Feder-Dämpfer System

In dieser Aufgabe soll das Verhalten des in Abbildung 1 dargestellten Masse-Feder-Dämpfer Systems in Matlab modelliert und simuliert werden.

Modellierung

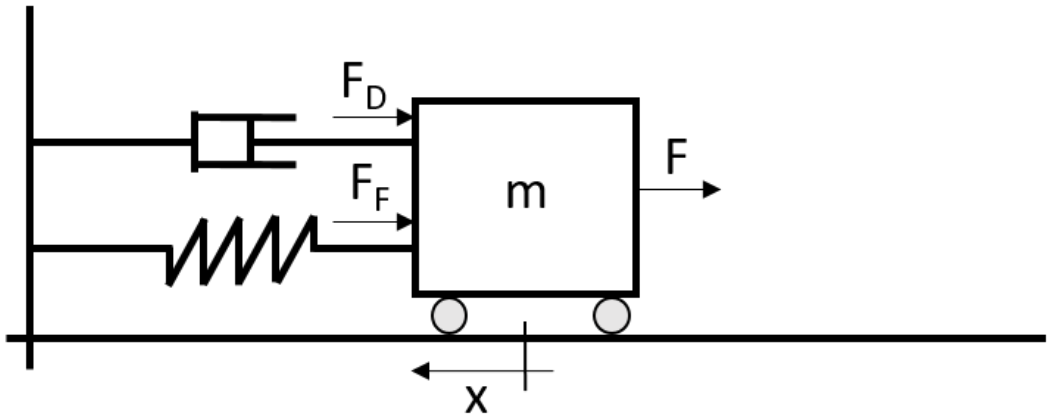

Abbildung 1: Einfaches Masse-Feder-Dämpfer System

Dazu kann zunächst unter Vernachlässigung von Reibung die folgende Kräftebilanz aufgestellt werden:

$$F_T = -F_D - F_F - F \tag{1}$$

Die Trägheitskraft F_T setzt sich als Summe aus Federkraft F_F Dämpferkraft F_D und einer äußeren angreifenden Kraft F nach Gleichung 1 zusammen.

$$F_T = m \cdot \ddot{x} \tag{2}$$
$$F_D = d \cdot \dot{x} \tag{3}$$
$$F_F = c \cdot x \tag{4}$$
$$F = F(t) \tag{5}$$

In den beschreibenden Gleichungen bezeichnet m die Masse des Objekts, d die Dämpfungskonstante, c die Federkonstante, $x = x(t)$ den Ort, $\dot{x} = \dot{x}(t)$ die Geschwindigkeit und $\ddot{x} = \ddot{x}(t)$ die Beschleunigung der Masse. Durch Einsetzen von Gleichung 2 bis 4 in 1 erhält folgende gewöhnliche Differentialgleichung zweiter Ordnung:

$$m \cdot \ddot{x} = -d \cdot \dot{x} - c \cdot x - F \tag{6}$$

Durch normieren von Gleichung 6 nach der Objektmasse m ergibt sich eine gewöhnliche Differentialgleichung zweiter Ordnung:

$$\ddot{x} = -\left(\frac{d}{m} \cdot \dot{x} + \frac{c}{m} \cdot x + \frac{F}{m}\right) \tag{7}$$

Um Matlabs ode-Solver* zum Lösen der beschreibenden Gleichung verwenden zu können, muss diese zunächst in ein System erster Ordnung überführt werden†. Mit Einführung der Variablen v als Geschwindigkeit‡, kann Gleichung 7 in ein Diffentialgleichungssystem erster Ordnung überführt werden:

$$\dot{v} = -\left(\frac{d}{m} \cdot v + \frac{c}{m} \cdot x + \frac{F}{m}\right) \tag{8}$$

$$\dot{x} = v \tag{9}$$

Mit Einführung des Zustandsvektors

$$y = \begin{pmatrix} x \\ v \end{pmatrix} \tag{10}$$

kann das Differentialgleichungssystem 8 und 9 in die allgemeinen Form eines Anfangswertproblem (AWP) 11 geschrieben werden

$$\dot{y} = f(y) = f(y(x, v)), \quad y(0) = y_0 \tag{11}$$

*ode = **o**rdinary **d**ifferential **e**quations

†Matlabs solver verlangen Systeme erster Ordnung

‡$\dot{x} = v \rightarrow \ddot{x} = \dot{v}$

Simulation

Im Folgenden werden die von MATLAB/Simulink zur Auswahl stehenden Solver aufgelistet, hinsichtlich ihrer Charakteristik eingeteilt und das zugrundeliegende numerische Verfahren vorgestellt. Allgemein lassen sich die verfügbaren Simulink-Solver bezüglich

Tabelle 1: Simulink-Solver

variable step		**fixed step**	
ode113	Adams	ode1	Euler
ode45	Dormand-Prince	ode2	Heun
ode23	Bogacki-Shampine	ode3	Bogacki-Shampine
ode15s	NDF	ode4	Runge-Kutta
ode23s	Rosenbrock	ode5	Dormand-Prince
ode23t	Trapezoidal	ode8	Dormand-Prince
ode23tb	TR-BDF2	ode14x	Euler

ihrer Schrittweite in „*fixed-step*“ und „*variable-step*“ Solver einteilen.

- Variable-step Solver passen die Integrationsschrittweite durch eine interne Schrittweitensteuerung der Dynamik des Modells so an, dass die geforderte Genauigkeit der Lösungsnäherung erreicht wird. Bei einer schnellen Änderung der Zustände wird die Schrittweite verringert und vice versa.
- Fixed-step Solver generieren bei der Integration ein konstantes Zeitgitter, da die Schrittweite für jeden Zeitschritt fest ist. Dabei werden etwaige hochfrequente Lösungsanteile immer dann nicht erfasst, wenn die Zustände sich über einen Zeitschritt schnell ändern. Es werden keine zusätzlichen Funktionsauswertungen für eine Schrittweitensteuerung oder Steifigkeitserkennung benötigt.

In Tabelle 1 sind die von Simulink zur Lösung eines AWP 11 zur Verfügung gestellten Integratoren gelistet.

Fixed-Step-Solver

Alle fixed-step solver sind Vertreter der Runge-Kutta Familie. Ein Zustand y^{j+1} wird ausgehend von dem aktuellen Zustand y^j mit einer Steigung Dy^j und der Zeitschrittweite h nach der Verfahrensvorschrift 12 berechnet.

$$y^{j+1} = y^j + h \cdot Dy^j \tag{12}$$

ode1 bis **ode8** sind *explizite Einschrittverfahren*, welche nach Verfahrensvorschrift 12 arbeiten, wobei sie sich lediglich in der Bestimmung der Steigung Dy^j unterscheiden. Die Ordnung des Verfahrens und damit die Komplexität der Bestimmung von Dy^j steigt von eins[§] (**ode1**) bis acht (**ode8**) an.

[§]Expliziter Euler

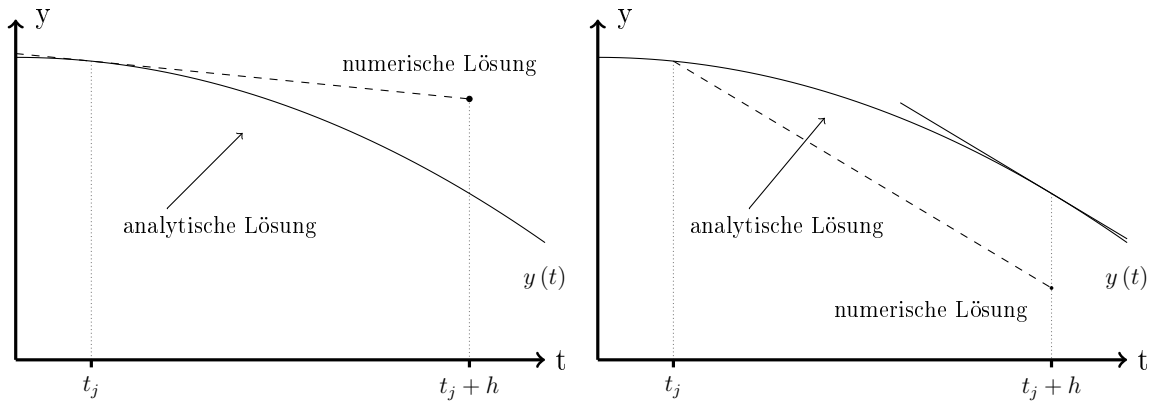

Abbildung 2: Visualisierung eines expliziten (links), impliziten (rechts) Zeitschrittes

ode14x ist ein *implizites Einschrittverfahren*, welches den Zustand zum neuen Zeitpunkt als implizite Funktion des aktuellen Zustands und der Ableitung am neuen Zustand nach der im allgemeinen nicht-linearen Gleichung 13 berechnet.

$$y^{j+1} = y^j + h \cdot Dy^{j+1} \tag{13}$$

Es kommt dabei das Newton-Verfahren zum Lösen von Gleichung 13 sowie eine ordnungssteigernde Extrapolation durch Interpolation von mit unterschiedlichen Schrittweiten generierten Lösungen zum Einsatz. Der Anwender kann die Anzahl der Newton-Iterationen sowie die Ordnung der Extrapolation vorgeben. Wird die Anzahl der Iterationen auf 1 gesetzt, erhält man für jeden Integrationsschritt das linear-implizite-Eulerverfahren. Die Ordnung der Extrapolation determiniert die Einteilung eines major-steps in minor-steps. Abbildung 3 illustriert schematisch den Zusammenhang zwischen minor-time-steps und der finalen, extrapolierten Lösung für drei Lösungen am Ende des Integrationsintervalls h. Die niedrigste Extrapolationsordnung ist linear. Je mehr Newton-Iterationen zum Lösen von Gleichung 13 aufgewendet werden, bzw. je höher die Extrapolationsordnung gewählt wird, desto mehr Rechenzeit wird benötigt, um genauere Zustandsvektoren y zu generieren. Abbildung 2 visualisiert exemplarisch den Unterschied zwischen einem expliziten und impliziten Integrationsschritt.

Variable-Step-Solver

Im Unterschied zu fixed-step-solvern sind Integratoren mit variabler Zeitschrittweite mit einer Schrittweitensteuerung ausgestattet, welche sich auf Verfahren zur lokalen Fehlerschätzung stützen. Am Ende eines jeden Zeitschritts berechnet der Solver die Zustände und deren geschätzten Fehler. Nach dem Vergleich von akzeptierbarem und berechnetem Fehler entscheidet der Solver, ob er die Schrittweite verkleinern und den Zeitstempel nochmal rechnen muss. Das ist der Fall, sobald der Fehler eines Modellzustandes den zu akzeptierenden Fehler überschreitet. Letzterer ergibt sich aus den Vorgaben von relativem ($rtol$) und absolutem Fehler ($atol$). Der relative Fehler beschreibt die prozentuale Abweichung der Näherungslösung zur entsprechenden analytischen Lösung. Dabei determiniert $rtol = 10^{-3}$ die erlaubte Abweichung der Modellzustände auf 0.1%. Die

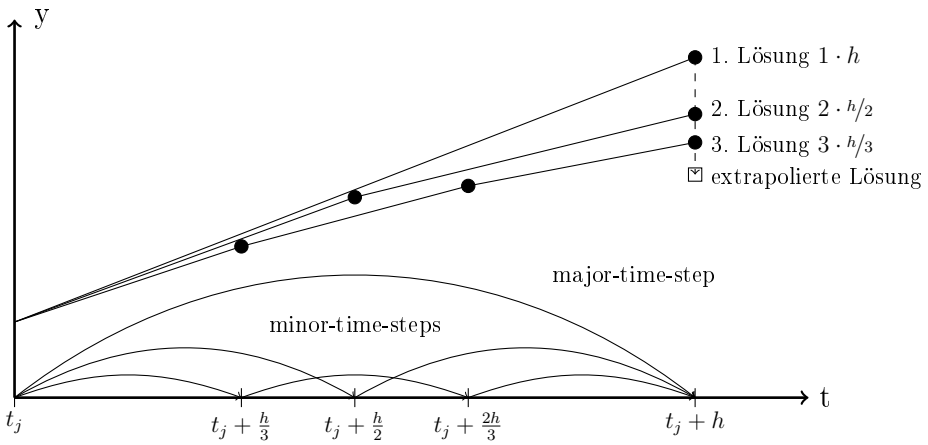

Abbildung 3: Exemplarische Darstellung der Berechnung einer extrapolierten Lösung durch Einteilung eines Zeitschrittes in minor-steps nach [9].

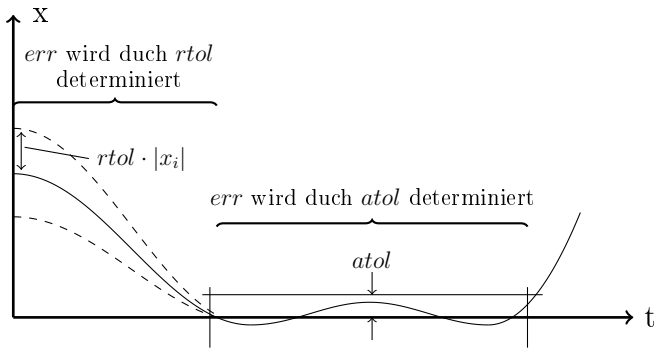

Abbildung 4: Visualisierung des relativen Fehlers *rtol* und absoluten Fehlers *atol*

absolute Toleranz ist als Grenzwert der Abweichung zu verstehen. Sie ist wichtig, wenn sich ein zu berechnender Zustand dem Wert null nähert, wodurch ($rtol$) ebenfalls zu null gedrückt wird. Damit der Solver nicht in einer Endlosschleife mit immer kleiner werdenden Schrittweite stecken bleibt, greift ab einem Grenzwert *atol*. Abbildung 4 visualisiert das Zusammenspiel der beiden Fehlertypen. Des Solvers Kriterium für den Fehler err_i ist dabei für jeden Eintrag x_i des Zustandsvektors $x \in \mathbb{R}^n$ nach Gleichung 14 zu bestimmen.

$$err_i = max\left(rtol \cdot |x_i|, atol_i\right) \tag{14}$$

Prinzipiell stehen zwei Methoden für die Implementierung einer Schrittweitensteuerung zur Verfügung.

1. Vergleich verschiedener Lösungen, welche mit unterschiedlichen Schrittweiten generiert wurden.

2. Vergleich verschiedener Lösungen, welche mit Verfahren unterschiedlicher Ordnung generiert wurden.

Die resultierende Differenz wird als Maß für den lokalen Abbruchfehler $\tilde{\delta}_{j,h}$ herangezogen. Dabei gilt nach [4]:

$$\tilde{\delta}_{j,h} = y\left(t_{j+1}; t_j, y^j\right) - y^{j+1} \approx \overline{y}^{j+1} - y^{j+1}. \tag{15}$$

Es ist dabei $y\left(t_{j+1}; t_j, y^j\right)$ die analytische Lösung zum neuen Zeitpunkt, y^{j+1} die numerische Näherung und $\overline{y}^{j+1}$ eine zweite numerische Lösung zum Ende des Integrationsintervalls. Wie die zweite Lösung generiert wird hängt von dem jeweiligen Integrationsverfahren ab.

ode23 und ode45 beruhen auf expliziten Runge-Kutta Verfahren zweiter bzw. vierter Ordnung mit einer eingebetteten dritten bzw. fünften Stufe zur Fehlerschätzung. Es wird mit dem Verfahren zweiter (vierter) Ordnung y^{j+1} berechnet, um im Anschluss mit einer zusätzlichen Funktionsauswertung den Wert $\overline{y}^{j+1}$ zu erhalten. Als explizite Einschrittverfahren erfolgt die Wahl der Koeffizienten (vgl. Abschnitt Runge-Kutta Verfahren) sowie die Implementierung nach Bogacki und Shampine [2] bzw. nach Dormand und Prince [5].

ode23s ist Vertreter der linear impliziten Runge-Kutta Familie (vgl. Abschnitt Runge-Kutta Verfahren). Im Gegensatz zu expliziten Verfahren wird bei impliziten Methoden die Jacobi Matrix benötigt, welche beim ode23s-Solver aus Stabilitätsgründen zu jedem Zeitschritt ausgewertet wird. Darüber hinaus arbeitet der Algorithmus nach dem *first same as last* (FSAL)-Prinzip. Die letzte Funtionsauswertung des vorangegangenen major-steps wird als erste Steigung für den aktuellen Integrationsschritt verwendet. Auch hier wird der lokale Abbruchfehler durch den Vergleich der Lösung aus dem Verfahren zweiter Ordnung und einer eingebetteten dritten Stufe abgeschätzt. Die explizite Verfahensvorschrift entnehme der Leser aus [11]

ode23t gehört zur Familie der impliziten Runge-Kutta Methoden. Auch bekannt unter Verfahren nach Huen (Trapezregel) besitzt dieser Solver neben der Fehlerordnung zwei eine dritte eingebettete Stufe zur Schrittweitensteuerung [10].

ode23tb entspringt der Idee, Verfahren zu konstruieren, welches die starke Stabilität von Mehrschrittverfahren, insbesondere einer Rückwärtsdifferenzenmethode, wie sie die BDF2 besitzt, jedoch ohne den Einfluss von vorausgegangenen Lösungen auf die Berechnung des neunen Zustandes auskommt. Für nicht autonome AWP's sind aufgrund einer hohen Dynamik, die von außen auf das System einwirken, Einschrittverfahren zu bevorzugen. Bei einem BDF2-Verfahren wird neben dem initialen Lösungsvektor noch eine weitere Lösung benötigt, welche mit einer Einschrittmethode (Huen) ermittelt wird. In [1] beschreiben Bank et al, wie nach einem ersten minor-step ($y^j \to y^{j+\gamma}, \gamma \in (0,1)$) mit der Trapezmethode ein zweiter minor-step ($y^{j+\gamma} \to y^{j+1}$) mit dem BDF2-Verfahren ausgeführt und dieser Prozess periodisch wiederholt werden kann. Darüber hinaus kann durch unterschiedliche Schittweiten der verschiedenen impliziten Verfahren sichergestellt

werden, dass der Effizienzvorteil einer BDF Methode, welcher sich in der Wiederverwendung einer vereinfachten Newton-Iterationsmatrix niederschlägt, beibehalten wird. Ein TR-BDF2 Code kann aufgrund der abwechselnden Anwendung eines Einschritt- und Mehrschrittverfahrens als „periodisches lineares Mehrschrittverfahren“ klassifiziert werden [6]. Bank et al. schlagen jedoch eine Fehlerschätzung vor, die ohne vergangene Zustände auskommt, weshalb ode23tb in der Literatur vorrangig als implizites Einschrittverfahren verstanden wird.

ode113 und ode15s sind Ausfühungen der linearen Mehrschrittmethoden. Ihnen liegt die Idee zugrunde, bereits berechnete Zustände und Steigungen durch ein Interpolationspolynom auf dem bekannten Intervall bis zum neuen Zeitpunkt in die Zukunft zu integrieren. Prinzipiell kann das Interpolationspolynom mit Steigungen oder Zuständen als Stützstellen gebildet werden. Dazu wird das Polynom mit Newton-Cotes-Formeln bzw. durch Lagrange-Fundamentalpolynome diskret approximiert. Die Adams-Methoden verwenden Funktionsauswertungen für das Interpolationspolynom, wobei diese sich wiederum in explizite (Adams-Bashford) und implizite Verfahren (Adams-Moulton) einteilen[¶]. Aus der Kombination von Adams-Bashford und Adams-Moulton Methoden resultiert ein Prediktor-Korrektor Verfahren (Predict– Evaluate– Correct– Evaluate: PECE)[‖], welches zunächst einen neuen Zustand mittels expliziter Vorschrift (AB) ermittelt, um diesen als Startwert der Iteration des impliziten Korrektors (AM) zu verwenden [4]. Der ode113 code stützt sich auf ein Prädiktor-Korrektor Verfahren vom Typ ABAM. Als Integrator mit variabler Schrittweite sowie variabler Ordnung (1-13) wird die maximale Ordnung durch lokale Extrapolation der Lösung erhalten. Wird das Interpolationspolynom durch bereits berechnete Zustände sowie dem neuen Zustand gebildet, erhält man die Klasse der Rückwärtsdifferenzenmethoden (Backward Differentiation Formula: BDF). Diese Methodenklasse wird erstmals von Curtiss und Hirschfelder in [3] zur Lösung steifer Systeme vorgeschlagen. ode15s arbeitet mit einem modifizierten BDF-Verfahren (Numerical Differentiation Formulas: NDF) und kann seine Ordnung variabel zwischen 1 und 5 während der Integration auf die Dynamik des Systems anpassen. Die detaillierte Beschreibung des Codes entnehme der Leser aus [11]. Eine Variation in der Schrittweite verlangt die Berechnung von zusätzlichen Punkten auf der bereits generierten Lösungstrajektorie, welche nicht in dem Zeitgitter der alten Schrittweite liegen. Dadurch ist eine Schrittweitensteuerung im Gegensatz zu Einschrittverfahren bei Mehrschrittverfahren mit zusätzlichem Aufwand verbunden.

[¶] Adams Bashford (AB): Interpolationspolynom enthält lediglich k alte Steigungen
Adams-Moulton (AM): Interpolationspolynom enthält zusätzlich neue Steigung (k+1)

[‖] auch: Adams-Bashford-Moulton Methode (ABAM)

Runge-Kutta Verfahren

In diesem Abschnitt wird zunächst die vollständige Verfahrensklasse der Runge-Kutta Familie abgeleitet, um anschließend den Spezialfall des klassischen vierstufigen Runge-Kutta Algorithmus, auf welchem der ode45 fußt zu motivieren. Ziel ist es, Einschrittverfahren mit hoher Konsistenzordnung zu konstruieren. Ein neuer Zustandsvektor $y^{j+1} \in \mathbb{R}^d$ wird ausgehend von dem aktuellen Zustand y^j per Integration der rechten Seite über den Zeitschritt $h_j = t_{j+1} - t_j$ gewonnen. Die numerische Diskretisierung des analytischen Integrals wird durch eine Quadraturformel erhalten. Für die mathematischen Grundlagen der numerischen Integration sowie die Theorie der gängigsten Quadraturmethoden sei auf die einschlägige Literatur verwiesen (z.B. [4]).

$$\begin{aligned} y^{j+1} &= y^j + \int_{t_j}^{t_{j+1}} y'(t)\,dt = y^j + \int_{t_j}^{t_{j+1}} y(t, y(t))\,dt \\ &\approx h \cdot \sum_{i=1}^{s} b_i f(t_j + c_i h, \eta_i) \end{aligned} \tag{16}$$

Die Bestimmung der Knoten c_i und Gewichte b_i erfolgt in Abhängigkeit der festgelegten Stufenanzahl s durch die zugrunde gelegte Quadraturmethode. An dieser Stelle wird mit Hinblick auf die exakte Approximation einer etwaigen auf dem Intervall h_j konstanten Funktion f gefordert, dass die Summe der Gewichte entsprechend der ersten Ordnungsbedingung für eine Quadraturformel der Ordnung** $p \geq 1$ eins ergibt.

$$\sum_{i=1}^{s} b_i = 1 \tag{17}$$

Die zu den Funktionsauswertungen benötigten Zustandsvektoren η_i werden analog zu Gleichung 16 über eine numerische Integration vom Startpunkt des Intervalls aus erhalten.

$$\eta_i = y^j + \int_{t_j}^{t_j + c_i h} f(t, y(t))\,dt \approx y^j + h \cdot \sum_{l=1}^{s} a_{il} f(t, y(t)) \tag{18}$$

Dabei werden die gleichen Quadraturpunkte wie für die Approximation in Gleichung 16 verwendet. Entsprechend Gleichung 17 soll auch diese Quadraturformel eine konstante Funktion exakt approximieren $i = 1, ..., s$.

$$\sum_{l=1}^{s} a_{il} = c_i \tag{19}$$

Im Allgemeinen beschreibt 18 ein $s \times d$-dimensionales nichtlineares Gleichungssystem, welches für die Koeffizienten a_{il} ab den Indizes $l \geq i$ mit

$$a_{il} \begin{cases} = 0 & \text{explizit} \\ \neq 0 & \text{implizit} \quad \text{ist.} \end{cases}$$

**Eine Quadraturformel besitzt die Ordnung p, falls sie exakte Lösungen für alle Polynome vom Grad $\leq p - 1$ liefert, wobei p maximal ist.

Mit $k_i := f\left(t_j + c_i h_j, \eta_i\right)$, gegebenen Koeffizienten a_{il}, b_i, c_i sowie den Gleichungen 17 und 19 ergibt sich die Runge-Kutta-Verfahrensvorschrift zu:

- Bestimmung von $k_i \in \mathbb{R}^d$ durch

$$k_i = f(t_j + c_i h_j, y^j + h_j \cdot \sum_{l=1}^{s} a_{il} k_l), i = 1, ..., s \tag{20}$$

- Bestimmung von $y^{j+1} \in \mathbb{R}^d$ durch

$$y^{j+1} := y^j + h_j \cdot \sum_{i=1}^{s} b_i k_i \tag{21}$$

Das Butcher-Tableau beinhaltet alle Koeffizienten und legt damit die Verfahrensvorschrift der entsprechenden Methode eindeutig fest [8], [7].

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}, \quad A = \begin{bmatrix} a_{11} & \dots & a_{1l} & \dots & a_{1s} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{il} & \dots & a_{is} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{s1} & \dots & a_{sl} & \dots & a_{ss} \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_s \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_s \end{bmatrix}$$

Beispiel: Klassisches Runge-Kutta-Verfahren

Das „klassische“ Runge-Kutta-Verfahren[††] ist ein vier-stufiges, explizites Runge-Kutta-Verfahren und soll hier für ein besseres Verständnis der Verfahrensklasse als Beispiel vorgestellt werden. Zudem liegt es MATLABs ode45 zugrunde. Mit einer unteren Dreiecksmatrix A ergibt sich das Butcher-Tableau zu

$$\begin{array}{c|cccc} 0 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1/2 & 0 & 0 & 1 & \\ 1 & 1/6 & 1/3 & 1/3 & 1/6 \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Die finale Steigung an der aktuellen Stelle wird nach folgender Mischungsregel berechnet.

$$\sum_{i=1}^{4} = \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$$

[††] Carl Runge und Martin Wilhelm Kutta

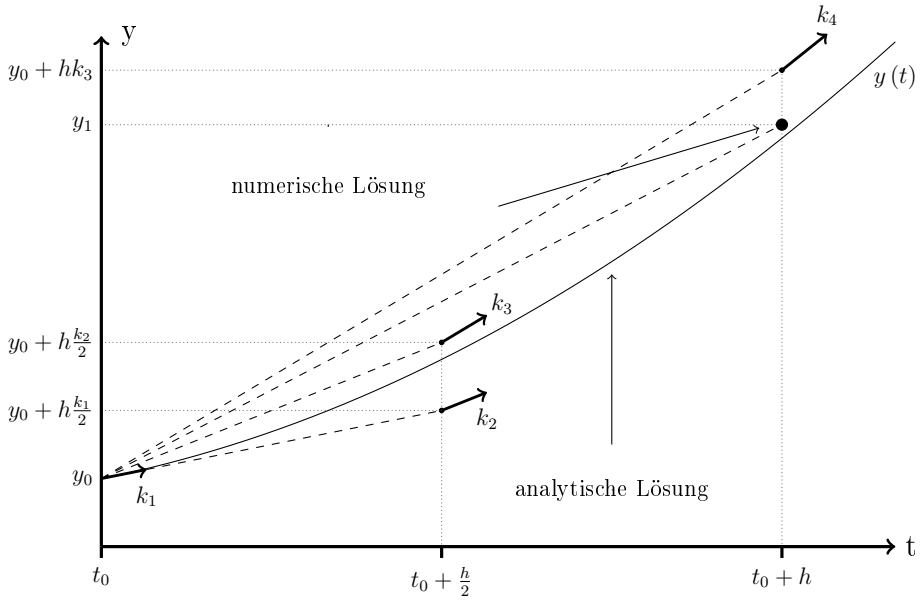

Abbildung 5: Visualisierung der Steigungen in der klassischen Runge-Kutta-Methode

Mit der fixen Zeitschrittweite h und

$$\begin{aligned} k_1 &= f\left(t_j, y^j\right) \\ k_2 &= f\left(t_j + \frac{h}{2}, y^j + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_j + \frac{h}{2}, y^j + \frac{h}{2}k_2\right) \\ k_4 &= f\left(t_j + h, y^j + hk_3\right) \end{aligned}$$

ergibt sich die Rekursionsgleichung zur Berechnung des neuen Zustands zu Gleichung 22.

$$y^{j+1} = y^j + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{22}$$

Abbildung 5 veranschaulicht die Konstruktion der finalen Steigung mittels den vier Hilfssteigungen.

Literatur

[1] Randolph Bank u. a. "Transient Simulation of Silicon Devices and Circuits". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 4 (Okt. 1985), S. 436–451. DOI: 10.1109/TCAD.1985.1270142.

[2] P. Bogacki und L.F. Shampine. "A 3(2) pair of Runge - Kutta formulas". In: *Applied Mathematics Letters* 2.4 (1989), S. 321 –325. ISSN: 0893-9659. DOI: https://doi.org/10.1016/0893-9659(89)90079-7. URL: http://www.sciencedirect.com/science/article/pii/0893965989900797.

[3] C. F. Curtiss und J. O. Hirschfelder. "Integration of Stiff Equations". In: *Proceedings of the National Academy of Sciences* 38.3 (1952), S. 235–243. ISSN: 0027-8424. DOI: 10.1073/pnas.38.3.235. eprint: https://www.pnas.org/content/38/3/235.full.pdf. URL: https://www.pnas.org/content/38/3/235.

[4] W. Dahmen und A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2008. ISBN: 9783540764939. URL: https://books.google.de/books?id=d8MfBAAAQBAJ.

[5] J.R. Dormand und P.J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1 (1980), S. 19 –26. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0771-050X(80)90013-3. URL: http://www.sciencedirect.com/science/article/pii/0771050X80900133.

[6] M E. Hosea und Lawrence Shampine. "Analysis and implementation of TR-BDF2". In: *Applied Numerical Mathematics - APPL NUMER MATH* 20 (Feb. 1996), S. 21–37. DOI: 10.1016/0168-9274(95)00115-8.

[7] E. Hairer, S.P. Nørsett und G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2008. ISBN: 9783540566700. URL: https://books.google.de/books?id=F93u7VcSRyYC.

[8] E. Hairer, S.P. Nørsett und G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Lecture Notes in Economic and Mathematical Systems. Springer, 1996. ISBN: 9783540604525. URL: https://books.google.de/books?id=m7c8nNLPwaIC.

[9] William H. Press u. a. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3. Aufl. New York, NY, USA: Cambridge University Press, 2007. ISBN: 0521880688, 9780521880688.

[10] L. Shampine, M. Reichelt und J. Kierzenka. "Solving Index-1 DAEs in MATLAB and Simulink". In: *SIAM Review* 41.3 (1999), S. 538–552. DOI: 10.1137/S003614459933425X. eprint: https://doi.org/10.1137/S003614459933425X. URL: https://doi.org/10.1137/S003614459933425X.

[11] Lawrence F Shampine und Mark W Reichelt. "The matlab ode suite". In: *SIAM journal on scientific computing* 18.1 (1997), S. 1–22.