1)})$$

3AW einsetzen

1) Definiere: $z_0(t) = y(t)$

$$z_1(t) = y'(t) = \dot{z}_0(t)$$

$$z_2(t) = y''(t) = \dot{z}_1(t)$$

$$\vdots$$

$$z_{n-1}(t) = y^{(n-1)}(t) = \dot{z}_{n-2}(t)$$

$$\vec{z}(t) = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{pmatrix}, \quad \vec{z}(t_0) = \begin{pmatrix} z_0(t_0) \\ z_1(t_0) \\ \vdots \\ z_{n-1}(t_0) \end{pmatrix}$$

2) Ableite $\vec{z}(t)$:

$$\text{System von } n \text{ gew. DGL 1. Ordnung } \dot{\vec{z}} = \begin{pmatrix} \dot{z}_0 \\ \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \end{pmatrix} = \begin{pmatrix} y'(t) \\ y''(t) \\ \vdots \\ y^{(n)}(t) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ f(t, z_0, z_1, \dots, z_{n-1}) \end{pmatrix}$$

$$\text{Bsp: Stromkreis } \vec{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}, \quad \begin{pmatrix} \dot{Q} \\ \dot{Q} \end{pmatrix} = \begin{pmatrix} Q \\ T \end{pmatrix}$$

autonom: Eine gew. DGL heißt autonom, falls die rechte Seite nicht explizit von t abhängt: $\dot{y} = \tilde{f}(y(t))$

Autonomisierung von DGL

o) Betrachte $\dot{y} = f(t, y(t))$, $y \in \mathbb{R}^n$

1) Führe neue Variable t hinzu:

$$\tilde{y}(t) = \begin{pmatrix} y(t) \\ t \end{pmatrix} = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \\ t \end{pmatrix} \text{ mit rechte Seite } \tilde{f}(\tilde{y}) = \begin{pmatrix} f(t, y(t)) \\ 1 \end{pmatrix} = \tilde{f}(\tilde{y}(t))$$

2) Erhält man eine autonome gew. DGL $\dot{\tilde{y}} = \tilde{f}(\tilde{y})$, $\tilde{y}(0) = \begin{pmatrix} y(0) \\ 0 \end{pmatrix}$

Picard-Lindelöf'schen Iterationsverfahren.

Wir betrachten das autonome AWP, ein Schritt der Verfahren ist:

$$\begin{cases} \dot{\tilde{y}}(t) = \tilde{A}\tilde{y}(t) \\ \tilde{y}(0) = \tilde{c} \end{cases} \quad y_{k+1}(t) = c + \int_0^t A y_k(\tau) d\tau$$

$$y_0 = c, y_1(t) = c + \int_0^t A c d\tau = c(At) \quad y_2(t) = c + \int_0^t A c(A\tau) d\tau = (1 + At + \frac{A^2}{2}t^2)c$$

$$y_k(t) = e^{At}$$

Existenz, Eindeutigkeit und Stabilität

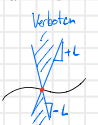
Wir betrachten: $\dot{y}(t) = f(t, y(t))$ allgemeines AWP
 $\tilde{y}(t_0) = y_0$

Lipschitz-stetigkeit

$f: I \times D \rightarrow \mathbb{R}^n$ ist Lipschitz-stetig in y mit Lipschitz-Konstante $L \geq 0$ wenn $\forall t \in I$ und $\tilde{y}, \tilde{z} \in D$ gilt:

$\|f(t, \tilde{y}) - f(t, \tilde{z})\| \leq L \|\tilde{y} - \tilde{z}\|$

Bem.: (i) Beschränkt die Steigung der Funktion um ein Punkt:



(ii) Stetig diffbar, mit beschränkter Ableitung auf $I \Rightarrow$ Lipschitz.

Bsp: $f(x) = |x|$, für $x, y \in \mathbb{R}$

$$\begin{aligned} |x| &= |x - y + y| \leq |x - y| + |y| \Rightarrow |x| - |y| \leq |x - y| \\ |y| &= |y - x + x| \leq |y - x| + |x| \Rightarrow -(x - y) \leq |x - y| \end{aligned}$$

Picard-Lindelöf: Sei $\tilde{f}$ stetig in (t, y) , Lipschitz in y auf $[t_0, t_0 + \delta] \times D$ mit $\delta > 0$ und D eine Umgebung von y_0 .

Dann existiert eine eindeutige Lösung $\tilde{y}(t)$ des AWP:

$\tilde{y}(t) = \tilde{f}(t, \tilde{y}(t))$, $\tilde{y}(t_0) = y_0$
 für mindestens eine kurze Zeit $[t_0, t_0 + \varepsilon]$ $\varepsilon > 0$

Satz II.2: Sei $\tilde{f}$ Lipschitz, mit Lipschitz- L in einer Umgebung von y_0, z_0

und seien $\tilde{y}(t), \tilde{z}(t)$ die Lösungen des jeweiligen AWP.
 $\|\tilde{y}(t) - \tilde{z}(t)\| \leq L(t - t_0) \|\tilde{y}_0 - \tilde{z}_0\|$ für $t \in [t_0, t_0 + \varepsilon]$

Runge-Kutta Verfahren

Betrachten wir das skalare AWP $y(t) = f(t, y(t))$, $y(t_0) = y_0$. Die exakte Lösung können wir formal durch Integration erhalten:

$y(t) = y_0 + \int_{t_0}^t f(\tau, y(\tau)) d\tau$ ~ Nicht viel besser wie mit der DGL. Jedoch wir können mittels QR die Integral approximieren.

Ein s-stufiges Runge-Kutta Verfahren (RK-ESV) ist def. durch:

$y_{j+1} = y_j + h \sum_{i=1}^s b_i \cdot k_i$

wobei die Stufen/Stagungen:

$k_i = f(t_j + c_i \cdot h, y_j + h \sum_{\ell=1}^s a_{i\ell} k_{\ell})$

Weiter nennt man

s ... Anzahl Stufen
 b_i ... Gewichte
 c_i ... Knoten
 $a_{i\ell}$... Runge-Kutta Matrix-Koeffs.

RK-ESVn schreibt man in einem Butcher Tablean (BT):

c_1	a_{11}	a_{12}	$\dots$	a_{1s}	$= \tilde{c} \quad A$
c_2	a_{21}	a_{22}	$\dots$	a_{2s}	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}	
	b_1	b_2	$\dots$	b_s	

Herleitung BT Methode von Heun (p=2)

$y(t_1) = y_0 + \int_{t_0}^{t_1} f(\tau, y(\tau)) d\tau = y_0 + h \int_0^1 f(t_0 + h\tau, y(t_0 + h\tau)) d\tau$

QR-TR! $\approx y_0 + A_{1,1} [f] = y_0 + \frac{h-a}{2} (f(t_0, y(t_0)) + f(t_0 + h, y(t_0 + h)))$

$= y_0 + \frac{1}{2} (f(t_0, y_0) + f(t_0 + h, y(t_0 + h)))$

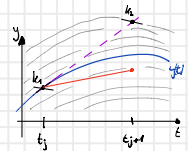
Wir kennen jetzt nun $y(t_0 + h) = y(t_1) = y_1$ nicht, aber wir benutzen den Euler Verfahren um dies zu approximieren.

$y(t_0 + h) \approx y_0 + h \cdot f(t_0, y_0) = y_1$

$\Rightarrow y_1 = y_0 + \frac{1}{2} (f(t_0, y_0) + f(t_0 + h, y_1))$

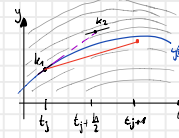
Und mit: $k_1 = f(t_j, y_j)$
 $k_2 = f(t_j + h, y_j + h k_1)$

$y_{j+1} = y_j + \frac{h}{2} (k_1 + k_2)$



BT der verbesserten Polygonzug-Methode von Euler: (p=1)

$y_{j+1} = y_j + h \cdot (b_1 \cdot k_1 + b_2 \cdot k_2) = y_j + h \cdot k_2$
 $k_1 = f(t_j, y_j)$
 $k_2 = f(t_j + \frac{1}{2}h, y_j + \frac{1}{2}h k_1)$



(verb. Euler-p=2)

Das RK-ESV (RK4) (p=4)

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

Explizite RK-Verfahren haben BT mit ober-dreieck nur Nullen

Implizite Mittelpunkts-Methode

$y_{j+1} = y_j + h k_1$
 $k_1 = f(t_j + \frac{1}{2}h, y_j + \frac{1}{2}h k_1)$ *implizite Def. von k1*

Methode der Taylorreihe

Taylorreihe der Lösung $y(t)$ um t_0 entwickelt:
 $y(t) = y(t_0) + y'(t_0)(t - t_0) + \frac{y''(t_0)}{2!}(t - t_0)^2 + \dots + \frac{y^{(p)}(t_0)}{p!}(t - t_0)^p + R_{p+1}$

Sei $(t - t_0) = h$, wenn wir den Restterm R_{p+1} vernachlässigen:

$y(t_{j+1}) \approx y_{j+1} = y_j + h y_j' + \frac{h^2}{2} y_j'' + \dots + \frac{h^p}{p!} y_j^{(p)}$

$y(t_j), y(t_j), \dots$ Kann man durch Differenzierung der Rechten Seite berechnen. Jedoch wird schnell unübersichtlich. Wir machen daher den Ansatz für $y(t)$

$y(t) = y_j + c_1(t - t_j) + c_2(t - t_j)^2 + c_3(t - t_j)^3 + \dots$
 $\dot{y}(t) = c_1 + 2c_2 h + 3c_3 h^2 + 4c_4 h^3 + \dots$

Wir setzen dann

$y(t) = c_1 + 2c_2 h + 3c_3 h^2 + \dots = f(t_j + h, y_j + c_1 h + c_2 h^2 + \dots)$
 Und finden $c_i(t_j, y_j)$ durch Koeffizientenvergleich.

$y_{j+1} = y_j + c_1 h + c_2 h^2 + c_3 h^3 + \dots$

Fehlerbetrachtung für Einschritt-Verfahren

Ziel: Genauigkeit von ESV untersuchen.

Wir betrachten ein allgemeines ESV der Form:

$$y_{i+1} = y_i + h \cdot \phi(t_i, y_i, h)$$

$$\phi(t_i, y_i, h) = \frac{y_{i+1} - y_i}{h}$$

wobei ϕ : Verfahrens-/Inkrementfunktion.

Bem: (i) Bei RK-ESV ist $\phi(t_i, y_i, h) = \sum_{j=1}^s b_j k_j$

(ii) Bei einem expliziten ESV kann man ϕ durch einsetzen berechnen.

Bei impliziten ESV muss man um ϕ zu berechnen nicht-lineares Gleichungssystem lösen

Globale Diskretisierungsfehler (GDF)

$$E_j = y(t_j) - y_j \quad \sim \text{Fehler addieren sich}$$

exakte - approx.-r. Lösung zur Zeit t_j : $h = \frac{T-t_0}{N}$

Konvergenzordnung p :

$$E = \max_{j=0, \dots, N} |y(t_j) - y_j| = O(h^p) \quad \text{für } h \text{ klein genug.}$$

Lokale Diskretisierungsfehler (LDF)

$$e_j = y(t_j) - [y(t_{j-1}) + h \cdot \phi(t_{j-1}, y(t_{j-1}), h)]$$

exakt abweichung nach 1 Schritt

Konsistenzfehler (KF)

$$\tau_j = \frac{e_j}{h} = \frac{y(t_j) - y(t_{j-1})}{h} - \phi(t_{j-1}, y(t_{j-1}), h)$$

Konsistenzordnung p

$$\tau = \max_{j=0, \dots, N} |\tau_j| = O(h^p)$$

totale - partielle Ableitungen

$$\frac{d}{dt} \geq \frac{\partial}{\partial t} \quad df = \sum_{i=1}^n \frac{\partial f}{\partial x_i} dx_i$$

Ein ESV heißt konsistent falls $p \geq 1$.

Damit Verfahren konsistent, muss gelten: $\lim_{h \rightarrow 0} \frac{y(t) - y_0}{h} \rightarrow 0 \iff \sum_{i=1}^s b_i = 1$

Die genaue Wert von p können wir folgendermassen berechnen:

Wir betrachten das starke AWP 1. Ordnung: $y'(t) = f(t, y(t)) \leftarrow$ rechten Seite (genau oft stetig-differ)

Um die Konsistenz bzw Konvergenzordnung p (siehe II.3) eines Verfahrens mit Verfahrens-Funktion ϕ zu bestimmen, müssen wir den KF τ_j mittels

Taylor-Entwicklung abschätzen. Hierzu entwickelt man die Lösung und die Verfahrens-Funktion in Potenzreihen der Schrittweite h .

$$\tau_{j+1} = \frac{y(t_{j+1}) - y(t_j)}{h} - \phi(t_j, y(t_j), h) \xrightarrow{h \rightarrow 0} y'(t_j) - \phi(t_j, y(t_j)) \stackrel{!}{=} 0$$

$$= \frac{y(t_j + h) - y(t_j)}{h} - \phi(t_j, y(t_j), h) = \frac{y(t_j) + h y'(t_j) + \frac{h^2}{2} y''(t_j) + \dots - y(t_j)}{h} - (\phi(t_j, y(t_j), 0) + h \phi'(t_j, y(t_j), 0) + \frac{h^2}{2} \phi''(t_j, y(t_j), 0) + \dots)$$

$$= y(t_j) - \phi(t_j, y(t_j), 0) + \frac{h}{2} [y''(t_j) - 2\phi'(t_j, y(t_j), 0)] + \frac{h^2}{6} [y'''(t_j) - 3\phi''(t_j, y(t_j), 0)] + \dots + O(h^p)$$

Die Frage ist dann welche h^p dominiert?

- ↳ Kleinste (und man je gegen 0 geht) Potenz mit Koeff $\neq 0$, d.h. zu überprüfen ist
 - $y(t_j) = \phi(t_j, y(t_j), 0) \quad \textcircled{1}$
 - $y'(t_j) = 2\phi'(t_j, y(t_j), 0) \quad \textcircled{2}$
 - $y''(t_j) = 3\phi''(t_j, y(t_j), 0) \quad \textcircled{3}$

Und wo sich die Taylorentwicklung auf hört ist die Konsistenzordnung p .

Bsp & Vorhansurchein: Verbesserte Euler

i) Verfahren explizit aufschreiben und Verfahrensfunktion bestimmen

$$k_1 = f(t_j, y(t_j)), k_2 = f(t_j + \frac{h}{2}, y(t_j) + \frac{h}{2} k_1), y_{j+1} = y(t_j) + h k_2$$

$$\phi = k_2 = f(t_j + \frac{h}{2}, y(t_j) + \frac{h}{2} k_1)$$

ii) ϕ Taylor entwickeln, (falls ϕ explizit von $k_1 = f(t_j, y(t_j))$ abhängt dann muss dieser Teil nicht entwickelt werden ($\partial \phi / \partial y = 0$)

Allgemein gilt:

$$f(t + \Delta t, y + \Delta y) = f(t, y)$$

$$+ \frac{\partial f}{\partial t}(t, y) \Delta t + \frac{\partial f}{\partial y}(t, y) \Delta y$$

$$+ \frac{1}{2} \frac{\partial^2 f}{\partial t^2}(t, y) \Delta t^2 + \frac{\partial^2 f}{\partial t \partial y}(t, y) \Delta t \Delta y + \frac{1}{2} \frac{\partial^2 f}{\partial y^2}(t, y) \Delta y^2$$

$$\hookrightarrow \phi(t_j, y(t_j), h) = k_2 = f(t_j + \frac{h}{2}, y(t_j) + \frac{h}{2} k_1)$$

$$= f(t_j, y(t_j))$$

$$+ \frac{\partial f}{\partial t}(t_j, y(t_j)) \cdot \frac{h}{2} + \frac{\partial f}{\partial y}(t_j, y(t_j)) \cdot \frac{h}{2} k_1$$

$$+ \frac{1}{2} \frac{\partial^2 f}{\partial t^2}(t_j, y(t_j)) \left(\frac{h}{2}\right)^2 + \frac{\partial^2 f}{\partial t \partial y}(t_j, y(t_j)) \left(\frac{h}{2}\right) k_1 + \frac{1}{2} \frac{\partial^2 f}{\partial y^2}(t_j, y(t_j)) \left(\frac{h}{2} k_1\right)^2 + \dots$$

iii) Ableitungen von ϕ mit Koeff. Vergleich bestimmen

$$\phi(t_j, y(t_j), h) = \phi(t_j, y(t_j), 0) + \phi'(t_j, y(t_j), 0)h + \frac{\phi''(t_j, y(t_j), 0)}{2}h^2$$

iv) Ableitungen der Lösung berechnen (sind immer gleich)

$$y(t) = f(t, y(t))$$

$$\dot{y}(t) = \frac{d}{dt} (y(t)) = \frac{\partial f}{\partial t}(t, y(t)) + \frac{\partial f}{\partial y}(t, y(t)) \cdot f(t, y(t))$$

$$\ddot{y}(t) = \frac{d}{dt} (\dot{y}(t)) = \frac{\partial^2 f}{\partial t^2}(t, y(t)) + 2 \frac{\partial^2 f}{\partial t \partial y}(t, y(t)) \cdot f(t, y(t)) + \frac{\partial^2 f}{\partial y^2}(t, y(t)) f(t, y(t))^2 + \frac{\partial f}{\partial t}(t, y(t)) \frac{\partial f}{\partial y}(t, y(t)) + \left(\frac{\partial f}{\partial y}(t, y(t))\right)^2 \cdot f(t, y(t))$$

v) Vergleichen

- ① $\phi(t_j, y(t_j), 0) = f(t_j, y(t_j)) \stackrel{?}{=} f(t_j)$ ✓
- ② $2 \cdot \phi'(t_j, y(t_j), 0) \stackrel{?}{=} \frac{\partial f}{\partial t}(t_j, y(t_j)) + \frac{\partial f}{\partial y}(t_j, y(t_j)) f(t_j, y(t_j)) \stackrel{?}{=} \dot{y}(t_j)$ ✓
- ③ $3 \cdot \phi''(t_j, y(t_j), 0) \stackrel{?}{=} \left(\frac{\partial^2 f}{\partial t^2} + 2 \frac{\partial^2 f}{\partial t \partial y} f + \frac{\partial^2 f}{\partial y^2} f^2\right) \stackrel{?}{=} \ddot{y}(t)$

$\Rightarrow$ Konsistenzordnung $p=2$ bzw. $\tau = O(h^2)$

(Achten auf die Faktoren $h, h^2, \dots$ wenn man ϕ mit Taylor-entwicklung vergleicht)

Man kann das Konsistenzordnung p experimentell / empirisch berechnen mit einer AWP

$$f = \phi(t, y, y)$$

$$y_0 = 2; \quad \frac{dy}{dt} = f; \quad t_0 = 0$$

$$\{y(t) = y(t), y(0) = 2\}$$

% Spectral allocation

```

exact_fun = @(t) (y0 * exp(t));
h = zeros(7, 1);
err = zeros(7, 1);
for ii = 0:6
    h(ii+1) = 2^(-ii-3);
    N = 2^(ii+3);
    [t1, y1] = quadtruncRK(f, t0, T, y0, N);
    err(ii+1) = abs(y1(end) - exact_fun(T));
end
logh(h, err, '-x');
xlabel('h');
ylabel('log(N - y(T))');
fit = polyfit(logh, log(err), 1);
fprintf('Empirische Konvergenzrate für die RK basierend auf quadrat %12f \n', fit(1));
    
```

Für ein Allgemeines 3-stufiges explizites RK:

Erhält man folgende Bedingungen:	C_1	C_2	C_3
$b_1 + b_2 + b_3 = 1$	1. Ordnung	a_{21}	$a_{31} \quad a_{32}$
$b_1 c_1 + b_2 c_2 + b_3 c_3 = \frac{1}{2}$	2. Ordnung	b_1	$b_2 \quad b_3$
$a_{21} b_2 + (a_{31} + a_{32}) b_2 = \frac{1}{2}$			
$b_2 c_1^2 + b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}$			
$a_{21} b_2 c_1 + (a_{31} + a_{32}) b_2 c_1 = \frac{1}{3}$			
$a_{21} b_2 c_1 + a_{31} b_3 c_1 + a_{32} b_3 c_2 = \frac{1}{6}$			
$a_{21} a_{31} b_3 = \frac{1}{6}$			

Falls ein Verfahren autonomisierungs-invariant ist, i.e.:

$$I: \sum_{i=1}^s b_i = 1 \quad (\text{konsistent}) \quad II: c_i = \sum_{j=1}^s a_{ij} \quad \forall i=1, \dots, s$$

Dann kann man alle Ableitungen nach t vernachlässigen und vereinfacht

Diese Eigenschaft stellt sicher, dass man identische Ergebnisse für das autonomisierte $y(t) = f(y(t))$ und das ursprüngliche nicht autonome

AWP $y(t) = f(t, y(t))$ erhält.

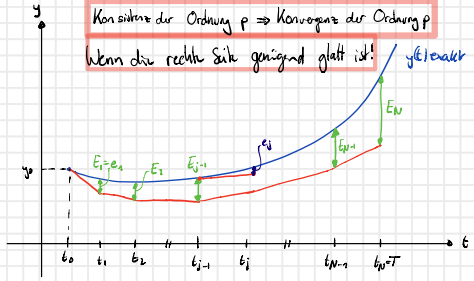
Satz II.3: Sind $\vec{f}(t, \vec{y})$ und $\vec{\phi}(t, \vec{y}, h)$ Lipschitz-stetig in $\vec{y}$, dann gilt:

$$E = \max_{j=0, \dots, N} \|\vec{y}(t_j) - \vec{y}_j\| \leq \underbrace{\|\vec{y}(t_0) - \vec{y}_0\|}_{\text{AW-Fehler}} + \underbrace{\sum_{j=1}^N \|\vec{e}_j\|}_{\text{Fehler in jeder Schritt summiert sich schrittweise auf}} \cdot e^{\int_{t_0}^T L(t, \vec{y}) dt}$$

Diese Aussage impliziert:

Konsistenz der Ordnung $p \Rightarrow$ Konvergenz der Ordnung p

Wenn die rechte Seite genügend glatt ist!



Beweis der Formel: Betrachte den GDF zur Endzeit $t_N=T$.

$$\begin{aligned} E_N &= |y(t_N) - y_N| \\ &= |y(t_N) + h \frac{y(t_N) - y(t_{N-1})}{h} - (y_N + h \phi(t_{N-1}, y(t_{N-1}), h))| \\ &= |y(t_N) - y_N + h(\phi(t_{N-1}, y(t_{N-1}), h) - \phi(t_{N-1}, y_N, h))| \\ &= |y_N + h \phi(t_{N-1}, y(t_{N-1}), h) - y_N - h \phi(t_{N-1}, y_N, h)| \end{aligned}$$

Δ -UG

$$\begin{aligned} &\leq |y(t_{N-1}) - y_{N-1}| + h \left| \frac{y(t_N) - y(t_{N-1})}{h} - \phi(t_{N-1}, y(t_{N-1}), h) \right| + h |\phi(t_{N-1}, y(t_{N-1}), h) - \phi(t_{N-1}, y_N, h)| \\ &\leq |E_{N-1}| (1 + hL) + |e_N| \end{aligned}$$

$$\begin{aligned} \Rightarrow |E_N| &\leq (1 + hL) |E_{N-1}| + |e_N| \\ &\leq (1 + hL) [(1 + hL) |E_{N-2}| + |e_{N-1}|] + |e_N| \\ &\leq \dots \\ &\leq (1 + hL)^N |E_0| + \sum_{j=1}^N (1 + hL)^{N-j} |e_j| \leq (1 + hL)^N (|E_0| + \sum_{j=1}^N |e_j|) \\ &\leq e^{hL \tilde{N}} \left(|E_0| + \sum_{j=1}^N |e_j| \right) = e^{\int_{t_0}^T L(t, \vec{y}) dt} \left(|E_0| + \sum_{j=1}^N |e_j| \right) \end{aligned}$$

III Adaptive Schrittweitensteuerung

III.1 Fehlerbetrachtung und Toleranzen

Gegeben sei ein skalares AWP $\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases}$ für $t \in [t_0, T]$

Wir wollen dieses AWP näherungsweise bis auf eine Toleranz mit möglichst wenigen Integrationschritten lösen. $E = \max_{j=0, \dots, N} |y(t_j) - y_j| < TOL$

$$E = \left(|y(t_0) - y_0| + \sum_{j=1}^N |e_j| \right) e^{\int_{t_0}^T L(t, \vec{y}) dt} < TOL \Rightarrow |e_1| + \dots + |e_1| + \dots + |e_N| < TOL$$

Anwende kein Fehler in AW LDF "wenn möglich" die I können wir nicht wissen, wenn wir es bereits, würde wir auch die exakte Lösung kennen.

Idee: Wähle $|e_j| < tol$, damit $\sum_{j=1}^N |e_j| = N \cdot tol < TOL$

Wir kennen aber N (# Schritte) nicht im Voraus!

Bei äquidistanter Schrittweite gilt $h = \frac{T-t_0}{N} \sim N = \frac{T-t_0}{h}$

Damit liegt nahe: $\frac{T-t_0}{h} \cdot tol \approx TOL$ Toleranz Kriterium

Also wähle h_j so, dass $|e_j| \leq tol$ (TK1)

(Aber, wir haben $e^{\int_{t_0}^T L(t, \vec{y}) dt}$ vorausgesetzt, was i.A. kein gute Idee ist. $\Rightarrow E_N$ ist i.A. schätzung (bzw. ungenau) von den lokalen DF auf die globale DF zu schließen.)

Deshalb können wir nur die lokalen Fehler mit lokalen Toleranzen kontrollieren, z.B.

$|e_j| \leq atol$ (allg. absolut toleranz) (TK2)

$|e_j| \leq |y_j| \cdot rtol$ (allg. relative toleranz) (TK3)

$|e_j| \leq atol + |y_j| \cdot rtol$ (abs. + rel.) (TK4)

Trotz der mögliche Aufsummierung aller e_j , funktioniert es relativ gut. Aber keine Garantie!

III.2 Lokale Fehlerschätzer

Wir benutzen die a priori Fehlerschätzer (s. II.4)

$$|e_j| = |y(t_j) - y(t_{j-1}) - h_j \phi(t_{j-1}, y(t_{j-1}), h_j)| = O(h^{p+1}) \leq C \cdot h^{p+1}$$

für ein ESV der Konvergenzordnung p .

In C stecken höhere Ableitungen der Lösung $\rightarrow$ a priori Fehlerschätzer nicht direkt berechnen

$\rightarrow$ Neue Idee: Vergleichen das Resultat eines Verfahrens mit dem Resultat eines genaueren Verfahrens (sog. Kontroll-Verfahren (KV))

III.2.1 Schrittweitzuhalburung

Gegeben ein Verfahren mit KO p , berechne im j -ten Schritt ein Schritt mit Schrittweite h :

$$y_{j+1} = y_j + h \cdot \phi(t_j, y_j, h)$$

und 2 Schritte mit halber Schrittweite $\frac{h}{2}$ ($\leftarrow$ Kontroll-Verfahren)

$$\tilde{y}_{j+1/2} = y_j + \frac{h}{2} \phi(t_j, y_j, \frac{h}{2})$$

$$\tilde{y}_{j+1} = \tilde{y}_{j+1/2} + \frac{h}{2} \phi(t_{j+1/2}, \tilde{y}_{j+1/2}, \frac{h}{2})$$

Der LDF ist dann:

$$\begin{aligned} |e_{j+1}| &= |y(t_{j+1}) - y_{j+1}| = C \cdot h^{p+1} + O(h^{p+2}) \\ |e_{j+1}| &= |y(t_{j+1}) - \tilde{y}_{j+1} + \tilde{y}_{j+1} - y_{j+1}| \leq |y(t_{j+1}) - \tilde{y}_{j+1}| + |\tilde{y}_{j+1} - y_{j+1}| \\ &= \frac{1}{2^p} \cdot C \cdot h^{p+1} + O(h^{p+2}) = \frac{|e_{j+1}|}{2^p} \end{aligned}$$

Nun wollen wir $|e_{j+1}|, |\tilde{e}_{j+1}|$ in Verbindung mit $|y_{j+1} - \tilde{y}_{j+1}|$ bringen:

$$|e_{j+1}| = |y(t_{j+1}) - \tilde{y}_{j+1} + \tilde{y}_{j+1} - y_{j+1}| \leq |y(t_{j+1}) - \tilde{y}_{j+1}| + |\tilde{y}_{j+1} - y_{j+1}|$$

$$\Rightarrow \begin{cases} |e_{j+1}| \approx \frac{2^p}{2^p - 1} |y_{j+1} - \tilde{y}_{j+1}| = \varepsilon_{j+1} \\ |e_{j+1}| \approx \frac{1}{2^p - 1} |y_{j+1} - \tilde{y}_{j+1}| = \tilde{\varepsilon}_{j+1} \end{cases} \leftarrow \text{benutze eig. immer dieses}$$

Dies sind a posteriori Fehlerschätzer

III.2.2 Eingetaktete RK-Verfahren

Jetzt versuchen wir ein neues Strategie, und der Fehler wird durch Vergleich mit eines Verfahrens der KO p mit einem Verfahren höherer KO ($p+1$)

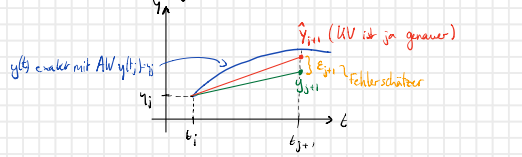
C_1	a_{21}	$\vdots$	$a_{s-1,1}$
C_2	a_{22}	$\vdots$	$a_{s-1,2}$
C_3	a_{31}	a_{32}	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_s	a_{s1}	a_{s2}	$\dots a_{s,s-1}$
b_1	b_2	$\dots b_{s-1}$	b_s
$\tilde{b}_1$	$\tilde{b}_2$	$\dots \tilde{b}_{s-1}$	$\tilde{b}_s$

z.B. Euler + Heun: $\begin{matrix} 0 & 1 \\ 1 & 0 \\ 1/2 & 1/2 \end{matrix}$

A posteriori Fehlerschätzer ist dann $|e_{j+1}| \approx |\tilde{y}_{j+1} - y_{j+1}| = \varepsilon_{j+1}$

III.3 Adaptive Schrittweitensteuerung

Ziel: Wähle Schrittweite klein genug um die gewünschte lokale Präzision zu erreichen und gross genug um effizient zu sein.



$$\varepsilon_{j+1} \approx C \cdot h^{p+1} \Rightarrow C \approx \frac{\varepsilon_{j+1}}{h^{p+1}}$$

Wir wollen: $tol_{j+1} \approx C \cdot h^{p+1} \Rightarrow H \approx h \cdot \left(\frac{tol_{j+1}}{\varepsilon_{j+1}} \right)^{\frac{1}{p+1}}$

Wenn $\varepsilon_{j+1} > tol_{j+1}$, ist $H < h$
 $\varepsilon_{j+1} < tol_{j+1}$, ist $H > h$

$$\Rightarrow H = h \cdot \min \left(\frac{tol_{j+1}}{\varepsilon_{j+1}}, \max \left(\frac{tol_{j+1}}{\varepsilon_{j+1}}, \frac{1}{2} \right) \right)$$

function [t,y] = adaptHeun(f,t0,T,y0,h0,atol,rtol)

```
p=2;
hmin=10*x eps; % KO des Heunverfahrens
fac=0.9; % eps = kleine unterschied zwischen 2 doubles
facmin=0.5; % Sicherheitsfaktor
facmax=1.5; % kleinste zugewiesene Schrittweite-Verkleinerung
t=t0; % grösste
h=h0; % Schrittweite
epsilonjp=atol+norm(y0)*rtol; % initialisieren Fehlerschätzer

while (t<T)
    tj=t(end);
    yj=y(end); % Zwischenspeicherung
    tolj=atol+norm(yj)*rtol; % Schätze die Schrittweite
    hj=hj*min(facmax,max(facmin,fac*(tolj/epsilonjp)^(1/(p+1))));
    if(hj<hmin)
        break
    end
    h2j=min(T-tj,hj); % Endzeit wird genau erreicht
    y2j=Heunschritt(f,tj,yj,h2j); % Heun mit vollen Schritt
    yi=Heunschritt(f,tj,yj,h2j*0.5); % Heun mit halbierte Schritt
    yhat2j=Heunschritt(f,tj+h2j,yi,h2j*0.5); % d
    epsilonjp=norm(yhat2j-y2j)/(2^p-1); % Fehlerschätzer für y2j
    if(epsilonjp<tolj) % Akzeptiere Zeitschritt, damit die LDF kleiner
        t=[t,tj+h2j]; % als das TK (tolj)
        y=[y,yhat2j];
    end
    % Wenn Zeitschritt h2j nicht akzeptiert, wiederhole
    % while Schleife bis h2j klein genug ist.
end
end
function y=Heunschritt(f,t,y,h)
    k1=f(t,y,yj);
    k2=f(tj+h,yj+h*k1);
    y=yj+h/2*(k1+k2);
end
```

Korn Test ESU

```
t0=0;
y0=5;
f=@(t,y)-2*y;
N=2.^linspace(0,10); % Endzeit
T=5;
error=zeros(1,length(N));
y_exact=@(t) y0*exp(-2*t);
for k=1:length(N)
    [t,y_approx]=TwoStepAdamsBashforth(f,t0,T,y0,N(k));
    error(k)=abs(y_N-approx(end)-y_N_exact(T));
end
loglog(N,error);
p=polyfit(log(N),log(error),1);
conv_order=-p(1);
display(conv_order);
```

IV Das Newton Verfahren

Ziel: Nicht lineare Gleichungen lösen.

IV.1 Skalare nicht lineare Gleichungen

Problem: Löse f(x)=0 für f:D=[a,b] -> R

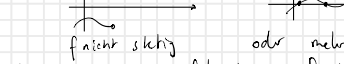
↳ benutze iterative Verfahren!

$x^{(k)} \rightarrow x^{(k+1)} \rightarrow \dots \rightarrow x^* = \lim_{k \rightarrow \infty} x^{(k)}$

f ist stetig

f(x*)=0 ist die Lösung.

Bem: Es gibt i.A. weder Existenz noch Eindeutigkeit.



Wir nehmen aber in folgenden an: f ist stetig!

IV.1.1 Fixpunktiteration und Grundlagen

Problem f(x)=0 umschreiben in Fixpunktproblem (FPP) $x = \phi(x)$

ϕ heißt Fixpunktfunktion und ein x^* welches das FPP löst, d.h. $x^* = \phi(x^*)$

heißt Fixpunkt (FP).

Ein FPP heißt konsistent mit Nullstellenproblem falls:

$f(x^*)=0 \Leftrightarrow x^* = \phi(x^*)$

Ein Verfahren der Form:

$x^{(k+1)} = \phi(x^{(k)})$ ($k=0,1,2,\dots$)

heißt Fixpunktiteration (FPI)

Bsp: (1) Löse $f(x) = x^2 - 1 = 0$,

Wir wandeln Problem in FPP um:

- (i) $x = e^{-x} = \phi_1(x)$
- (ii) $x = \phi_2(x)$ mit $\phi_2(x) = \frac{x^2 e^{x+1}}{e^{x+1} + x}$
- (iii) $x = \phi_3(x)$ mit $\phi_3(x) = x + 1 - x \cdot e$

(2) FPP konsistent mit NS Problem?

$(i) x = \phi_1 = x^2 e^{x+1} \Leftrightarrow x^2 (1+x) = x^2 e^{x+1} + x$
 $\Leftrightarrow x^2 e^{x+1} = x^2 e^{x+1} + x$
 $\Leftrightarrow x^2 e^{x+1} = x^2 e^{x+1} + x$

Bem: - FPI nicht eindeutig (wir haben mehr aufgeschrieben)

- FPI können divergieren.

- Falls sie konvergieren, können sie es mit verschiedenen Geschwindigkeiten

Um die Geschwindigkeit zu charakterisieren:

Eine konvergente Folge $x^{(k)}$ mit Grenzwert x^* hat Konvergenzordnung p falls es eine Konstante C (Konvergenzrate) gibt mit:

$|x^{(k+1)} - x^*| \leq C \cdot |x^{(k)} - x^*|^p$

Für $p=1$ muss $0 < C < 1$

Fehler in der k-ten Iteration: $\epsilon^{(k)} = |x^{(k)} - x^*|$

$\epsilon^{(k)} \leq C \cdot \epsilon^{(k-1)}$

$\epsilon^{(k+1)} \leq C \cdot (\epsilon^{(k)})^p$

Wenn man nach p und C auflöst, bekommt man:

$p = \frac{\log(\epsilon^{(k+1)}) - \log(\epsilon^{(k)})}{\log(\epsilon^{(k+1)}) - \log(\epsilon^{(k-1)})}$

$C = \frac{\epsilon^{(k+1)}}{(\epsilon^{(k)})^p} = \frac{\epsilon^{(k)}}{(\epsilon^{(k-1)})^p}$

Aber wie berechnet man $\epsilon^{(k)} = |x^{(k)} - x^*|$?

Mögliche Abbruchkriterien (ABK)

(i) $|x^k - x^{(k-1)}| \leq \text{atol}$ ABK 1

(ii) $|x^k - x^{(k-1)}| \leq \text{rtol} \cdot |x^k|$ ABK 2

(iii) $|x^k - x^{(k-1)}| \leq \text{atol} + \text{rtol} \cdot |x^k|$ ABK 3

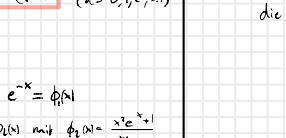
(iv) $|f(x^k)| \leq \text{ftol}$ ABK 4

IV.1.2 Bisektionsverfahren

Annahme: Wir kennen a & b mit $f(a) < 0, f(b) > 0$.

↳ D.h. Nullstelle ist eingeschlossen! (f ist stetig)

Idee: Halbiere der Intervall [a,b] und behalte Intervallhälfte welche die Annahme wieder erfüllt.



Bem: - Verfahren ist sehr einfach

- A priori fehlerschätzer: $\epsilon^{(k)} = \frac{b-a}{2^{k+1}}$

- Langsame Konvergenz: $C = \frac{1}{2}, p=1$ (linear)

- Nicht direkt anwendbar auf Systeme

IV.1.3 Newton Verfahren

Idee: Linearisiere f

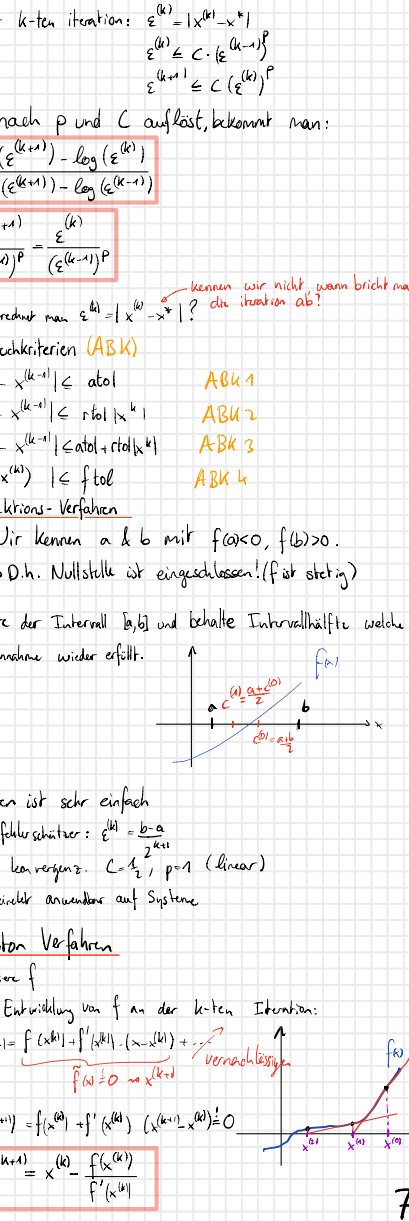
Taylor Entwicklung von f an der k-ten Iteration:

$f(x) = f(x^{(k)}) + f'(x^{(k)}) \cdot (x - x^{(k)}) + \dots$

vernachlässigen

$\tilde{f}(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)}) \cdot (x^{(k+1)} - x^{(k)}) \stackrel{!}{=} 0$

$\rightarrow x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$



Für Systeme:

Taylor-Entwicklung von $\vec{f}$ um $\vec{x}^{(k)}$ an der k-ten Iteration:

$$\vec{f}(\vec{x}) = \vec{f}(\vec{x}^{(k)}) + \underbrace{D_{\vec{f}}(\vec{x}^{(k)})}_{\text{Jacobi-Matrix}} (\vec{x} - \vec{x}^{(k)}) + \dots + \frac{1}{n!} D_{\vec{f}}^n(\vec{x}^{(k)}) (\vec{x} - \vec{x}^{(k)})^n + \dots$$

vernachlässigen

$$\vec{f}(\vec{x}^{(k+1)}) = \vec{f}(\vec{x}^{(k)}) + D_{\vec{f}}(\vec{x}^{(k)}) (\vec{x}^{(k+1)} - \vec{x}^{(k)})$$

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + D_{\vec{f}}(\vec{x}^{(k)})^{-1} (\vec{f}(\vec{x}^{(k)}))$$

$$(A^{-1} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \frac{1}{\det(A)})$$

Inverse der Jacobi Matrix sollten wir nie explizit ausrechnen!

Stattdessen $D_{\vec{f}}(\vec{x}^{(k)}) \cdot \Delta \vec{x} = -\vec{f}(\vec{x}^{(k)})$ ausrechnen ($\Delta \vec{x} = \vec{x}^{(k+1)} - \vec{x}^{(k)}$)

Bem: (i) Benötigt die Jacobi Matrix

(ii) Quadratische Konvergenz wenn nahe an Nullstelle

(iii) Kann auch schief gehen: wenn Jacobi Matrix Singular oder Abbildung ist 0!

function $x = \text{fixpunkt}(\phi, x_0, \text{maxitr}, \text{atol}, \text{rtol})$

```
x = x0;
xold = x0;
xnew = phi(xold)
err = abs(xnew - xold);
itr = 0;
while (itr < maxitr) && err > atol + rtol * abs(xnew);
    xnew = phi(xold);
    err = abs(xnew - xold);
    x = xnew;
    xold = x;
    itr = itr + 1;
end
```

fixpunkt Konvergenz: definieren phi, x0, maxitr, atol, rtol

```
x = fixpunkt(phi, x0, maxitr, atol, rtol);
err = abs(x(end) - x(1:end-1));
```

```
for i = 2:length(x)-2
    p = (log(err(i+1)) - log(err(i))) / (log(err(i)) - log(err(i-1)));
    c = err(i) / (err(i-1))^p;
    fprintf('Iteration %d in: %i\n', i-1, p);
end
```

function $x = \text{newton}(x_0, F, DF, nMax, atol, rtol)$

```
x = x0;
for ii = 1:nMax;
    delta_x = -DF(xnew)\F(xnew);
    if norm(DF(x)\F(x)) < atol + rtol * norm(x)
        break
    end
    xnew = x + delta_x;
end
```

% z.B. DF in orden 2 sein, so definieren (gleich f(x) / f'(x))
 % DF = @(x)[2*x(1), 1; x(2); exp(x(1)), exp(x(1))];

IV.2 Nichtlineare Gleichungssysteme

Problem: Gegeben, $\mathbb{R}^n$ -Vektorwertige Funktion $f: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$,
 bei $\vec{f}(\vec{x}) = 0$, Kurzschnittpunkte für: $f_1(x_1, \dots, x_n) = 0$
 $f_2(x_1, \dots, x_n) = 0$
 $\vdots$
 $f_n(x_1, \dots, x_n) = 0$ $\vec{f}(\vec{x}) = 0$

Bsp: $n=2$ $D = [0, 2]^2 \subset \mathbb{R}^2$
 $f_1(x_1, x_2) = x_1^2 + x_2^2 - 2 = 0$
 $f_2(x_1, x_2) = x_1 x_2 - 2 = 0$
 FPI konvergiert sich auf $\vec{x}^{(k+1)} = \vec{f}(\vec{x}^{(k)})$

V Stabilitätsanalyse und implizite Verfahren

Ziel: - Stabilität, Stabilitätsgebiete, A-Stabilität

- Implizite RK-ESV

- Steife Probleme (effizienter gelöst mit Implizit Verfahren)

V.1 Stabilitätsgebiete und A-Stabilität

Betrachten wir den Test AWP: $\begin{cases} y(t) = \lambda y(t) \\ y(0) = y_0 \end{cases} \lambda \in \mathbb{C}$
 (oder Dahlquist Test-Gleichung)

Benötigt man um Stabilität zu bestimmen. Die Lösung: $y(t) = y_0 e^{\lambda t}$

Wenden wir das explizite Euler Verfahren:

$$y_{i+1} = y_i + h \cdot f(t_i, y_i) = y_i + h \lambda y_i = (1 + h \lambda) y_i$$

$$y_i = (1 + h \lambda)^i y_0$$

Wir wollen den Lösungsvorlauf qualitativ zwischen der exakten und genäherten Lösung vergleichen.

(i) $\lambda > 0$: $y(t)$ nimmt zu

y_i nimmt zu (mit $(1 + h \lambda)^i \geq 1$)

(ii) $\lambda < 0$: $y(t)$ nimmt ab

y_i ? hängt von der Schrittweite h ob $(1 + h \lambda)^i < 1$ (zu)

↳ Für grobe Schritte ist das explizite Euler Verfahren instabil!

Dies erklärt das "explodieren" des Euler-Verfahrens für $\lambda < 0$ (SICAM).

II) Wenden wir das implizite Verfahren

$$y_{i+1} = y_i + h \cdot f(t_{i+1}, y_{i+1}) = y_i + h \lambda y_{i+1}$$

$$\Rightarrow y_{i+1} = \frac{1}{1 - h \lambda} y_i = \left(\frac{1}{1 - h \lambda} \right)^i y_0$$

(i) $\lambda > 0$: $y(t)$ nimmt zu

y_i nimmt zu (mit $\left(\frac{1}{1 - h \lambda} \right)^i > 1$)

(ii) $\lambda < 0$: $y(t)$ nimmt ab

y_i nimmt auch ab (unabhängig von h) $\left(\frac{1}{1 - h \lambda} \right) < 1 \quad \forall \lambda < 0$

Def: ESV angewendet auf Dahlquist Test-AWP kann man in folgender Form schreiben: $y_{i+1} = g(\lambda) y_i$

wobei $z = h \lambda$ und $g(z)$ heißt Stabilitätsfunktion (SF)

- z.B. - expliziter Euler $g(z) = 1 + z$
- impliziter Euler $g(z) = \frac{1}{1 - z}$ ($\propto 1 + z$)
- verbesserte Euler-Verfahren $g(z) = 1 + z + \frac{1}{2} z^2$
- Heun-Verfahren $g(z) = 1 + z + \frac{1}{2} z^2$
- klassisches RK $g(z) = 1 + z + \frac{1}{2} z^2 + \frac{1}{6} z^3 + \frac{1}{24} z^4$

↳ Sicht genau aus wie die Taylorreihe Entwicklung der Lösung ($y_0 e^{\lambda t}$) bis zu KO des Verfahrens!

Für $\lambda \in \mathbb{C}$ $|y_{i+1}| < |y_i|$ (Absolute Stabilität)

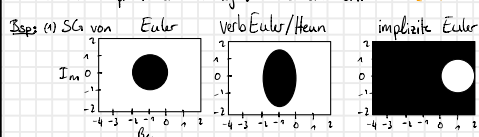
$$\Rightarrow |y_{i+1}| = |g(z) y_i| = |g(z)| |y_i| < |y_i| \Rightarrow |g(z)| < 1$$

Def: Gegeben ein ESV und zugehörige SF $g(z)$. Das Gebiet

$$S_g = \{ z \in \mathbb{C} \mid |g(z)| < 1 \}$$

heißt Stabilitätsgebiet (S_g) des Verfahrens.

Für $\lambda \in \mathbb{R}$ spricht man analog vom Stabilitätsintervall (SI)



(2) Sei $\lambda = -200$ und wir verwenden das explizite Eulerverfahren. Wie müssen h wählen um absolut stabil zu sein?

$$|g(z)| = |1 + z| < 1 \Leftrightarrow -2 < 1 + h \lambda < 1 \Leftrightarrow -2 < h \lambda < 0 \Leftrightarrow -2 < -200 h < 0$$

$$\Rightarrow \frac{2}{200} > h > 0 \Rightarrow 0 < h < \frac{1}{100}$$

Sei $\lambda \in \mathbb{C}$, so muss $|g(z)|^2 < 1$ gelten.

(3) Für lineare Systeme, nennt man das System robust falls

$$\text{Re}(\lambda_i) < 0 \text{ und } S = \frac{\max_{i=1, \dots, n} |\text{Re}(\lambda_i)|}{\min_{i=1, \dots, n} |\text{Re}(\lambda_i)|} \gg 1$$

Dann, wenn man ein Verfahren auf das System anwendet, bekommt man durch das System ein Stabilitätsintervall $z \in (a, b)$

$\Rightarrow h \lambda \in (a, b)$ dann, ein Schranke für die Schrittweite h ist

$$h \in \left(\frac{b}{\max |\text{Re}(\lambda_i)|}, \frac{a}{\max |\text{Re}(\lambda_i)|} \right) \quad (\Delta \text{Re}(\lambda_i) < 0)$$

weil das System höchstens mit $\max |\text{Re}(\lambda_i)|$ abfällt

Def: Für das Implizit Euler-Verfahren gibt es keine Einschränkung des Zeitschritts aus Stabilitätsgründen. (es kann nur wegen λ instabil werden).
Da $|g(s)| = \left| \frac{1}{1-s} \right| < 1$ stets gilt für $\operatorname{Re}(\lambda) < 0$

Def: Ein Verfahren heißt **A-stabil** falls die gesamte linke s -Halbebene im SG enthalten ist: $\{z \in \mathbb{C} \mid \operatorname{Re}(z) < 0\} \subset \operatorname{SG}$

Also: das {explizit, implizit} Euler-Verfahren ist {nicht, A-stabil}

Bsp: (1) SG der Impliziten Mittelpunktmethode $\frac{1}{1-s}$
 $k_1 = f(t_j + \frac{1}{2}, y_j + \frac{1}{2} k_1) = \lambda(y_j + \frac{1}{2} k_1) \Leftrightarrow k_1(1 - \frac{1}{2}\lambda) = \lambda y_j$
 $\Leftrightarrow k_1 = \frac{2}{1-\frac{1}{2}\lambda} \lambda y_j$
 $y_{j+1} = y_j + h k_1 = y_j + \frac{h\lambda}{1-\frac{1}{2}\lambda} y_j = \left(1 + \frac{h\lambda}{1-\frac{1}{2}\lambda}\right) y_j = \frac{1+\frac{h\lambda}{2}}{1-\frac{1}{2}\lambda} y_j$
 $\Rightarrow g(s) = \frac{1+\frac{s}{2}}{1-\frac{s}{2}} \quad \left| \frac{1+\frac{s}{2}}{1-\frac{s}{2}} \right| < 1 \quad \Rightarrow \text{A-stabil}$

Def: Ein Verfahren welches **A-stabil** ist und $\lim_{h \rightarrow 0} g(s) = 0$ erfüllt ist **L-stabil**.
z.B. implizit Euler-Verfahren ist L-stabil. Die implizite Mittelpunktmethode jedoch nicht.

Zur numerischen Behandlung von steifen Problemen sind explizite Verfahren ungeeignet

explizit	implizit
• günstig pro Schritt	• teuer pro Schritt
• Schrittweite limitiert durch schnellste abfallende Komponente	• Schrittweite nicht durch gewünschte Genauigkeit limitiert.
• Schrittweite aus Stabilitätsgründen gewählt.	• Schrittweite aus Genauigkeitsgründen gewählt.

V.2 Implizite Runge-Kutta Verfahren

→ siehe Kapitel RK-Verfahren, unter Diagonalmatrix → explizit, sonst implizit

Implizit → müssen ein nichtlineares Gleichungssystem lösen!

$$\begin{cases} k_1 = f(t_j + c_1, h, y_j + h \cdot (a_{11} k_1 + a_{12} k_2 + \dots + a_{1s} k_s)) \\ \vdots \\ k_s = f(t_j + c_s, h, y_j + h \cdot (a_{s1} k_1 + a_{s2} k_2 + \dots + a_{ss} k_s)) \end{cases}$$

Für skalare DGL sind dies s i.A. nicht-lineare Gl. für s Unbekannte (k_i)

Für ein System von n DGLen sind s.n. s.n. s.n. s.n.

→ sehr aufwendig und werden nur wenn es sich lohnt: steife Probleme.

V.3 Mehrschritt Verfahren

Mehrschrittverfahren: Berechnung von y_{j+1} benutzt sowohl y_j als auch die bekannte Approximation der Lösung zu vorherigen den Zeiten $t_{j-1}, t_{j-2}, \dots$

Lineare Mehrschrittmethoden: $\sum_{l=0}^k \alpha_l y_{j+l-1} = h \sum_{l=0}^k \beta_l f_{j+l-1}$

Adams-Bashforth:

Ausgangspunkt ist (1) $y(t_j) = y_j$ und $f(t_j, y_j)$, und die Integral mittels Interpolation zu approximieren. Also für $t_j, t_{j+1}, \dots, t_{j+m}$ sind die Approximationen bekannt (wir kennen $y_j, \dots, y_{j+m}$). Daraus bilden wir das eindeutig bestimmte Interpolationspolynom zu den m Stützpunkten $(t_j, y_j), \dots, (t_{j+m}, y_{j+m})$: $P_{m-1}(t) = \sum_{k=0}^m \beta_{j+k-1} L_{j+k-1}^m(t)$.

Wir verwenden P_{m-1} zur Approx. von (1): Lagrange Polynome

$$y_{j+1} = y_j + \int_{t_j}^{t_{j+1}} P_{m-1}(t) dt = y_j + h \sum_{l=0}^m \beta_l = y_j + h \sum_{l=0}^m \beta_l$$

Ähnlich: Adams-Moulten: $\alpha_0 = 1, \alpha_1 = -1, \alpha_l = 0$ für $l > 1$
 $\beta_0 = 0 \leftarrow \text{explizit}$

Backward Differentiation methods

Bsp: BDF2: $k=2$

1) Bestimme das Interpolationspolynom durch y_{j+1}, y_j, y_{j-1} :

$$p_2(t) = y_{j+1} \cdot \frac{1}{2h^2} (t-t_j)(t-t_{j-1}) - y_j \cdot \frac{1}{h^2} (t-t_{j-1})(t-t_{j-2}) + y_{j-1} \cdot \frac{1}{2h^2} (t-t_{j-1})(t-t_j) = y_{j+1} \cdot \frac{1}{2h^2} (t^2 - t(t_j+t_{j-1}) + t_j t_{j-1}) - y_j \cdot \frac{1}{h^2} (t^2 - t(t_j+t_{j-1}) + t_j t_{j-1}) + y_{j-1} \cdot \frac{1}{2h^2} (t^2 - t(t_j+t_{j-1}) + t_j t_{j-1})$$

(Verifikation $p_2(t_{j-1}) = y_{j-1} \cdot \frac{1}{2h^2} (t_{j-1}-t_j)(t_{j-1}-t_{j-2}) = y_{j-1} \cdot \frac{1}{2h^2} (t_{j-1}-t_j)^2 = y_{j-1}$)

Die Ableitung zur Zeit t_{j+1} ist näherungsweise:

$$\frac{dp_2(t)}{dt} \Big|_{t=t_{j+1}} = y_{j+1} \cdot \frac{1}{2h} - y_j \cdot \frac{1}{h} + y_{j-1} \cdot \frac{1}{2h} \approx y'(t) \Big|_{t=t_{j+1}} = f(t_{j+1}, y(t_{j+1}))$$

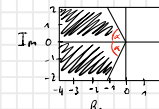
Und damit: $\frac{1}{h} (y_{j+1} - y_j + \frac{1}{2} y_{j-1}) = f(t_{j+1}, y_{j+1})$

bzw. $1 \cdot y_{j+1} - \frac{2}{3} y_j + \frac{1}{3} y_{j-1} = \frac{2}{3} h f(t_{j+1}, y_{j+1})$
 $\alpha_0 = 1, \alpha_1 = -\frac{2}{3}, \alpha_2 = \frac{1}{3}, \beta_0 = \frac{2}{3}$

Bem.: - BDF Verfahren werden oft bei steifen Problemen verwendet

- BDF können auch A(s)-stabil sein:

- Bei allen Mehrschrittverfahren muss man die ersten k-Schritte z.B. mit einem geeigneten ESV berechnen



V.4 Steife Probleme

Begegnen man bei Systemen von DGL welche Prozesse mit stark unterschiedlichen abklingzeiten modellieren.

D.h. die Prozesse laufen auf sehr unterschiedlichen Zeitskalen ab.

Für ein lineares inhomogenes System: $\vec{y}' = A \vec{y}(t) + \vec{b}(t)$ $A \in \mathbb{C}^{n \times n}$ bezeichnet man als **stiff** wenn für die Eigenwerte (EW) von A gilt

$$\operatorname{Re}(\lambda_i) < 0 \quad \text{und} \quad S = \frac{\max_{i=1, \dots, n} |\operatorname{Re}(\lambda_i)|}{\min_{i=1, \dots, n} |\operatorname{Re}(\lambda_i)|} \gg 1$$

Bei **nichtlinearen Systemen**, definiert man ein lokales Maß der Stiffheit durch Linearisieren an einem Punkt $t_i, \vec{y}_i$: $\vec{f}(t_i, \vec{y}_i) = \vec{f}(t_i, \vec{y}_i) + \frac{\partial \vec{f}}{\partial \vec{y}}(t_i, \vec{y}_i) (\vec{y} - \vec{y}_i) + \dots$
Ist obige Linearisierung stiff so nennt man das nicht-lineare System lokal stiff um den Punkt $(t_i, \vec{y}_i)$. Bei einem nichtlinearen Problem kann sowohl Stiffe als auch nicht Stiffe Abschnitte auftreten. Bei hohen Stiffheit braucht man kleine Schritte

MATLAB

Befehle:

$x = \text{linspace}(X1, X2, N)$
generates an array of N linearly equally spaced points between $X1$ and $X2$ and stores it in x (row-vector)

$x = X1:N:X2$;
is equivalent to linspace .

$P = \text{polyfit}(X, Y, N)$
finds the coefficients of a polynomial $P(X)$ of degree N that fits the data Y . P is a row vector of length $N+1$ containing the polynomial coefficients in descending powers. $P(1) \cdot x^N + P(2) \cdot x^{N-1} + \dots + P(N+1)$

$Y = \text{polyval}(P, X)$
returns the value of a polynomial P evaluated at X . P is as defined above.
 $Y = P(1) \cdot X^N + P(2) \cdot X^{N-1} + \dots + P(N+1)$

clear
deletes all variables and constants in workspace

clc
deletes contents of command window

$X = \text{zeros}(M, N)$
 $Y = \text{ones}(M, N)$
 M -by- N matrix of zeros
 M -by- N Matrix of ones

$M = [4 \ 2; 5 \ 3]$
 $M = \begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix}$
 $\sum_{i=1}^{\text{length}(X)} X(i)$
 $\sum_{i=1}^{\text{length}(M)-1} X(i)$

$\text{cat}(DIM, A, B)$
 $\text{cat}(2, A, B) \Rightarrow [A, B]$
 $\text{cat}(1, A, B) \Rightarrow [A; B]$

i
complex i

Functions: can be defined in two ways:

I/ $f1 = @(t) \ t^2$; % $f_1(t) = t^2$

II/ i. define x-axis values: $T = \text{linspace}(-1, 1, 100)$;
ii. store y-axis values in a vector: $f2 = T.^2$

function $[t, y] = \text{expEuler}(f, t0, T, y0, N)$
 $h = (T - t0) / N$;
 $t = \text{zeros}(1, N+1)$;
 $y = \text{zeros}(\text{length}(y0), N+1)$;
 $y(:, 1) = y0$;
 $t(1) = t0$;
for $k = 1:N$
 $t(k+1) = t0 + k \cdot h$;
end $y(:, k+1) = y(:, k) + h \cdot f(t(k), y(k))$;

Linear Algebra

$x = y'$ % Transpose $x = y^T$
when working with matrices M , means element-wise

Plot:

$xx = \text{linspace}(-1, 1, 200)$;
 $yy = 1 ./ (1 + 5 \cdot xx.^2 \cdot 5)$ % $f(x) = \frac{1}{1+5x^2}$
 $\text{plot}(xx, yy, 'r', 'Line Width', 2)$;
line type: dotted

$\text{title}('text')$; adds text to the current plot.

$\text{legend}('f1(x)', 'f2(x)')$;

grid on ; % shows grid lines

box on ; % encloses the graph

Multiple plots

figure
 $\text{subplot}(2, 1, 1)$;
 $\text{plot}(t, \text{real}(y - t), 'r', 'label(''Re(y(t))'')$;
 $\text{subplot}(2, 1, 2)$;
 $\text{plot}(t, \text{imag}(y - t), 'r', 'label(''Im(y(t))'')$

Loops:

$x = \text{zeros}(1, 10)$
for $i = 1: 10$ % i is incremented at each loop end from 1 to 10.
 $x(i) = i$;
end
equivalent to $x = 1:10$;

while (true)
 $dx = dx / s$;
if (abs(dx) < tol)
break
end

Klassische RK

function $[t, y] = \text{klassischRK}(f, t0, T, y0, N)$
 $h = (T - t0) / N$;
 $t = \text{zeros}(1, N+1)$;
 $y = \text{zeros}(\text{length}(y0), N+1)$;
 $y(:, 1) = y0$;
 $t(1) = t0$;
for $j = 1:N$
 $t(j+1) = t(j) + h$;
 $k_1 = f(t(j), y(j, :))$;
 $k_2 = f(t(j) + h/2, y(j, :) + h \cdot k_1 \cdot 0.5)$;
 $k_3 = f(t(j) + h/2, y(j, :) + h \cdot k_2 \cdot 0.5)$;
 $k_4 = f(t(j+1), y(j, :) + h \cdot k_3)$;
 $y(:, j+1) = y(:, j) + (h/6) \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$;
end