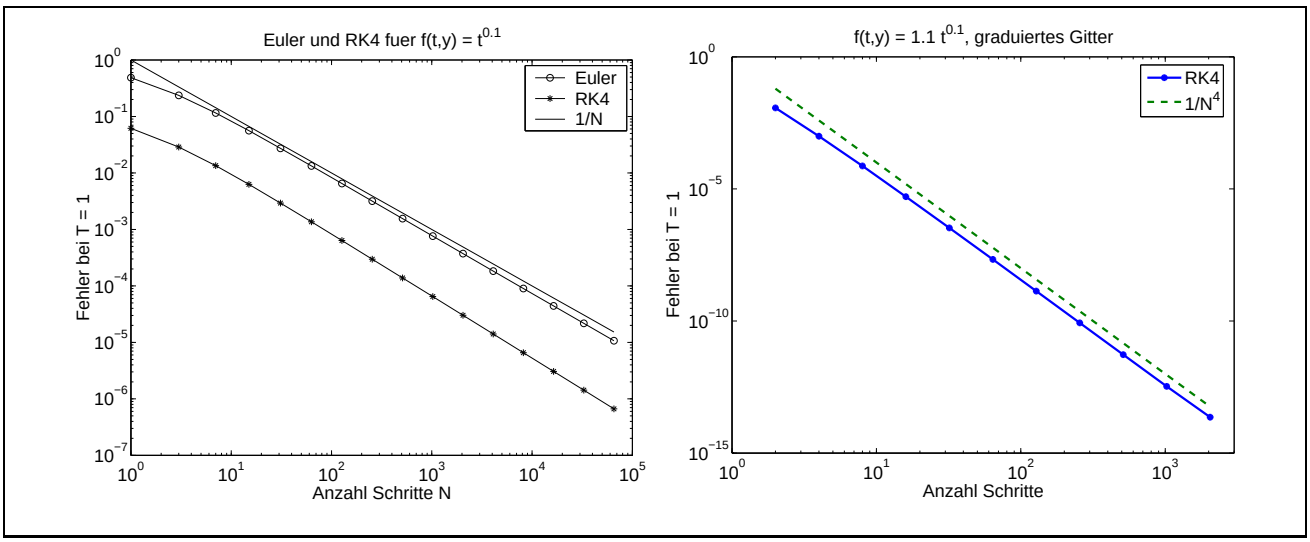


Figur 2.3: (vgl. Beispiel 2.20) Links: Konvergenzverhalten des expliziten Eulerverfahrens und des RK4 für Lösung $y(t) = t^{1.1}$; konstante Schrittweite $h = 1/N$. Rechts: RK4-auf graduierten Gittern.

In Beispiel 2.20 konnte dank präziser Kenntnis des Lösungsverhaltens eine Strategie angegeben werden, die die optimale Konvergenzrate liefert. Bei adaptiven Verfahren wie wir sie im folgenden Abschnitt kennenlernen werden, legt der Algorithmus die Knoten fest, ohne daß präzise Kenntnis über das Lösungverhalten vorliegen muß. Trotzdem wird oft bei adaptiven Algorithmen ein optimales oder fast optimales Konvergenzverhalten beobachtet.

2.4 Adaptive Verfahren mit Schrittweitensteuerung

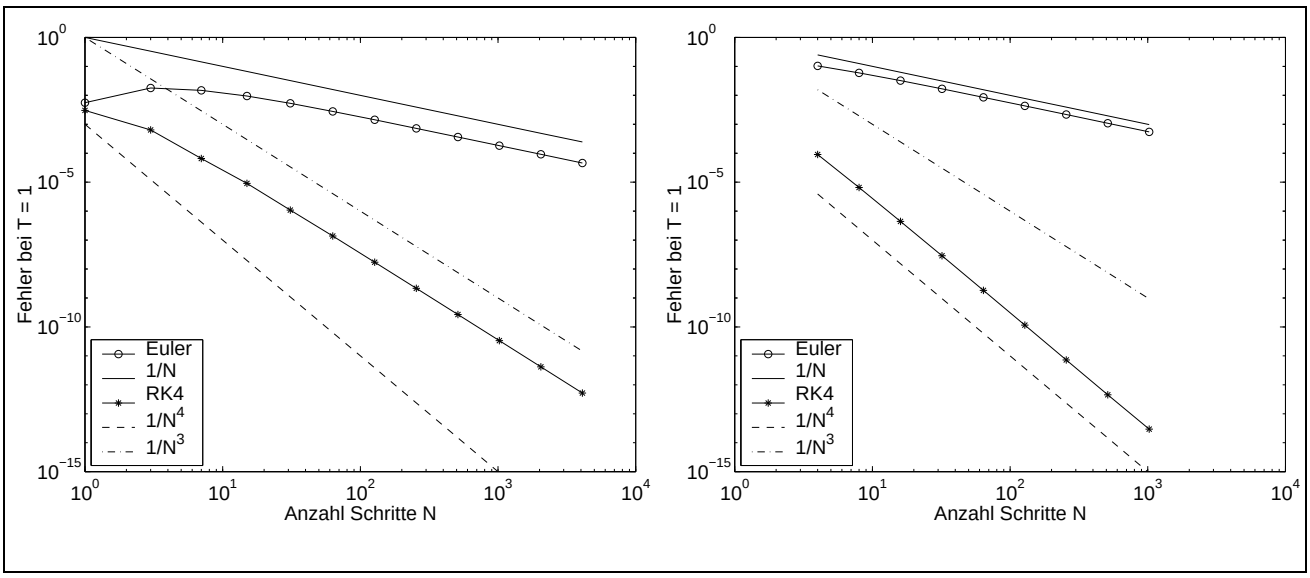
In Satz 2.10 haben wir für Einschrittverfahren mit konstanter Schrittweite h ein Konvergenzresultat vorgestellt. In der Praxis ist man aber nicht an Verfahren mit konstanter Schrittweite interessiert. Vielmehr arbeiten ausgereifte Computerprogramme zur numerischen Lösung von Anfangswertproblemen der Form (1.2) mit *adaptiven Verfahren*, bei denen der Computer in jedem Schritt eine möglichst gute Schrittweite h_i wählt, um eine vorgegebene Genauigkeit möglichst effizient zu erreichen. Gesichtspunkte bei der Entwicklung von adaptiven Verfahren sind:

1. *Effizienz:* Ein adaptives Verfahren soll mit möglichst wenig Funktionsauswertungen ein Ergebnis liefern, das (ungefähr) die vorgegebene Genauigkeit hat.

2. *Robustheit/Zuverlässigkeit:* Um eine zuverlässige numerische Lösung zu erhalten, muß die Schrittweite an das Verhalten der Lösung angepaßt sein. Legt ein Benutzer die Schrittweite selber fest, so er muß relativ detailiertes Wissen über das Verhalten der gesuchten Lösung haben, wie wir im Beispiel 2.19 gesehen haben. Bei adaptiven Verfahren paßt der Computer die Schrittweite selbständig dem lokalen Verhalten der Lösung an; moderne adaptive Verfahren sind deshalb recht zuverlässig.

finis 4.DS

Bei adaptiven Einschrittverfahren wird für jeden Knoten t_i eine Schrittweite h_i bestimmt (und damit der nächste Knoten $t_{i+1} := t_i + h_i$ festgelegt). Üblicherweise wird für eine vom Benutzer vorgegebene Toleranz $\tau_0 > 0$ eine von den beiden folgenden Strategien verwendet:



Figur 2.4: (vgl. Beispiel 2.20) Vergleich explizites Eulerverfahren und RK4. Fehler gegen Anzahl Schritte N bei nicht-glatter Lösung. Links: konstante Schrittweite h. Rechts: stückweise konstante Schrittweite, so daß $t = 1/3$ Knoten ist.

1. *Kontrolle des Fehlers pro Schritt (error per step):* Man will erreichen, daß für den Konsistenzfehler gilt $|\tau(t_i, y_i, h_i)| \simeq \tau_0$.

2. *Kontrolle des Fehlers pro Einheitsschritt (error per unit step):* Man will erreichen, daß für den Konsistenzfehler gilt $|\tau(t_i, y_i, h_i)| \simeq h_i\tau_0$.

Wir werden im weiteren die zweite Variante verwenden. Diese ist sinnvoll, wenn man davon ausgeht, daß sich die Fehler, die in jedem Schritt gemacht werden, aufsummieren. Man erwartet dann, daß der Fehler am Endpunkt $t = T$ von der Größenordnung von τ_0 ist (vgl. Bem. 2.22). Ein adaptiver Algorithmus, der auf Schrittweitensteuerung mit "Kontrolle des Fehlers pro Einheitsschritt" basiert, ist damit

Algorithmus 2.21 (Schrittweitensteuerung: Rohfassung)
% input: Toleranz τ_0
$t := t_0$
$y := y_0$
while $(t < T)$ do {
  wähle h derart, daß $|\tau(t, y, h)| \approx \tau_0 h$ % Fehler pro Schritt $\leq \tau_0$
  $y := y + h\Phi(t, y, h)$
  $t := t + h$
}

Algorithmus 2.21 ist natürlich so nicht implementierbar, weil der Konsistenzfehler $\tau(t, y, h)$ i.a. nicht explizit bestimmt werden kann. Typischerweise behilft sich damit, den Konsistenzfehler $\tau(t, y, h)$ durch Vergleich zweier verschiedener numerischer Approximationen zu schätzen. Im folgenden stellen wir zwei einfache Varianten vor.

Bemerkung 2.22 Das Konvergenzresultat aus Satz 2.10 besagt, daß unter geeigneten Annahmen an die gesuchte Lösung und die Inkrementfunktion aus Konsistenz die Konvergenz

des Verfahrens folgt. In diesem Sinn ist Algorithmus 2.21 eine sinnvolle Strategie, denn der Algorithmus erzwingt Konvergenz für $\tau_0 \to 0$. Algorithmus 2.21 kontrolliert jedoch nur den in jedem Schritt gemachten lokalen Fehler und berücksichtigt nicht seine Fortpflanzung. Der globale Fehler kann deshalb erheblich größer sein als τ_0; in Hinblick auf Satz 2.10 muß mit einem globalen Fehler der Größenordnung $Ce^{L_\Phi(T-t_0)}\tau_0$ gerechnet werden. ∎

Schrittweitensteuerung durch Extrapolation

Algorithmus 2.21 benötigt (eine Schätzung) des Konsistenzfehlers $\tau(t_0, y_0, h)$ als Funktion von h zu gegebenem (t_0, y_0). Ist Φ die Inkrementfunktion eines Verfahrens der Ordnung p, so erwarten wir, daß der Konsistenzfehler $\tau(t_0, y_0, h) = y_{t_0,y_0}(t+h) - (y_0 + h\Phi(t_0, y_0, h))$ von der Form

$$\tau(t_0, y_0, h) \approx \gamma(t_0, y_0)h^{p+1}$$

für eine Funktion $\gamma = \gamma(t_0, y_0)$ ist[5]. Bei diesem Ansatz gilt es nun, γ (für gegebenes t_0, y_0) zu bestimmen. Führt man einen *Probeschritt* mit Schrittweite H durch, so erhält man die numerische Approximation

$$\tilde{y} = y_0 + H\Phi(t_0, y_0, H),$$

und es ergibt sich damit für den Konsistenzfehler

$$y_{t_0,y_0}(t_0 + H) - \tilde{y} = \tau(t_0, y_0, H) \approx \gamma H^{p+1}. \tag{2.18}$$

Wir erhalten eine verbesserte Approximation an $y_{t_0,y_0}(t+H)$, wenn wir das Verfahren zweimal mit Schrittweite $H/2$ anwenden:

$$\widehat{y} := y_{1/2} + \frac{H}{2}\Phi(t_0 + H/2, y_{1/2}, H/2), \qquad y_{1/2} := y_0 + \frac{H}{2}\Phi(t_0, y_0, H/2),$$

Eine ganz einfache Schätzung des Konsistenzfehlers ergibt sich dadurch, daß man $\widehat{y}$ als "wesentlich genauere" Approximation an $y_{t_0,y_0}(t_0 + H)$ ansieht, d.h. man setzt

$$\tau(t_0, y_0, H) = y_{t_0,y_0}(t_0 + H) - \tilde{y} \stackrel{!}{=} \widehat{y} - \tilde{y}$$

Aus dem Ansatz $\tau(t_0, y_0, H) \approx \gamma(t_0, y_0)H^{p+1}$ ergibt sich damit $\gamma(t_0, y_0) = \frac{\widehat{y}-\tilde{y}}{H^{p+1}}$ und damit

$$\tau(t_0, y_0, h) \approx (\widehat{y} - \tilde{y}) \left(\frac{h}{H}\right)^{p+1}. \tag{2.19}$$

Grundsätzlich ist diese Idee gut, allerdings stimmt (2.19) nicht ganz. Die Differenz $y_{t_0,y_0}(t_0 + H) - \widehat{y}$ ist nämlich nicht "wesentlich kleiner" als $y_{t_0,y_0}(t_0 + H) - \tilde{y}$ sondern nur einen Faktor 2^p kleiner, wie wir nun nachrechnen[6]. Wir werden nun zeigen, daß tatsächlich gilt:

$$\tau(t_0, y_0, h) \approx \gamma h^{p+1} \approx \frac{\tilde{y} - \widehat{y}}{(1 - 2^{-p})H^{p+1}}h^{p+1}. \tag{2.20}$$

[5]Beispiel: für expl. Euler ist $\tau(t_0, y_0, h) = \frac{1}{2}y''_{t_0,y_0}(t_0)h^2 + O(h^3)$

[6]für $p = 4$ ist $2^4 = 16$, was praktisch schon "wesentlich genauer" ist!

Dies ergibt sich aus einer genaueren Betrachtung von $y_{t_0,y_0}(t_0+H)-\widehat{y}$. Führen wir die Abkürzungen $t_{1/2}=t_0+H/2$, $t_1=t_0+H$, $\overline{y}_{1/2}=y_{t_0,y_0}(t_{1/2})$ ein, so ergibt sich wie im Beweis von Satz 2.10

$$\begin{aligned}\widehat{y} &= y_{1/2}+\frac{H}{2}\Phi(t_{1/2},y_{1/2},H/2)\\ y_{t_0,y_0}(t_1) &= \overline{y}_{1/2}+\frac{H}{2}\Phi(t_{1/2},\overline{y}_{1/2},H/2)+\tau(t_{1/2},\overline{y}_{1/2},H/2)\end{aligned}$$

Nehmen wir an, daß die Inkrementfunktion differenzierbar ist, dann ergibt sich durch Taylorentwicklung und Vernachlässigung Terme höherer Ordnung mit $A:=\partial_y\Phi(t_{1/2},\overline{y}_{1/2},H/2)$

$$\begin{aligned}y_{t_0,y_0}(t_0+H)-\widehat{y} &\approx \overline{y}_{1/2}-y_{1/2}+\frac{H}{2}A(\overline{y}_{1/2}-y_{1/2})+\tau(t_{1/2},\overline{y}_{1/2},H/2)\\ &= (1+AH/2)\tau(t_0,y_0,H/2)+\tau(t_{1/2},\overline{y}_{1/2},H/2)\end{aligned}$$

Falls man annimmt, daß $A\frac{H}{2}$ klein gegenüber 1 ist, dann erhalten wir

$$y_{t_0,y_0}(t_0+H)-\widehat{y}\approx\tau(t_0,y_0,H/2)+\tau(t_{1/2},\overline{y}_{1/2},H/2).$$

Nimmt man nun an, daß τ stetig ist[7], dann können wir für kleine H erwarten

$$y_{t_0,y_0}(t_0+H)-\widehat{y}\approx 2\tau(t_0,y_0,H/2).$$

Zusammenfassend haben wir also:

$$\begin{aligned}y_{t_0,y_0}(t_0+H)-\widehat{y} &\approx 2\tau(t_0,y_0,H/2)\approx 2\gamma(t_0,y_0)\left(\frac{H}{2}\right)^{p+1}\\ y_{t_0,y_0}(t_0+H)-\tilde{y} &= \tau(t_0,y_0,H)\approx\gamma(t_0,y_0)H^{p+1}.\end{aligned}$$

Ersetzt man $\approx$ durch $=$ und bildet man die Differenz der dadurch entstehenden Gleichungen, so ergibt sich

$$\tilde{y}-\widehat{y}=\gamma(t_0,y_0)H^{p+1}(1-2^{-p}).$$

Wir erhalten somit für den Konsistenzfehler einen berechenbaren Ausdruck:

$$\tau(t_0,y_0,h)\approx\gamma(t_0,y_0)h^{p+1}\approx\frac{\tilde{y}-\widehat{y}}{(1-2^{-p})H^{p+1}}h^{p+1}. \tag{2.21}$$

Wir beobachten insbesondere, daß

$$\mathrm{EST}:=\frac{|\tilde{y}-\widehat{y}|}{1-2^{-p}}\approx|\tau(t_0,y_0,H)|=|y_{t_0,y_0}(t_0+H)-\tilde{y}| \tag{2.22}$$

einen Schätzwert für den Fehler $|y_{t_0,y_0}(t_0+H)-\tilde{y}|$ liefert. Wir suchen nun eine Schrittweite h, so daß $\tau(t_0,y_0,h)\sim h\tau_0$ gilt. Diese können wir zu

$$h:=\left(\frac{\tau_0}{\mathrm{EST}}H^{p+1}\right)^{1/p} \tag{2.23}$$

berechnen. Ein implementierbarer Algorithmus ist damit:

[7] dies folgt im Wesentlichen aus der Darstellung $\tau(t_0,y_0,h)=y_{t_0,y_0}(t+h)-(y_0+h\Phi(t_0,y_0,h))$, aus der Stetigkeit von Φ und der stetigen Abhängigkeit der Lösung y_{t_0,y_0} vom Anfangsdatum y_0

Algorithmus 2.23 (adaptive Schrittweitensteuerung mit Extrapolation)

% input: Toleranz τ_0; minimale Schrittweite h_{min}; Anfangsschrittweite h;
% input: Sicherheitsfaktor $\rho \in (0,1]$; Vergrößerungsschranke $\eta \geq 1$
% Voraussetzung: Φ ist Inkrementfunktion eines Verfahrens der Ordnung p
$t := t_0$; $y := y_0$
while $(t < T)$ do {
  $\tilde{y} := y + h\Phi(t, y, h)$
  $y_{1/2} := y + \frac{h}{2}\Phi(t, y, h/2)$, $\quad \widehat{y} := y_{1/2} + \frac{h}{2}\Phi(t + h/2, y_{1/2}, h/2)$
  $\mathrm{EST} := \frac{|\tilde{y} - \widehat{y}|}{1 - 2^{-p}}$ % schätze Fehler
  if $\Big((\mathrm{EST}/h) \leq \tau_0$ or $h \leq h_{min}\Big)\{$ % Genauigkeit oder minimale Schrittweite erreicht
    $y := \widehat{y}$ % akzeptiere beste Approximation $\widehat{y}$; $y_{i+1} := \widehat{y}$
    $t := t + h$ % nächster Knoten: $t_{i+1} = t_i + h$
    $h := \max\left\{h_{min}, \min\left\{\eta h, \rho\left(\frac{\tau_0}{\mathrm{EST}} h^{p+1}\right)^{1/p}\right\}\right\}$ % neuer Schrittweitenvorschlag
    if $t + h > T$, then $h := T - t$ % stelle sicher, daß letzter Schritt bei T endet
  }
  else $h := h/2$ % andernfalls: verwerfe Probeschritt und probiere $h/2$
}

Algorithmus 2.23 bedarf einiger Erläuterungen. Sein wesentliches Element ist die Schrittweitensteuerung

$$h := \max\left\{h_{min}, \min\left\{\eta h, \rho\left(\frac{\tau_0}{\mathrm{EST}} h^{p+1}\right)^{1/p}\right\}\right\}. \tag{2.24}$$

Diese Wahl der Schrittweite ist weitgehend durch (2.23) motiviert und soll die gewünschte Beziehung $|\tau(t_i, y_i, h_i)| \sim h_i\tau_0$ gewährleisten. Die weiteren Terme in (2.24) sind durch folgende Überlegungen motiviert:

1. Da man die Anzahl Funktionsauswertungen möglichst klein halten will, ist man daran interessiert, das Ergebnis des Schrittweitenvorschlags zu akzeptieren. Die Funktion des Sicherheitsfaktors $\rho \in (0, 1]$ ist deshalb, die Wahrscheinlichkeit zu erhöhen, daß der vorgeschlagene Schritt direkt akzeptiert wird.

2. Die Wahl der Schrittweite in (2.23) erlaubt auch, daß $h > H$, d.h. eine Vergrößerung der Schrittweite. Man wird die Schrittweite sinnvollerweise nicht zu schnell wachsen lassen. Der Faktor $\eta \geq 1$ kontrolliert, wie schnell die Schrittweite wachsen kann. Desgleichen kann man die Schrittweite nach oben beschränken.

3. Um sicherzustellen, daß das Verfahren sich nicht "festfrißt", kann man z.B. die Schrittweite nach unten beschränken wie in Algorithmus 2.23 gemacht wird.

Im wesentlichen geht Algorithmus 2.23 so vor: Aus dem vorangegangen Schritt liegt ein Schrittweitenvorschlag h vor. Es wird nun ein Probeschritt mit dieser Schrittweite gemacht und der Fehler geschätzt. Ist die erhaltene Approximation aus dem Probeschritt genau genug, wird sie akzeptiert und ein Schrittvorschlag für den nächsten Schritt wird mithilfe von (2.24) bestimmt. Ist die erhaltene Approximation zu ungenau, dann wird die Schrittweite halbiert und der Probeschritt wiederholt (t wird nicht verändert!).

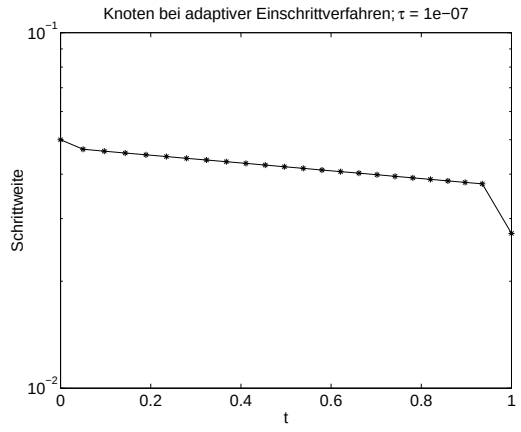
Bemerkung 2.24 (2.22) zeigt, daß die Fehlerschätzung EST in Algorithmus 2.23 genaugenommen nur den Fehler $|\tilde{y} - y(t+h)|$ schätzt. Da wir aber annehmen, daß $\widehat{y}$ eine bessere Approximation an das gesuchte $y(t+h)$ ist, wäre es widersinnig, diese nicht zu verwenden. Dies wurde in Algorithmus 2.23 auch gemacht.
Der Schrittweitenvorschlag ist eigentlich die "korrekte" Schrittweite für den akzeptierten Schritt. Unter der Annahme, daß sich die korrekte Schrittweite von Schritt zu Schritt nur wenig ändert, ist es jedoch eine sinnvolle Strategie.
Es muß noch eine Startschrittweite gewählt werden. Für ein Verfahren der Ordnung p ist eine sinnvolle Wahl ist $h = \rho' \tau^{1/p}$, wobei der Sicherheitsfaktor ρ' z.B. als $\rho' = 0.1$ gewählt wird. ∎

Abschließend zeigen wir numerisch das Verhalten von Algorithmus 2.23. In den folgenden Beispielen 2.25–2.27 wird das RK4-Verfahren aus Beispiel 2.16 eingesetzt; die weiteren Parameter von Algorithmus 2.23 sind $\rho = 0.8$, $\eta = 2$, $h_{min} = \tau_0$ und Startschrittweite $h = 0.1$.

Beispiel 2.25 Wir verwenden den adaptiven Algorithmus für das Problem

$$y'(t) = y(t), \qquad y(0) = 1.$$

Gesucht ist der Wert $y(1)$. Fig. 2.5 zeigt das Verhalten des Algorithmus. Da die gesuchte Lösung keine starken Variationen auf dem Intervall $[0, 1]$ aufweist, erwarten wir, daß eine ungefähr uniforme Schrittweite die effizienteste Schrittweitenwahl ist. Wie man für $\tau_0 = 10^{-7}$ in Fig. 2.5 sieht, ist dies in der Tat der Fall. Die Tabelle in Fig. 2.5 zeigt die Anzahl benötigter Schritte und den tatsächlich erreichten Fehler. ∎



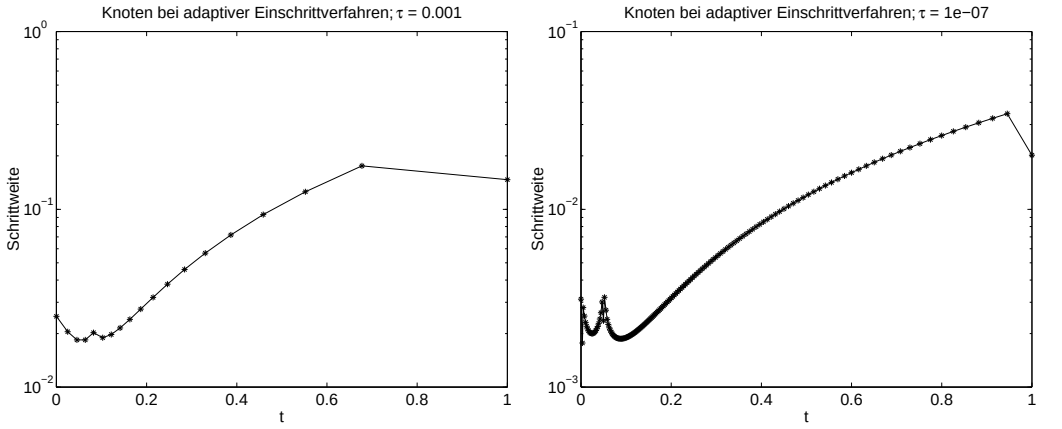
Toleranz τ_0	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
Anzahl Schritte	5	8	14	24	43	77
Anzahl rejects	0	0	0	1	2	3
Fehler	0.91E-5	0.48E-6	0.54E-7	0.50E-8	0.45E-9	0.44E-10

Figur 2.5: Adaptives RK4-Verfahren mit glatter Lösung $y(t) = e^t$. Oben: Knoten und lokale Schrittweite für $\tau_0 = 10^{-7}$. Unten: Verhalten des adaptiven Algorithmus für verschiedene Toleranzen τ_0.

Beispiel 2.26 Wir verwenden den adaptiven Algorithmus für das Problem

$$y'(t) = -200ty^2(t), \qquad y(0) = 1.$$

mit exakter Lösung $y(t) = \frac{1}{1+100t^2}$. Gesucht ist der Wert $y(1)$. Die Lösung variiert relativ stark im Intervall $[0, 1]$, und wir erwarten, daß Algorithmus 2.23 die Schrittweite entsprechend anpaßt. Wie man für $\tau_0 = 10^{-3}$ und $\tau_0 = 10^{-7}$ in Fig. 2.6 sieht, unterscheiden sich maximale und minimale Schrittweite um über einen Faktor 10. Die Tabelle in Fig. 2.5 zeigt wieder die Anzahl benötigter Schritte und den tatsächlich erreichten Fehler. ∎



Toleranz τ_0	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
Anzahl Schritte	33	55	93	163	286	506
Anzahl rejects	5	6	6	7	8	8
Fehler	0.20E-6	0.27E-7	0.10E-7	0.45E-9	0.85E-10	0.48E-11

Figur 2.6: Adaptives RK4-Verfahren bei Lösung $y(t) = 1/(1 + 100t^2)$. Oben: Knoten und lokale Schrittweite für $\tau_0 = 10^{-3}$ und $\tau_0 = 10^{-7}$. Unten: Verhalten des adaptiven Algorithmus für verschiedene Toleranzen τ_0.

Beispiel 2.27 Wir verwenden den adaptiven Algorithmus für das Problem

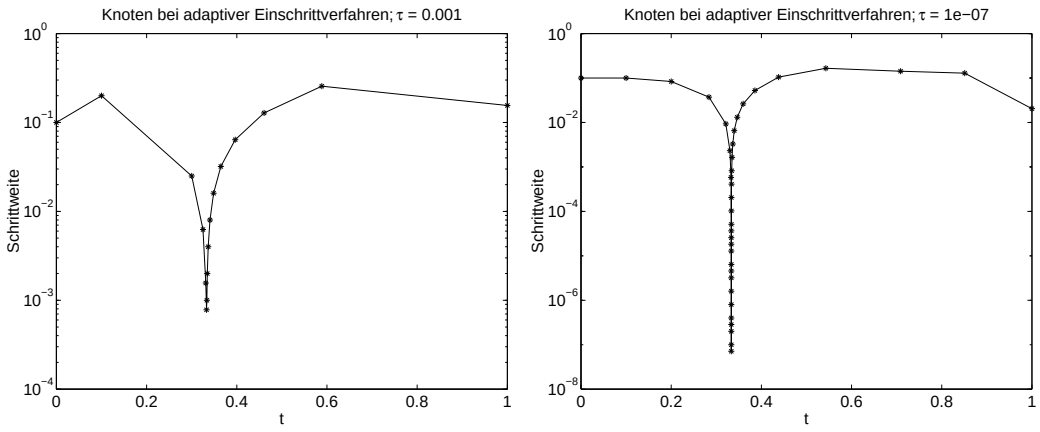
$$y'(t) = f(t) \qquad y(0) = 0, \qquad f(t) = \begin{cases} \sin t & t \le 1/3 \\ \sin(1/3 - t) & t > 1/3. \end{cases}$$

Gesucht ist der Wert $y(1)$. Fig. 2.7 zeigt das Verhalten von Algorithmus 2.23. Hier erwarten wir, daß der Algorithmus "erkennt", daß die gesuchte Lösung nur stückweise glatt ist und daß eine Verfeinerung bei $t = 1/3$ benötigt wird. Wie man für $\tau_0 = 10^{-3}$ und $\tau_0 = 10^{-7}$ in Fig. 2.7 sieht, ist dies in der Tat der Fall. Algorithmus 2.23 wurde mit $h_{min} = \tau_0$ angewendet, was die Begrenzung der Schrittweite in Fig. 2.7 erklärt. Wird keine Schrittweitenbegrenzung eingesetzt, dann "frißt" sich der Algorithmus bei $t = 1/3$ fest und erreicht überhaupt nicht $t = T$. Die Tabelle in Fig. 2.7 gibt wieder die Anzahl benötigter Schritte und den tatsächlich erreichten Fehler an. ∎

finis 5.DS

Schrittweitensteuerung mit eingebetteten Runge-Kutta-Verfahren

Wesentlich bei Algorithmus 2.21 ist, zu gegebenem t, y den Konsistenzfehler $h \mapsto \tau(t, y, h)$ zu schätzen. Oben wurde hierzu das Verfahren mit Schrittweite h und $h/2$ angewendet, um



Toleranz τ_0	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
Anzahl Schritte	21	27	34	37	45	70
Anzahl rejects	17	23	29	31	34	44
Fehler	0.25E-5	0.12E-6	0.48E-7	0.32E-8	0.84E-10	0.17E-10

Figur 2.7: Adaptives RK4-Verfahren bei stückweise glatter Lösung. Oben: Knoten und lokale Schrittweite für $\tau_0 = 10^{-3}$ und $\tau_0 = 10^{-7}$. Unten: Verhalten des adaptiven Algorithmus für verschiedene Toleranzen τ_0.

zwei verschiedene Approximationen zu erhalten, die miteinander verglichen werden konnten. Der Nachteil der obigen Vorgehensweise ist, daß, falls der Schritt akzeptiert wird, zusätzliche Funktionsauswertungen gemacht wurden, nämlich diejenigen, die zum Schritt mit Schrittweite h gehören. Um die Anzahl Funktionsauswertungen gering zu halten, werden deshalb *eingebettete Runge-Kutta-Verfahren* eingesetzt. Ein wichtiger Vertreter solcher Verfahren ist das RK5(4)-Verfahren ("DOPRI5") von Dormand & Prince:

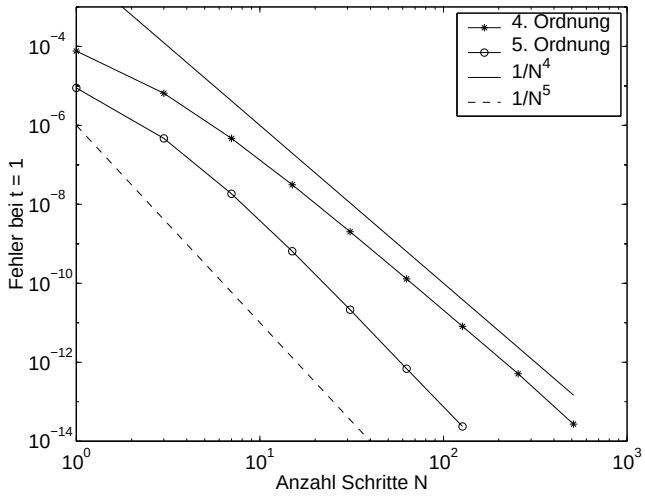
Beispiel 2.28 (RK5(4) von Dormand & Prince) Bei eingebetteten Runge-Kutta-Verfahren werden dieselben Werte k_i, $i = 1, \ldots, s$, auf zwei verschiedene Arten linear kombiniert, d.h. die entsprechenden Tableaus unterscheiden sich nur in der Wahl der Parameter b_i, $i = 1, \ldots, s$. Als Tableaus werden solche Verfahren üblicherweise notiert wie in Fig. 2.8 für das RK5(4)-Verfahren mit $s = 7$ Stufen von Dormand und Prince, [3], vorgeführt: die untersten beiden Zeilen geben die Werte b_i an. In Fig. 2.8 liefert die erste Wahl der Parameter b_i ein Verfahren 5. Ordnung während die zweite Wahl ein Verfahren 4. Ordnung ergibt. Damit ergibt sich eine einfache Möglichkeit, zwei verschiedene Approximationen (mit unterschiedlicher Genauigkeit) zu erzeugen, die zur Schrittweitensteuerung miteinander verglichen werden können.
In Fig. 2.9 illustrieren wir diese Möglichkeiten für das Anfangswertproblem $y'(t) = y(t)$ mit $y(0) = 1$ mit konstanter Schrittweite $h = 1/N$: Verwenden des ersten Satzes von Parametern b_i liefert ein Verfahren 5. Ordnung während des zweite Satz von Parametern nur ein Verfahren 4. Ordnung ergibt. ∎

Algorithmus 2.29 (adaptive Schrittweitensteuerung mit eingebetteten RK-Verfahren)

% input: Toleranz τ_0; minimale Schrittweite h_{min}; Anfangsschrittweite h;
% input: Sicherheitsfaktor $\rho \in (0, 1]$; Vergrößerungsschranke $\eta \geq 1$

0								
$\frac{1}{5}$	$\frac{1}{5}$							
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$						
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$					
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$				
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$			
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$		
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	(5. *Ordnung*)
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$	(4. *Ordnung*)

Figur 2.8: Das RK5(4) Verfahren von Dormand & Prince.



Figur 2.9: Fehlerverhalten beim eingebetteten RK 5(4)-Verfahrens bei Lösung $y(t) = e^t$: Die Stufen können zu einem Verfahren der Ordnung 5 und zu einem Verfahren der Ordnung 4 kombiniert werden.

```
% Voraussetzung:  Φ ist Inkrementfunktion eines eingebetteten RK Verfahrens
%                  mit Ordnungen p+1, p
t := t_0;  y := y_0
while (t < T) do {
   bestimme Approximationen ỹ_{p+1}, ỹ_p mit einem eingebetteten RK Verfahren
   EST := |ỹ_{p+1} − ỹ_p|                    % schätze Fehler des schlechteren Verfahrens
   if ((EST/h) ≤ τ_0 or h ≤ h_min){          % Genauigkeit oder minimale Schrittweite erreicht
     y := y_{p+1}                            % akzeptiere beste Approximation ỹ_{p+1}; y_{i+1} := ỹ_p
     t := t + h                              % nächster Knoten: t_{i+1} = t_i + h
     h := max{h_min, min{ηh, ρ(τ_0/EST · h^{p+1})^{1/p}}}   % neuer Schrittweitenvorschlag
     if t + h > T, then h := T − t           % stelle sicher, daß letzter Schritt bei T endet
   }
```

`else` $h := h/2$ `%` andernfalls: verwerfe Probeschritt und probiere $h/2$
}

Die Grundidee von Algorithmus 2.29 ist derjenigen von Algorithmus 2.23 sehr ähnlich. Das eingebettete RK-Verfahren Φ liefert zwei Approximationen $\tilde{y}_{p+1}$, $\tilde{y}_p$ mit Konsistenzordnungen $p+1$ und p. Nimmt man $\tilde{y}_{p+1}$ als wesentlich genauere Approximation an, so ergibt sich damit für den Konsistenzfehler τ, der zum (ungenaueren) Verfahren der Ordnung p gehört:

$$\tau(t, y, h) \approx \gamma h^{p+1}, \qquad \gamma = \frac{\mathrm{EST}}{h^{p+1}}.$$

Für akzeptierte Schritte wird damit der Schrittweitenvorschlag für den nächsten Schritt so gemacht, daß er "richtig" für das Verfahren niedrigerer Ordnung ist.

Bemerkung 2.30 Die beiden vorgestellten adaptiven Algorithmen illustrieren nur die Grundideen der Schrittweitensteuerung. Modernen, ausgereiften Computerprogrammen steht eine reichhaltige Bibliothek von Verfahren und Fehlerschätzern zur Verfügung. Sie steuern damit nicht nur die Schrittweite nach Varianten des hier vorgestellten Prinzips, sondern erhöhen auch die Verfahrensordnung, wenn der Übergang zu einem Verfahren hoher Ordnung effizienter erscheint. Umgekehrt schalten sie auf billigere Verfahren niedriger Ordnung herunter, wenn sie erkennen, daß ein Verfahren hoher Ordnung ineffizient ist. Wird erkannt, daß eine steife Differentialgleichung vorliegt, wechseln sie zudem zu impliziten Verfahren. ∎