

Höhere Mathematik 2

Asha Schwegler

15. Juni 2022

Inhaltsverzeichnis

1 Funktionen mit mehreren Variablen	2
1.1 Partielle Ableitungen	2
1.2 Linearisierung von Funktionen	2
1.3 Das Newton-Verfahren für Systeme	2
1.3.1 Vereinfachtes Newton-Verfahren	3
1.3.2 Gedämpftes Newton-Verfahren	3
2 Ausgleichsrechnung	4
2.1 Ausgleichsrechnung	4
2.1.1 Polynominterpolation	4
2.1.2 Spline-Interpolation	5
2.1.3 Lineare Ausgleichsprobleme	7
2.1.4 Nichtlineare Ausgleichsprobleme	8
2.1.5 Das Gauss-Newton Verfahren	8
3 Numerische Integration	10
3.1 Rechteck- und Trapezregel	10
3.1.1 Summierte Rechteckregel / summierte Trapezregel	10
3.2 Simpsonregel	10
3.2.1 Summierte Simpsonregel	10
3.3 Fehler der summierten Quadraturformeln	11
3.4 Gauss-Formeln	11
3.5 Romberg-Extrapolation	11
4 Einführung in gewöhnliche Differentialgleichungen	11
4.1 Problemstellung ODE	11
4.2 Richtungsfelder DGL 1.Ordnung	12
4.3 Eulerverfahren	12
4.4 Mittelpunktverfahren	12
4.5 Das modifizierte Eulerverfahren	13
4.6 Fehlerordnung eines Verfahrens	13
4.7 Das klassische 4-Stufige Runge-Kutta Verfahren	13
4.7.1 Das allgemeine s-stufige Runge-Kutta Verfahren	13
4.8 Zurückführen DGL k-ter Ordnung auf k DGL 1.Ordnung	15
4.9 Lösen eines Systems von k DGL 1.Ordnung	16

1 Funktionen mit mehreren Variablen

1.1 Partielle Ableitungen

$m = f'(x_0)$ im Punkt $(x_0, f(x_0))$

Tangentengleichung

Beispiel: $P(1,3)$

$$t_x = \underbrace{f(x_1, x_2)}_{f(1,3)} + \underbrace{\frac{\delta f}{\delta x_1}(x_1^{(0)}, x_2^{(0)})}_{\text{nach } x_1} * (x_1 - x_1^{(0)}) + \underbrace{\frac{\delta f}{\delta x_2}(x_1^{(0)}, x_2^{(0)})}_{\text{nach } x_2} * (x_2 - x_2^{(0)})$$

1.2 Linearisierung von Funktionen

Jacobi-Matrix $Df(x)$

Jacobi-Matrix enthält sämtliche partiellen Abl.1.Ord.von f:

$$\begin{bmatrix} \frac{\delta f_1}{\delta x_1} & \frac{\delta f_1}{\delta x_2} & \cdots & \frac{\delta f_1}{\delta x_n} \\ \frac{\delta f_2}{\delta x_1} & \frac{\delta f_2}{\delta x_2} & \cdots & \frac{\delta f_2}{\delta x_n} \\ \frac{\delta f_m}{\delta x_1} & \frac{\delta f_m}{\delta x_2} & \cdots & \frac{\delta f_m}{\delta x_n} \end{bmatrix}$$

Beispiel:

$$f(x) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1^2 + x_2 - 11 \\ x_1 + x_2^2 - 7 \end{bmatrix}$$

$$x^{(0)} = (1, 1)^T$$

Partielle Ableitung:

$$Df(x_1, x_2) = \begin{bmatrix} 2x_1 & 1 \\ 1 & 2x_2 \end{bmatrix}$$

An der Stelle $x^{(0)}$:

$$Df(x_1^{(0)}, x_2^{(0)}) = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Linearisierung:

$$g(x) = f(x^{(0)}) + Df(x^{(0)}) * (x - x^{(0)})$$

$$g(x_1, x_2) = \begin{bmatrix} -9 \\ -5 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 - 12 \\ x_1 + 2x_2 - 8 \end{bmatrix}$$

Gleichung der Tangentialebene:

Alle angelegten Tangenten an die Bildfläche $y = f(x - 1, x_2)$ im Flächenpunkt $P = f(x_1^{(0)}, x_2^{(0)})$

1.3 Das Newton-Verfahren für Systeme

- Konvergiert quadratisch wenn $Df(x)$ regulär, und f 3-mal stetig differenzierbar ist.
- Vereinfachtes Newton Verfahren konvergiert linear.

Mögliche Abbruchkriterien: $\epsilon > 0$

1. $n \geq n_{max}$ (bestimmte Anzahl Iterationen)
2. $\|x^{n+1} - x^n\| \leq \|x^{n+1}\| * \epsilon$ (relativer Fehler)

3. $\|x^{n+1} - x^n\| \leq \epsilon$ (absoluter Fehler)
4. $\|f(x^{n+1})\| \leq \epsilon$ (max residual)

Algorithmus:

1. $Df(x^n)\delta^n = -f(x^n)$
2. nach δ^n auflösen
3. $x^{n+1} = x^n + \delta^n$

1.3.1 Vereinfachtes Newton-Verfahren

Konvergiert nur noch linear!!
 Natürlich deutlich langsamer!

Immer wieder $Df(x^0)$ verwenden

Algorithmus:

1. $Df(x^0)\delta^n = -f(x^n)$
2. nach δ^n auflösen
3. $x^{n+1} = x^n + \delta^n$

1.3.2 Gedämpftes Newton-Verfahren

Nach dem n-ten Schritt wenn $Df(x^n)$ schlecht konditioniert ist (nicht oder fast nicht invertierbar), dann $x^n + \delta^n$ verwerfen!!
 Funktioniert auch mit vereinfachtes Newton Verfahren.

1 Probieren:

$$x^n + \frac{\delta^n}{2}$$

Mit der Bedingung:

$$\|f(x^n) + \frac{\delta^n}{2}\|_2 < \|f(x^n)\|_2$$

Weil wir Iteration $\|f(x^n)\|_2$ gegen 0 erreichen wollen

Algorithmus:

```
1.  $Df(x^n)\delta^n = -f(x^{(n)})$ 
2. nach  $\delta^n$  auflösen
3. Finde minimale aus  $k \in \mathbb{N}$  mit  $\|f(x^n) + \frac{\delta^n}{2^k}\|_2 < \|f(x^n)\|_2$ ,  $k_{max} = 4$ 
   sofern nichts Anderes als sinnvoll angegeben

   while k <= k_max:
       new_residual = np.linalg.norm
       (f_lambda(x_n + (delta.reshape(x0.shape[0], ) /
       (2 ** k))), 2)
   #  $f(x(n) + \frac{\delta^n}{2^k})$ 
       if new_residual < last_residual:
           delta = delta / (2**k)
           x_next = x_n +
           delta.reshape(x0.shape[0], )
       #  $x(n+1) = x(n) + \frac{\delta^n}{2^k}$ 
           k_actual = k
       break
   # Minimales k, für welches das Residuum besser ist wurde gefunden
   -> abbrechen
       else:
           k=0

   k += 1

4.  $x^{n+1} = x^n + \frac{\delta^n}{2^k}$ 
```

2 Ausgleichsrechnung

2.1 Ausgleichsrechnung

- Datenpunkte mit gewissen Streuung durch einfache Funktion annähern
- Mehr Gleichungen als unbekannte (Mehr Datenpunkte als Parameter)

2.1.1 Polynominterpolation

Gesucht: $P_n(x)$ welche $n+1$ Stützpunkte interpoliert Jeder Stützpunkt gibt lin.Gleichung für die Bestimmung der Koeffizienten.

Grad n so wählen, dass lin.Gleichungssystem gleich viele Gleichungen wie unbekannte Koeffizienten hat.

$$P_n(x_0) = a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n = y_0$$

$$P_n(x_1) = a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n = y_1$$

.

.

.

$$P_n(x_n) = a_0 + a_1x_n + a_2x_n^2 + \dots + a_nx_n^n = y_n$$

$$= \left[\begin{array}{cccc} 1 & x_0 & \dots & x_0^n \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ 1 & x_n & \dots & x_n^n \end{array} \right] * \left[\begin{array}{c} a_0 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{array} \right] = \left[\begin{array}{c} y_0 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{array} \right] \left. \vphantom{\begin{array}{c} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{array}} \right\} \text{Vandermonde - Matrix = Schlechtkonditioniert. } \text{Abn} \geq 20 \text{ numerisch instabil}$$

Lagrange Interpolationsformel:

Lagrangeform von $P_n(x)$:

$$P(x) = \sum_{i=0}^n l_i(x) y_i$$

LagrangePolynome = $l_i(x)$:

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

Stückweise Interpolation:

Interpolationspolynom erster Ordnung: $n = 1$

Stützpunkte so wählen: x_{j-1} und x_{j+1} , somit 2 Werte: $n = 1$

Grösse des Fehlers an Stelle x wenn:

y_i Funktionswerte (genügend oft stetig differenzierbare Funktion)

$$|f(x) - P_n(x)| \leq \frac{|(x - x_0)(x - x_1) \dots (x - x_n)|}{(n + 1)!}$$

Max der $(n+1)$ -ten Ableitung der $f(x)$ Intervall $[x_0, x_n]$ kennen, Fehlerabschätzung nur dann möglich.

2.1.2 Spline-Interpolation

Bedingungen für S_i :

1. $S_i(x_i) = y_i, S_{i+1}(x_{i+1}) = y_{i+1}, \dots$ *Interpolation*
2. $S_i(x_{i+1}) = S_{i+1}(x_{i+1}), S_{i+1}(x_{i+2}) = S_{i+2}(x_{i+2}), \dots$ *Stetiger Übergang*
3. $S'_i(x_{i+1}) = S'_{i+1}(x_{i+1}), S'_{i+1}(x_{i+2}) = S'_{i+2}, \dots$ *Keine Knicke*
4. $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1}), S''_{i+1}(x_{i+2}) = S''_{i+2}, \dots$ *Gleiche Krümmung*
5. Mindestens den Grad 3 ... *Kubische Splines*

3 Intervalle, 4 Stützpunkte:

$[x_0, x_1], [x_1, x_2], [x_2, x_3]$

Ansatz:

$$S_0 = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3, x \in [x_0, x_1]$$

$$S_1 = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3, x \in [x_1, x_2]$$

$$S_2 = a_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3, x \in [x_2, x_3]$$

$3 * 4 = 12$ Koeffizienten $\implies 12$ Bedingungen

1. Interpolation der Stützpunkte:

$$1. S_0(x_0) = y_0$$

$$2. S_1(x_1) = y_1$$

$$3. S_2(x_2) = y_2$$

$$4. S_3(x_3) = y_3$$

2. Stetiger Übergang an Stellen x_1 und x_2 :

$$5. S_0(x_1) = S_1(x_1)$$

$$6. S_1(x_2) = S_2(x_2)$$

3. Erste Ableitung an Übergangstellen übereinstimmen:

$$7. S'_0(x_1) = S'_1(x_1)$$

$$8. S'_1(x_2) = S'_2(x_2)$$

4. Zweite Ableitung an Übergangstellen übereinstimmen:

$$9. S''_0(x_1) = S''_1(x_1)$$

$$10. S''_1(x_2) = S''_2(x_2)$$

= 10 Bedingungen

Die weiteren 2 Bedingungen können frei gewählt werden.
Diese beziehen sich häufig auf Randstellen x_0 und x_3 .

Beispiele:

Natürliche kubische Splinefunktion:

Mit möglichen Wendepunkt im Anfangs und Endpunkt.

$$\left. \begin{aligned} S'''_0(x_0) &= 0 \\ S'''_2(x_3) &= 0 \end{aligned} \right\}$$

Periodische kubische Splinefunktion:

Wenn man Periode $p = x_3 - x_0 = 0$ hat und damit y_0 bzw. $S_0(x_0) = S_2(x_3)$ gilt

$$\left. \begin{aligned} S'_0(x_0) &= S'_2(x_3) \\ S''_0(x_0) &= S''_2(x_3) \end{aligned} \right\}$$

mit not-a-knot Bedingung kubische Splinefunktion:

s.d. Auch dritte Ableitung in $x - 1, x - 2$ noch stetig ist. (x_1, x_2 keine echten Knoten)

$$\left. \begin{aligned} S'''_0(x_1) &= S'''_1(x_1) \\ S'''_1(x_2) &= S'''_2(x_2) \end{aligned} \right\}$$

Algorithmus: natürliche kubische Splinefunktion

$$S_i = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Koeffizienten a_i, b_i, c_i, d_i berechnen:

1. $a_i = y_i$

2. $h_i = x_{i+1} - x_i$

3. $c_0 = 0, c_n = 0$!!! (für periodisch ($s_1 = s_0$), für not a knot ($d_0 = d_1, d_{n-2} = d_{n-1}$))

4. Berechnung der Koeffizienten $c_1, c_2, c_3, \dots, c_{n-1}$

• $i = 1$:

$$-2(h_0 + h_1)c_1 + h_1c_2 = 3\frac{y_2 - y_1}{h_1} - 3\frac{y_1 - y_0}{h_0}$$

• falls $n \geq 4$ dann für $i = 2, \dots, n-2$

$$-h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_ic_{i+1} = 3\frac{y_{i+1} - y_i}{h_i} - 3\frac{y_i - y_{i-1}}{h_{i-1}}$$

• $i = n - 1$:

$$\begin{aligned}
& - 2(h_{n-2} + h_i) c_{n-2} + 2(h_{n-2} + h_{n-1}) c_{n-1} = 3 \frac{y_n - y_{n-1}}{h_{n-1}} - 3 \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\
5. \quad b_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3} (c_{i+1} + 2c_i) \\
6. \quad d_i &= \frac{1}{3h_i} (c_{i+1} - c_i)
\end{aligned}$$

Ergibt das Gleichungssystem $A \mathbf{c} = \mathbf{z}$

- A ist immer invertierbar
- Numerische Lsg. durch Gauss-Algo
- System ist gut konditioniert
- Pivotsuche nicht erforderlich

$i = 0, \dots, n-1$

$$A = \begin{bmatrix} \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \end{bmatrix} \quad c = \begin{bmatrix} \text{ } \\ \text{ } \\ \text{ } \end{bmatrix} = z = \begin{bmatrix} \text{ } \\ \text{ } \\ \text{ } \end{bmatrix}$$

2.1.3 Lineare Ausgleichsprobleme

Ausgleichsgerade: $f(x) = ax + b$: gesucht: $F = af_1 + bf_2 \mid a, b \in \mathbb{R}$ **Basisfunktion:** $f_1(x) = x$ und $f_2(x) = 1$

Form Fehlerfunktional:

$$E(f)(a, b) := \sum_{i=1}^n (y_i - f(x_i))^2 = \sum_{i=1}^n (y_i - (ax_i + b))^2$$

minimal = partiellen Ableitung nach a, b müssen verschwinden

$$\begin{bmatrix} \sum_{i=1}^n (x_i)^2 & \sum_{i=1}^n (x_i) \\ \sum_{i=1}^n (x_i) & n \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n (x_i y_i) \\ \sum_{i=1}^n (y_i) \end{bmatrix}$$

Nach a, b auflösen

Allgemeine Definition:

Gegeben:

n-Wertepaare: $(x_i, y_i), i = 1, \dots, n$

Basisfunktionen: $(f_1, \dots, f_m)[a, b]$

Ansatzfunktionen: $f := \lambda_1 f_1 + \dots + \lambda_m f_m$

Lineares Ausgleichsproblem mit Fehlerfunktional:

$$E(f) = \|y - f(x)\|_2^2 = \sum_{i=1}^n (y_i - \sum_{j=1}^m \lambda_j * f_j(x_i))^2 = \|y - A\lambda\|_2^2$$

Fehlergleichungssystem: $A\lambda = y$

$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \dots & f_m(x_n) \end{bmatrix} * y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \lambda \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}$$

Normalgleichungen: $\frac{\delta E(f)(\lambda_1, \dots, \lambda_m)}{\delta \lambda_j}$

Normalgleichungssystem: $A^T A \lambda = A^T y$

Lösung: λ beinhaltet gesuchte Parameter des linearen Ausgleichproblems
Verfahren am Besten mit QR-Zerlegung

2.1.4 Nichtlineare Ausgleichprobleme

Allgemeines Ausgleichproblem:

Fehlerfunktional:

f_p = Ansatzfunktion

$$E(f)(a, b) := \sum_{i=1}^n (y_i - f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_i))^2 = \left\| \begin{bmatrix} y_1 - f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_1) \\ y_2 - f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_2) \\ \vdots \\ y_n - f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_n) \end{bmatrix} \right\|_2^2 =$$

$$\|y - f(\lambda)\|_2^2$$

$$f(\lambda) := f(\lambda_1, \lambda_2, \dots, \lambda_m) := \begin{bmatrix} f_p(\lambda_1, \lambda_2, \dots, \lambda_m) \\ f_p(\lambda_1, \lambda_2, \dots, \lambda_m) \\ \vdots \\ f_p(\lambda_1, \lambda_2, \dots, \lambda_m) \end{bmatrix} := \begin{bmatrix} f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_1) \\ f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_2) \\ \vdots \\ f_p(\lambda_1, \lambda_2, \dots, \lambda_m, x_n) \end{bmatrix},$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}$$

Falls Ansatzfunktionen linear in den Parametern \Rightarrow

Spezialfall des Ausgleichsproblems: $f(\lambda) = A\lambda \Rightarrow$ Allg. Ausgleichsproblem = Minimums einer Funktion

Schritte:

1. Normalgleichungen aufstellen (part. Abl. von f)
2. $\lambda_i = 0$
3. Nichtlineare Gleichungssystem lösen

2.1.5 Das Gauss-Newton Verfahren

Quadratmittelpunktproblem: Problem einen Vektor x zu finden für den $E(x)$ minimal wird.

Nichtlineare Ausgleichsprobleme: $g(\lambda) := y - f(\lambda)$

Fehlerfunktional: $E(\lambda) := \|g(\lambda)\|_2^2 = \|y - f(\lambda)\|_2^2$

Gauss-Newton-Verfahren: Lin. Ausgl. RG + Newton-Verfahren = $g(\lambda) = g(\lambda^{(0)}) + Dg(\lambda^{(0)}) * (\lambda - \lambda^{(0)})$
= Verallg. Tangentengleichung!!

$$Dg(\lambda^0) = \begin{bmatrix} \frac{\delta g_1}{\lambda_1}(\delta\lambda^0) & \frac{\delta g_1}{\lambda_2}(\delta\lambda^0) & \dots & \frac{\delta g_1}{\lambda_m}(\delta\lambda^{(0)}) \\ \frac{\delta g_2}{\lambda_1}(\delta\lambda^0) & \frac{\delta g_2}{\lambda_2}(\delta\lambda^0) & \dots & \frac{\delta g_2}{\lambda_m}(\delta\lambda^{(0)}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta g_n}{\lambda_1}(\delta\lambda^0) & \frac{\delta g_n}{\lambda_2}(\delta\lambda^0) & \dots & \frac{\delta g_n}{\lambda_m}(\delta\lambda^{(0)}) \end{bmatrix}$$

Minimum Fehlerfunktional: $E(\lambda) = \underbrace{\|g(\lambda^{(0)})\|}_y + \underbrace{Dg(\lambda^0)^T}_{-A} \underbrace{\lambda - \lambda^0}_{\delta}$

Normgleichungssystem: $A^T A \delta = A^T y = Dg(\lambda^0)^T Dg(\lambda^0) \delta = Dg(\lambda^0)^T * g(\lambda^0) \implies$

QR-Zerlegung: $Dg(\lambda^0) = QR \geq R\delta = -Q^T g(\lambda^0)$

\implies Lösung: $\Lambda = (\lambda^0) + \delta$

Schritte:

1. Berechne Funktion

- $g(\lambda) := y - f(\lambda)$
- $Dg(\lambda)$

2. $k = 0, 1, \dots$:

δ^k als Lösung: $\min \|g(\lambda^k) + Dg(\lambda^k) * \delta^k\|_2^2$
 $\implies Dg(\lambda^k)^T * Dg(\lambda^k) \delta^k = Dg(\lambda^k)^T * g(\lambda^k)$

nach δ^k auflösen durch QR-Zerlegung.

- $Dg(\lambda^k) = Q^k R^k$
- $R^k \delta^k = -(Q^k)^T g(\lambda^k)$

3. Setze: $\lambda^{k+1} = \lambda^k + \delta^k$

Korrektur δ^k nur akzeptiert wenn Fehlerfunktional zur Abnahme führt: **Fehlerfunktional**

$$E(\lambda^{k+1}) = \|g(\lambda^{k+1})\|_2^2 < \|g(\lambda^k)\|_2^2 = E(\lambda^k)$$

Gedämpftes Gauss-Newton-Verfahren

1. Berechne Funktion:

- $g(\lambda) := y - f(\lambda)$
- $Dg(\lambda)$

2. $k = 0, 1, \dots$:

δ^k als Lösung: $\min \|g(\lambda^k) + Dg(\lambda^k) * \delta^k\|_2^2$
 $\implies Dg(\lambda^k)^T * Dg(\lambda^k) \delta^k = Dg(\lambda^k)^T * g(\lambda^k)$

nach δ^k auflösen durch QR-Zerlegung.

- $Dg(\lambda^k) = Q^k R^k$
- $R^k \delta^k = -(Q^k)^T g(\lambda^k)$

3. Finde das minimale $p \in 0,1,\dots,p_{max}$

$$\underbrace{\|\lambda^{(k)} + \frac{\delta^k}{2^p}\|_2^2}_{\lambda^{(k+1)}} < \|g(\lambda^{(k)})\|_2^2$$

4. Falls kein min. p gefunden:
 $p=0$ und weiterfahren

5. Setze: $\lambda^{(k)} + \frac{\delta^{(k)}}{2^p}$

Abbruchkriterium: $\|\frac{\delta^{(k)}}{2^p}\| < TOL$

Keine Garantie, dass die Näherung max. Abstand von TOL zum ges. Min. ist.

3 Numerische Integration

- Summierte Rechteckregel ist genauer als die summierte Trapezregel
- Summierte Simpsonregel ist am genauesten (verglichen mit sum.Recht.und Trap.)
- Faktor Fehlerabschätzung summ.Rechteckregel < summ.Trap.Regel = 2

3.1 Rechteck- und Trapezregel

Annäherung bestimmtes Integral.

Rechtecksregel / Mittelpunktsregel:

$$Rf = f\left(\frac{a+b}{2}\right) * (b-a)$$

$$Tf = \frac{f(a) + f(b)}{2} * (b-a)$$

3.1.1 Summierte Rechteckregel / summierte Trapezregel

$$h = \frac{b-a}{n}; x_i = a + i * h; x_n = b; (i=0,\dots,n-1)$$

$$Rf(h) = h * \sum_{i=0}^{n-1} f\left(x_i + \frac{h}{2}\right)$$

$$Tf(h) = h * \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right)$$

n = Anzahl Subintervalle $[x_i, x_{i+1}]$

3.2 Simpsonregel

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

3.2.1 Summierte Simpsonregel

$$Sf(h) = \frac{h}{3} \left(\frac{1}{2}f(a) + \sum_{i=1}^{n-1} f(x_i) + 2 \sum_{i=1}^n f\left(\frac{x_{i-1}+x_i}{2}\right) + \frac{1}{2}f(b)\right)$$

3.3 Fehler der summierten Quadraturformeln

$$|\int_a^b f(x)dx - Rf(h)| \leq \frac{h^2}{24}(b-a) * \max|f''(x)|$$

$$|\int_a^b f(x)dx - Tf(h)| \leq \frac{h^2}{12}(b-a) * \max|f''(x)|$$

$$|\int_a^b f(x)dx - Sf(h)| \leq \frac{12h^4}{2880}(b-a) * \max|f^{(4)}(x)|$$

Schritte berechnen bis $\overset{2880}{\text{Tol. erreicht}}$:

$$\frac{h^2}{24}(b-a) \leq Tol \mid * \frac{24}{(b-a)} \text{etc... (Analog für andere Formel)}$$

3.4 Gauss-Formeln

x_i Stützstellen müssen nicht äquidistant sein \implies so wählen, dass $\int_a^b f(x)dx$ optimal approximiert.
 a_i, x_i so wählen, dass Fehlerordnung möglichst hoch bzw. Fehler möglichst klein.

Gauss-Formeln für n=1,2,3: $\int_a^b f(x)dx \sim \frac{b-a}{2} \sum_{i=1}^n a_i f(x_i)$

$$n=1: G_1 f = (b-a) * f\left(\frac{b+a}{2}\right)$$

$$n=2: G_2 f = \frac{b-a}{2} \left[f\left(-\frac{1}{\sqrt{3}} * \frac{b-a}{2} + \frac{b+a}{2}\right) \right]$$

$$\text{n=3: } G32f = \frac{b-a}{2} [\frac{5}{9} * f(-\sqrt{0.6} * \frac{b-a}{2} + \frac{b+a}{2}) + \frac{3}{9} * f(\frac{b+a}{2})] + \frac{b-a}{2} [\frac{5}{9} * f(-\sqrt{0.6} * \frac{b-a}{2} + \frac{b+a}{2})]$$

3.5 Romberg-Extrapolation

$$T_{j0} = Tf(\underbrace{\frac{b-a}{2j}}_{(=h)}), \text{ Für } j=0,1,\dots,m-k$$

$$T_{jk} = \frac{4^k * T_{j+1,k-1} - T_{j,k-1}}{4^k - 1}, \text{ Für } k=1,2,...,m \text{ und } j=0,1,...,m-k$$

= Näherungen der Fehlerordnung $2k + 2$ gegeben.

Romberg-Folge: $h_j = \frac{b-a}{2^j}$

$$n_j = 2^j, x_i = a + ih_j$$

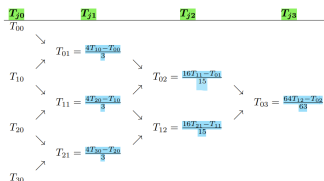


Abbildung 1: RombergExtrapolation.

4 Einführung in gewöhnliche Differentialgleichungen

4.1 Problemstellung ODE

$$y^n(x) = f(x, y(x), y'(x), \dots, y^{(n-1)}(x))$$

gesucht: $y = y(x) \implies$ Lösungen im Intervall $[a, b]$

Bemerkungen:

1. Neben den ODE gibt es noch die partiellen
2. Es gibt Systeme von Diff.Gleichungen, die u.Umständen gekoppelt sind, z.B gekoppelte Schwingungen
3. Allgemeine Lösung: DGL n-ter Ordnung enthält noch n unabhängige Parameter (Integrationskonstante unbestimmter Integrale)
4. Unterscheidung Anfangs- oder Randwertproblem, bei Anwendung numerischer Verfahren

Definition Anfangswertproblem (AWP)

1.Ordnung:

Gesucht: Spezifische Lösungskurve $y = y(x)$, durch vorgegebenen $P = (x_0, y(x_0))$

Gegeben: $y'(x) = f(x, y(x))$ und Anfangswert $y(x_0)$

2.Ordnung:

Gesucht: Spezifische Lösungskurve $y = y(x)$, durch vorgegebenen $P = (x_0, y(x_0))$ und im Punkt x_0 vorgegebene Steigung $y'(x_0) = m$

Gegeben: $y''(x) = f(x, y(x), y'(x))$ und Anfangswerte $y(x_0), y'(x_0)$

4.2 Richtungsfelder DGL 1.Ordnung

Linienelement: Pfeil.

Tangente $y'(x) = f(x, y(x))$ durch Pfeil graphisch anzeigen, diese geben in jedem Punkt, die Richtung der Lösungskurve an.

Idee: Lösung AWP dem Richtungsfeld möglichst genau zu folgen \implies Diskretisierung benötigt:

Intervall $[a, b] = [x_i, x_{i+1}]$

$$h = \frac{b - a}{n}$$

$$x_i = a + i * h$$

$$x_{i+1} = x_i + h$$

$$y_{i+1} = y_i + \text{Steigung}$$

4.3 Eulerverfahren

$$y'(x) = f(x, y(x)), y_0 = y(a), x_0 = a$$

Algorithmus:

Geg AWP:

$$y'(x) = f(x, y(x)), y_0 = y(a)$$

Verfahren:

$$x_{i+1} = x_i + h$$

$$y_{i+1} = y_i + h * f(x_i, y_i)$$

4.4 Mittelpunktverfahren

Algorithmus:

Geg AWP:

$$y'(x) = f(x, y(x)), y_0 = y(a)$$

Verfahren:

$$x_{h/2} = x_i + \frac{h}{2}$$

$$y_{h/2} = y_i + \frac{h}{2} * f(x_i, y_i)$$

$$x_{i+1} = x_i + h$$

$$y_{i+1} = y_i + h * f(x_{h/2}, y_{h/2})$$

4.5 Das modifizierte Eulerverfahren

Algorithmus:

Geg AWP:

$$y'(x) = f(x, y(x)), y_0 = y(a)$$

Verfahren:

$$x_{i+1} = x_i + h$$

$$yEuler_{i+1} = y_i + h * f(x_i, y_i)$$

$$k1 = f(x_i, y_i)$$

$$k2 = f(x_i, yEuler_{i+1})$$

$$y_{i+1} = y_i + h * \left(\frac{k1 + k2}{2} \right)$$

4.6 Fehlerordnung eines Verfahrens

Lokaler Fehler: Fehler nach einem Verfahren

$$\varphi(x_i, h) := y(x_{i+1}) - y_{i+1}$$

Konsistenzordnung p falls:

$$\varphi(x_i, h) \leq C * h^{p+1} \implies \text{genügend kleine } h \text{ und } C > 0$$

Globaler Fehler: Gesamtfehler also nach n-Iterationen

$$y(x_n) - y_n$$

Konvergenzordnung p falls:

$$|y(x_n) - y_n| \leq C * h^p \implies \text{genügend kleine } h \text{ und } C > 0$$

- Für die hier betrachteten Verfahren: Konsistenzordnung = Konvergenzordnung
- Verwendbar: Nur Verfahren mit Konvergenzordnung $p \geq 1 \implies$ Globaler Fehler gegen 0.

Eulerverfahren: $p = 1$

Mittelpunktsregel: $p = 2$

Mod.Eulerverfahren: $p = 2$

4.7 Das klassische 4-Stufige Runge-Kutta Verfahren

Algorithmus:

Geg AWP:

$$y'(x) = f(x, y(x)), y_0 = y(a)$$

Verfahren:

$$x_{i+1} = x_i + h$$

$$k1 = f(x_i, y_i)$$

$$k2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} * k1\right)$$

$$k3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} * k2\right)$$

$$k4 = f(x_i + h, y_i + h * k3)$$

$$y_{i+1} = y_i + h * \frac{1}{6} (k1 + 2k2 + 2k3 + k4)$$

4.7.1 Das allgemeine s-stufige Runge-Kutta Verfahren

$$K_n = f(x_i + c_n * h, y_i + h * \sum_{m=1}^{n-1} a_{nm} k_m) \quad n=1, \dots, s$$

$$y_{i+1} = y_i + h * \sum_{n=1}^s b_n k_n$$

s = Stufenzahl, a_{nm}, b_n, c_n = Konstanten

c_1					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
\cdot	\cdot	\cdot			
\cdot	\cdot	\cdot			
\cdot	\cdot	\cdot			
c_n	a_{n1}	a_{n2}	\dots	$a_{n,n-1}$	
\cdot	\cdot	\cdot	\dots	\cdot	
\cdot	\cdot	\cdot	\dots	\cdot	
\cdot	\cdot	\cdot	\dots	\cdot	
c_s	a_{s1}	a_{s2}	\dots	$a_{s,s-1}$	
	b_1	b_2	\dots	b_{s-1}	b_s

Euler-Verfahren,s=1

0	
	1

Mittelpunkt-Verfahren,s=2

0		
0.5	0.5	
	0	1

Modifiziertes-Verfahren,s=2

0		
1	1	
	0.5	0.5

Klass.Runge-Kutta Verfahren, s=4

0				
0.5	0.5			
0.5	0	0.5		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

```
def interpolate_runge_kutta_custom(f, x, h, y0):
    s = 4

    a = np.array([
        [0, 0, 0, 0],
        [1, 0, 0, 0],
        [1, 1, 0, 0],
        [0.75, 0.5, 0.75, 0]
    ], dtype=np.float64)
    b = np.array([0.1, 0.1, 0.4, 0.4], dtype=np.float64)
    c = np.array([0.75, 0.25, 0.75, 0.5], dtype=np.float64)

    y = np.full(x.shape[0], 0, dtype=np.float64)
    y[0] = y0

    for i in range(x.shape[0] - 1):
```

```

k = np.full(s, 0, dtype=np.float64)

for n in range(s):
    k[n] = f(x[i] + (c[n] * h), y[i] + h * np.sum([a[n][m] *
        k[m] for m in range(n - 1)]))

y[i + 1] = y[i] + h * np.sum([b[n] * k[n] for n in range(s)])

return y

```

4.8 Zurückführen DGL k-ter Ordnung auf k DGL 1.Ordnung

Beispiel (3.Ordnung):

$$y''' + 5y'' + 8y' + 6y = 10e^{-x}$$

AWP:

$$y(0) = 2, y'(0) = y''(0) = 0$$

1.Schritt:

Nach höchsten Ableitung auflösen:

$$y''' = 10e^{-x} - 5y'' - 8y' - 6y$$

2.Schritt:

Hilfsfunktionen bis (höchsten-1)Ableitung:

$$z_1(x) = y(x)$$

$$z_2(x) = y'(x)$$

$$z_3(x) = y''(x)$$

3.Schritt:

Hilfsfunktionen ableiten und in $z'_3 = y'''$ einsetzen:

$$z'_1(x) = y'(x) = (z_2)$$

$$z'_2(x) = y''(x) = (z_3)$$

$$z'_3(x) = y'''(x)$$

$$z'_3(x) = 10e^{-x} - 5z_3 - 8z_2 - 6z_1$$

4.Schritt:

DGL in vektorieller form:

$$z' = \begin{bmatrix} z'_1 \\ z'_2 \\ z'_3 \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{bmatrix} = f(x, z)$$

$$z(0) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} y(0) \\ y'(0) \\ y''(0) \end{bmatrix}$$

weil DGL 3.Ord, als LGL schreiben möglich: $z' = Az + b$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -8 & -5 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ 10e^{-x} \end{bmatrix}$$

Beispiel Eulerverfahren:

$$a = 0.$$

$$b = 1.$$

$$h = 0.1$$

$$n = \text{np.int}((b-a)/h)$$

$$\text{rows} = 4$$

```

x = np.zeros(n+1)
z = np.zeros([rows,n+1])

x[0] = a
z[:,0] = np.array([0.,2.,0.,0.])

def f(x,z):
    return np.array([z[1], z[2], z[3], np.sin(x)+5-1.1*z[3]+0.1*z[2]+0.3*z[0]])

for i in range(x.shape[0]-1):
    x[i+1]=x[i]+h
    k1 = f(x[i], z[:,i])
    k2 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k1)
    k3 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k2)
    k4 = f(x[i] + h, z[:,i] + h * k3)
    z[:,i+1] = z[:,i] + h*(1/6)*(k1+2*k2+2*k3+k4)

```

4.9 Lösen eines Systems von k DGL 1.Ordnung

Hier $y^{(i)}$ = Vektor nach i-ten Iteration!!!

Rezept Lösungsverfahren:

Beispiel

$$x_{i+1} = x_i + h$$

$$y^{(i+1)} = y^i + h * f(x_i, y^{(i)})$$

$$y''' = 10e^{-x} - 5y'' - 8y' - 6y$$

System

$$z' = \begin{bmatrix} z_1' \\ z_2' \\ z_3' \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{bmatrix} = f(x, z)$$

$$z(0) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} z_1^{(0)} \\ z_2^{(0)} \\ z_3^{(0)} \end{bmatrix}$$

i=0:

$$f(x_0, z^{(0)}) = \begin{bmatrix} z_2^{(0)} \\ z_3^{(0)} \\ 10e^{-x_0} - 5z_3^{(0)} - 8z_2^{(0)} - 6z_1^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix}$$

$$x_1 = x_0 + h = 0.5$$

$$z^{(1)} = z^{(0)} + h * f(x_0, z^{(0)}) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$$

i=1:

$$f(x_0, z^{(0)}) = \begin{bmatrix} z_2^{(1)} \\ z_3^{(1)} \\ 10e^{-x_0} - 5z_3^{(1)} - 8z_2^{(1)} - 6z_1^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0.9347 \end{bmatrix}$$

$x_2 = x_1 + h = 1$

$$z^{(2)} = z^{(1)} + h * f(x_1, z^{(1)}) = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ -1 \\ 0.9347 \end{bmatrix} = \begin{bmatrix} 2 \\ -0.5 \\ -1.4673 \end{bmatrix}$$

```

a = 0
b = 60
h = 0.1
n = int((b-a)/h)
c = 0.16
m = 1
l = 1.2
g = 9.81

rows = 2
phi0 = np.pi/2
x = np.zeros(n+1)
z = np.zeros([rows,n+1])
x[0] = a
z[:,0] = np.array([phi0,0], dtype=np.float64)

def f(x,z):
    return np.array([z[1], -((c/m)*z[1]) - (g/l)*np.sin(z[0])])

for i in range(x.shape[0] - 1):
    x[i+1]=x[i]+h
    k1 = f(x[i], z[:,i])
    k2 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k1)
    k3 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k2)
    k4 = f(x[i] + h, z[:,i] + h * k3)

    z[:,i+1] = z[:,i] + h * (1 / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)

```