Erzeugung von eingebetteten Runge-Kutta-Verfahren hoher Ordnung z.B. 10(8), 12(10) und 14(12) durch Modifikation bekannter Verfahren ohne Erhöhung der Stufenzahl

Dieter Kaiser
Friedrich-Schiller-Universität Jena, Numerische Mathematik
Ernst-Abbe-Platz 2, D 07743 Jena
E-mail: dieter.kaiser@uni-jena.de

1.Einleitung

Wir betrachten Anfangswertprobleme (AWP) für Systeme von Differentialgleichung 1. Ordnung der Form

$$y'(x) = f(x,y), \qquad y(x_0) = y_0, \qquad y(x) \in \mathbb{R}^n .$$

Diese können mit expliziten s-stufigen Runge-Kutta-Verfahren (RKV) definiert im ersten Schritt durch

$$k_i = f(x_0 + c_i h, y_0 + h\sum_{j=1}^{i-1} a_{i,j} k_j) \qquad i = 1,\dots,s$$

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i$$

numerisch gelöst werden.
Die jeweiligen konkreten (expliziten) RKV werden meist durch ein Butcher-Schema der Form

$$\begin{array}{c|cccc}
0 & 0 & \cdots & 0 & 0 \\
c_2 & a_{2,1} & \ddots & & \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
c_s & a_{s,1} & \cdots & a_{s,s-1} & 0 \\
\hline
 & b_1 & \cdots & b_{s-1} & b_s
\end{array}$$

oder kurz

$$\begin{array}{c|c}
c & A \\
\hline
 & b^T
\end{array}$$

angegeben.

2. Eingebettete Verfahren

Bei der Anwendung der Verfahren steht das Problem, in jedem Schritt h so zu wählen, dass die numerische Lösung auf dem gesamten Intervall $[x_0, T]$, auf dem die Lösung numerisch tabelliert werden soll, eine vorgegebene Genauigkeit *tol* einhalten soll. Ein Maß für die Genauigkeit des

RKV ist die Ordnung. Ein RKV hat die Ordnung p, wenn für den lokalen Fehler gilt

$$y_1 - y(x_0 + h) = O\left(h^{p+1}\right),$$

wobei $y(x)$ die exakte Lösung ist.
Zur Schätzung des lokalen Fehlers können zwei Verfahren (RKVp(q)) mit den Ordnungen q und p (q<p) eingesetzt werden. Um numerischen Aufwand zu sparen, sind dies meist sogenannte eingebettete RKV mit dem Butcher-Schema

$$\begin{array}{|c|ccccc|}
\hline
0 & 0 & \cdots & 0 & 0 & 0 \\
c_2 & a_{2,1} & \ddots & 0 & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s-2} & 0 & 0 \\
\hline
c_s & a_{s,1} & \cdots & a_{s,s-2} & a_{s,s-1} & 0 \\
\hline
 & \hat{b}_1 & \cdots & \hat{b}_{s-2} & \hat{b}_{s-1} & 0 \\
\hline
 & b_1 & \cdots & b_{s-2} & b_{s-1} & b_s \\
\hline
\end{array}$$

und

$$\hat{y}_1 = y_0 + h\sum_{i=1}^{s-1} \hat{b}_i k_i$$

sowie damit die Schätzung des lokalen Fehlers *est* durch

$$est = \frac{\|y_1 - \hat{y}_1\|}{|h|}.$$

Ist $est < tol$ (+) so wird der Schritt angenommen, sonst (-) wird er mit einer neuen Schrittweite h_{neu} wiederholt.
Die Steuerung der Schrittweite erfolgt in beiden Fällen ((+),(-)) meist nach der Formel

$$h_{neu} = h \bullet 0.9\left(\frac{tol}{est}\right)^{\frac{1}{q+1}}.$$

In der Literatur und in der Programmierpraxis gibt es eine Reihe solcher Verfahren. Dabei ist fast immer q=p-1.

3. Verfahren höherer Ordnung mit Fehlerschätzer

Bereits 1975 entwickelte Hairer [2,7] ein Verfahren der Ordnung 10 (s=17). Seit dem sind weitere Verfahren der Ordnung 10 z.B. [9],[11] aber auch der Ordnung 12 (s=25) [13],[15] und sogar 14 (s=35) [17] entwickelt worden.
In [6] wurden nun von Stone auf der Basis dieser RKV Verfahren mit Fehlerschätzer angegeben. Allerding sind dies keine eingebetteten Verfahren im Sinne der o.g. Form, sondern es wurden weitere Stufen hinzugefügt, um ein Verfahren niedrigerer Ordnung, meist q=p-2 oder q=p-3, einzuarbeiten‘. Diese Verfahren kann man deshalb als ‚aufgebettet‘ bezeichnen.
Das Butcher-Schema dafür hat das Aussehen:

$$
\begin{array}{c|cccc}
0 & 0 & \cdots & 0 & 0 \\
c_2 & a_{2,1} & \ddots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
c_r & a_{r,1} & \cdots & a_{r,r-1} & 0 \\
\hline
 & b_1 & \cdots & b_{r-1} & b_r \\
\hline
 & b_1^* & \cdots & b_{r-1}^* & b_r^*
\end{array}
$$

Das Schema enthält also das Originalverfahren und deshalb stimmen im Vektor $b^T = \left(b_{1,}, \ldots, b_s, b_{s+1}, \ldots, b_r\right)$ (meist) die ersten s Elemente mit dem Original überein und die restlichen Elemente sind Null.
Die Schätzung des lokalen Fehlers erfolgt nun analog der eingebetteten Verfahren mit

$$k_i = f(x_0 + c_{s+1}h, y_0 + h\sum_{j=1}^{i-1} a_{s+1,j}k_j) \qquad i = s+1, \ldots, r,$$

$$y_1^* = y_0 + h\sum_{i=1}^{r} b_i^* k_i$$

und

$$est^* = \frac{\left\| y_1 - y_1^* \right\|}{|h|}$$

sowie

$$h_{neu} = h \bullet 0.9 \left(\frac{tol}{est^*} \right)^{\frac{1}{q+1}} .$$

Wie man im Test sieht, funktionieren diese Verfahren sehr gut.
Die Erhöhung der Stufenzahl bringt allerdings auch eine Erhöhung des numerischen Aufwandes mit sich.

4. Neue Verfahren höherer Ordnung mit Fehlerschätzer ohne Stufenzahlerhöhung

Beim Betrachten der meisten konkreten Verfahren aus [6] fällt auf, dass der Vektor $\left(b_1, \ldots, b_s\right)^T$ eine teilweise Symmetrie aufweist. Angenommenes es ist $b_2 = -b_{s-1}$, dann liegt die Idee nahe, dass wir nun

$$\begin{aligned} \tilde{b}_i &= b_i \qquad i = 1, 3, \ldots, s-2, s \\ \tilde{b}_2 &= -\tilde{b}_{s-1} = c \end{aligned} \qquad (*)$$

wählen, und c so, dass die Überprüfung der Konsistenzbedingungen für das abgeänderte Verfahren die Ordnung q=p-2 ergibt.
Dies ist gelungen!

Das Butcher-Schema hat nun mit (*) folgendes Aussehen:

$$
\begin{array}{c|cccc}
0 & 0 & \cdots & 0 & 0 \\
c_2 & a_{2,1} & \ddots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
c_r & a_{s,1} & \cdots & a_{s,s-1} & 0 \\
\hline
 & b_1 & \cdots & b_{s-1} & b_s \\
\hline
 & \tilde{b}_1 & \cdots & \tilde{b}_{s-1} & \tilde{b}_s
\end{array}
$$

mit

$$\tilde{y}_1 = y_0 + h\sum_{i=1}^{s} \tilde{b}_i k_i$$

und

$$e\tilde{s}t = \frac{\|y_1 - \tilde{y}_1\|}{|h|}$$

sowie

$$h_{neu} = h \bullet 0.9 \left(\frac{tol}{e\tilde{s}t}\right)^{\frac{1}{q+1}}$$

erhalten wir nun ein wirklich eingebettetes Verfahren, wenn man davon absieht, dass in 2. $\hat{b}_s = 0$ gilt und damit das eingebettete Verfahren ein eigenes Verfahren des Stufe s-1 und Ordnung q=p-1 ist. Hier hat das eingebettete Verfahren die Stufe s und die Ordnung q=p-2. Dies führt aber zu keinem erhöhten Aufwand bzgl. der Aufrufe der Differentialgleichung!

5. Konkrete Verfahren

Die Verfahren wurden in Matlab umgesetzt und auch getestet.
Hier nun die Aufstellung der entwickelten bzw. getesteten Verfahren:

- Rktf14(12)dk - 35stufiges Verfahren von Terry Feagin [17] der Ordnung 14 mit 12er Einbettung der selben Stufe von Dieter Kaiser,
- Rktf12(10)dk – 25stufiges Verfahren von Terry Feagin [17] der Ordnung 12 mit 10er Einbettung der selben Stufe von Dieter Kaiser,
- Rktf12(9) – 25stufiges Verfahren von Terry Feagin [13]der Ordnung 12 mit 10er ‚Aufbettung' der Stufe 29 von Peter Stone [14],
- Rkho12(10)dk – 25stufiges Verfahren von Hiroshi Ono [15] der Ordnung 12 mit 10er Einbettung der selben Stufe von Dieter Kaiser,
- Rkthof12(9) – 25stufiges Verfahren von Hiroshi Ono [15] der Ordnung 12 mit 10er ‚Aufbettung' der Stufe 29 von Peter Stone [16],
- Rkeh10(8)dk – 17stufiges Verfahren von Ernst Hairer [7] der Ordnung 10 mit 8er Einbettung der selben Stufe von Dieter Kaiser,
- Rkeh10(8) – 17stufiges Verfahren von Ernst Hairer [7] der Ordnung 10 mit 8er ‚Aufbettung' der Stufe 20 von Peter Stone [8],

- Rktf10(8)dk - 17stufiges Verfahren von Terry Feagin [9] der Ordnung 10 mit 8er Einbettung der selben Stufe von Dieter Kaiser,
- Rktf10(8) – 17stufiges Verfahren von Terry Feagin [9] der Ordnung 10 mit 8er ‚Aufbettung' der Stufe 20 von Peter Stone [10],
- Rkho10(8)dk – 17stufiges Verfahren von Hiroshi Ono [11] der Ordnung 10 mit 8er Einbettung der selben Stufe von Dieter Kaiser,
- Rkthof10(8) – 17stufiges Verfahren von Hiroshi Ono [11] der Ordnung 10 mit 8er ‚Aufbettung' der Stufe 20 von Peter Stone [12],
- ode45 – Standard-Matlab-Programm

Die folgende Tabelle zeigt die Änderung c bei den jeweiligen ‚dk-Programmen':

Verfahren	Rktf14(12)dk	Rktf12(10)dk	Rkho12(10)dk	Rkeh10(8)dk	Rktf10(8)dk	Rkho10(8)dk
c	-1/512	-1/128	.11	-4/13	-.025	0

Für die damit erzeugten RKV konnten mit Hilfe des erzeugten Matlab-Programmes rkorderfnum.m und rkordersym.m [4] sowohl die originalen Ordnungen p als auch die Ordnungen q=p-2 der modifizierten RKV nachgewiesen werden. Dieses Programm basiert auf einer Arbeit von Fritsche [1], die ein C-Programm liefert, das die Konsistenzbedingungen in einer codierten Form erzeugt. Diese wurde in Matlab-Code umgewandelt. Auf Grund der Zahlendarstellung im Computer sind allerdings einige Konstanten ab Ordnung 13 außerhalb der Integerdarstellung von C. Diese wurden nachgerechnet und im Matlab-Programm umgesetzt.
Z.B. lautet die letzte Bedingung der **53272** Bedingungen für die Ordnung 14 in diesem Programm sinngemäß:

```
kk=A^14*ones(36,1); if abs(kk(end)*87178291200 -1)>tol, error(),end.
```

Dabei ist $A=\begin{bmatrix} a_{11} & \cdots & a_{1s} & 0 \\ \vdots & & \vdots & \vdots \\ a_{s1} & \cdots & a_{ss} & 0 \\ \bar{b}_1 & \cdots & \bar{b}_s & 0 \end{bmatrix}$, $\bar{b}_i = b_i$ oder $\bar{b}_i = \tilde{b}_i$ $i = 1,...,s$ je nach Verfahren.

Es kann sowohl symbolisch als auch numerisch (double) gerechnet werden. Im konkreten Fall wurden die im Internet angegebenen Ziffern benutzt und symbolisch mit dieser Genauigkeit gerechnet. Die numerische Rechnung wurde als function implementiert und die symbolische als scrip-file.
Es hat sich dabei herausgestellt, dass c in einem relativ weiten Bereich gewählt werden kann. Genommen wurde das c, bei dem die Testrechnungen am besten abschnitten. Meist funktionierte die Fehlerschätzung besser, wenn c möglichst weit weg von b_2 gewählt wurde. Manchmal war auch c=-b_2 (z.B. Rkho12(10)dk) möglich.

6. Tests

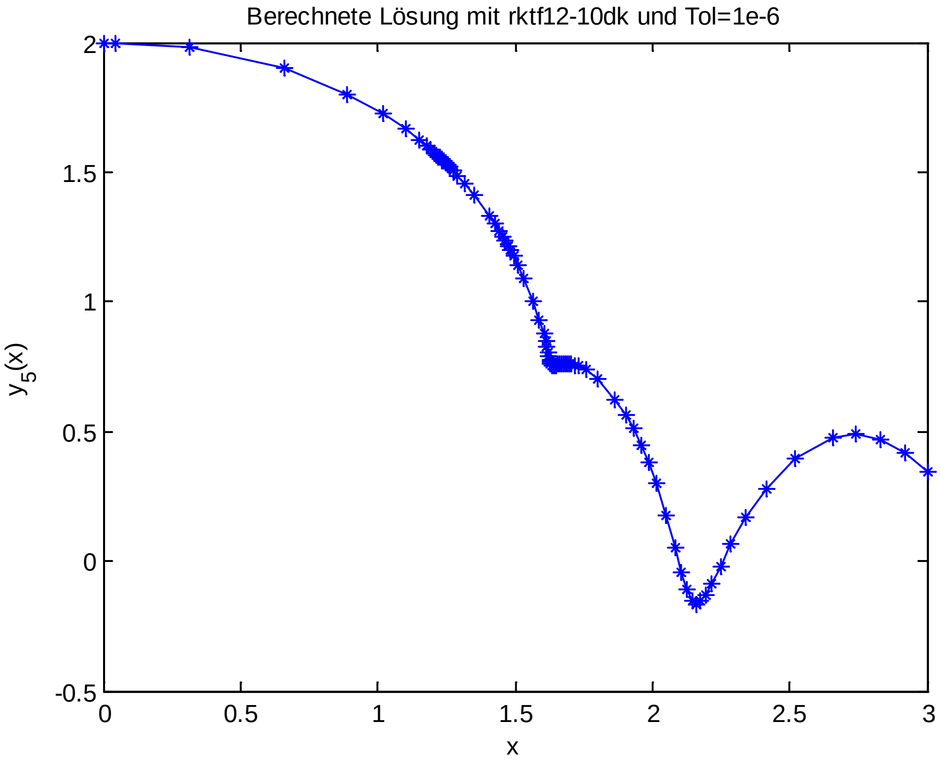
Zum Testen wurde ein Matlabprogramm geschrieben, das folgenden Funktions-Kopf hat:

```
function [tout, yout] = rungekuttaaut(F, ab, y, tol, RK)
%
% [tout, yout] = rungekutta(F, ab, y0, k, RK)
%
% INPUT:
% F     - Function containing the ode
%         Call: yout = F(x,y)
%         x       - scalar x
%         y       - Solution column-vector.
%         yout    - Returned derivative column-vector; yout(i) = dy(i)/dx
%
% ab    - Initial value of a=ab(1), b=ab(2)
% y     - Initial value column-vector
% tol   - Tolerance
% RK    - user-supplied RK-diagram
%         [a,b,b2,c,ord]=RK()
%         a              - Matrix A
%         b              - vector b for order p
%         b2             - vector b for order q
%         c              - vector c
%         ord            - order q
% OUTPUT:
% tout  - Returned integration time points (row-vector)
% yout  - Returned solution, one solution row-vector per tout-value
%
```

Dieses Programm kann mit allen angegeben RKV und auch mit den üblichen eingebetteten Verfahren arbeiten. Dabei muss die Routine RK das Butcher-Schema bzw. die Vektoren c und b, b2 und die Matrix A sowie die Ordnung q zu Verfügung stellen. Es ist die angegebene Schätzung des lokalen Fehlers und Steuerung von h umgesetzt. Die Testprobleme findet man als Matlabfiles unter [4].

Testproblem 1: Pleiades [5,4]

Dies sind 14 Gleichungen 2. Ordnung, die in 28 Gleichungen 1. Ordnung umgeschrieben wurden. Es ist ein beliebtes Bsp. insbesondere weil zur Lösung eine automatische Schrittweitensteuerung benötigt wird. Die Steuerung der Schrittweite zeigt die nachfolgende Grafik. Dabei steht das Zeichen * jeweils für einen Integrationspunkt y_i.



Es zeigt sich, dass die Integrationsschrittweite h stark schwankt. Bei dieser Rechnung beträgt ihr Minimum 5.9401e-004 und ihr Maximum 0.3465, d.h. das Verhältnis beträgt etwa 1:583. Die durchschnittliche Schrittweite beträgt 0.0297. Um die geforderte Genauigkeit bei konstanter Schrittweite zu erreichen, müsste man also mit h= 5.9401e-004 rechnen. Anders ausgedrückt, man würde 5050 Integrationsschritte an Stelle von hier 101 machen!

Ergebnisse:

a) Tol=1e-6

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	$\|y(3)-\tilde{y}(3)\|_\infty$
ode45	1	301	1825	3.8525e-005
rktf14_12dk	0.89	128	6126	1.8497e-007
rktf12_10dk	0.47	102	3426	4.2848e-007
rktf12_9	0.89	162	6352	1.0562e-008
rkho12_10dk	0.50	107	3701	4.6342e-007
rkho12_9	0.79	149	5917	1.6254e-009
rkeh10_8dk	0.57	195	4404	1.1814e-008
rkeh10_8	0.64	180	4881	3.6806e-008
rktf10_8dk	0.53	178	4030	7.4477e-007
rktf10_8	0.66	185	5061	1.1826e-007
rkho10_8dk	0.41	140	3197	3.1711e-007
rkho10_8	0.52	149	3981	8.8153e-008

Tol=1e-9

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	$\lVert y(3) - \tilde{y}(3) \rVert_\infty$
ode45	0.77	1387	8317	2.4269e-009
rktf14_12dk	0.55	218	10151	2.3114e-011
rktf12_10dk	0.31	194	6301	3.7210e-010
rktf12_9	0.63	353	12761	4.2570e-012
rkho12_10dk	0.33	199	6626	3.6650e-010
rkho12_9	0.57	320	11688	9.1172e-013
rkeh10_8dk	0.44	459	9062	3.8471e-012
rkeh10_8	0.53	431	11101	9.8197e-012
rktf10_8dk	0.42	426	8586	3.9164e-011
rktf10_8	1	484	20741	3.8909e-011
rkho10_8dk	0.35	331	7039	8.3591e-012
rkho10_8	0.42	349	8681	5.3041e-011

b) Tol=1e-11

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	$\lVert y(3) - \tilde{y}(3) \rVert_\infty$
ode45	0.025	3824	22939	3.4065e-011
rktf14_12dk	0.018	318	24641	2.4341e-012
rktf12_10dk	0.012	313	16676	2.1139e-012
rktf12_9	0.066	851	95672	9.9210e-013
rkho12_10dk	0.012	318	16676	3.2454e-012
rkho12_9	0.084	906	121656	7.9048e-012
rkeh10_8dk	0.022	877	31808	4.0741e-012
rkeh10_8	0.21	2287	289541	4.9623e-012
rktf10_8dk	0.019	788	27337	2.2735e-012
rktf10_8	1	8728	1447721	5.8409e-011
rkho10_8dk	0.015	603	20758	1.3967e-012
rkho10_8	0.096	1270	138781	1.6441e-011

Testproblem 2: Problemklasse D aus [3]

$$y_1' = y_3,$$
$$y_2' = y_4,$$
$$y_3' = -y_1 / (y_1^2 + y_2^2)^{3/2},$$
$$y_4' = -y_2 / (y_1^2 + y_2^2)^{3/2},$$

und

$$y_1(0) = 1 - \varepsilon,$$
$$y_2(0) = 0,$$
$$y_3(0) = 0,$$
$$y_4(0) = \sqrt{(1+\varepsilon)/(1-\varepsilon)}$$

Ergebnisse für $\varepsilon = .1$

c) Tol=1e-6

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_\infty$
ode45	1	46	271	1.7917e-006
rktf14_12dk	0.20	18	596	3.7282e-007
rktf12_10dk	0.088	13	351	1.0546e-006
rktf12_9	0.11	17	494	2.4131e-008
rkho12_10dk	0.065	14	326	3.8801e-007
rkho12_9	0.083	16	436	1.1208e-007
rkeh10_8dk	0.058	26	426	1.9202e-008
rkeh10_8	0.065	22	421	1.8117e-007
rktf10_8dk	0.048	18	307	5.1168e-008
rktf10_8	0.058	18	381	4.8645e-008
rkho10_8dk	0.042	16	273	3.4470e-007
rkho10_8	0.052	16	321	7.3028e-007

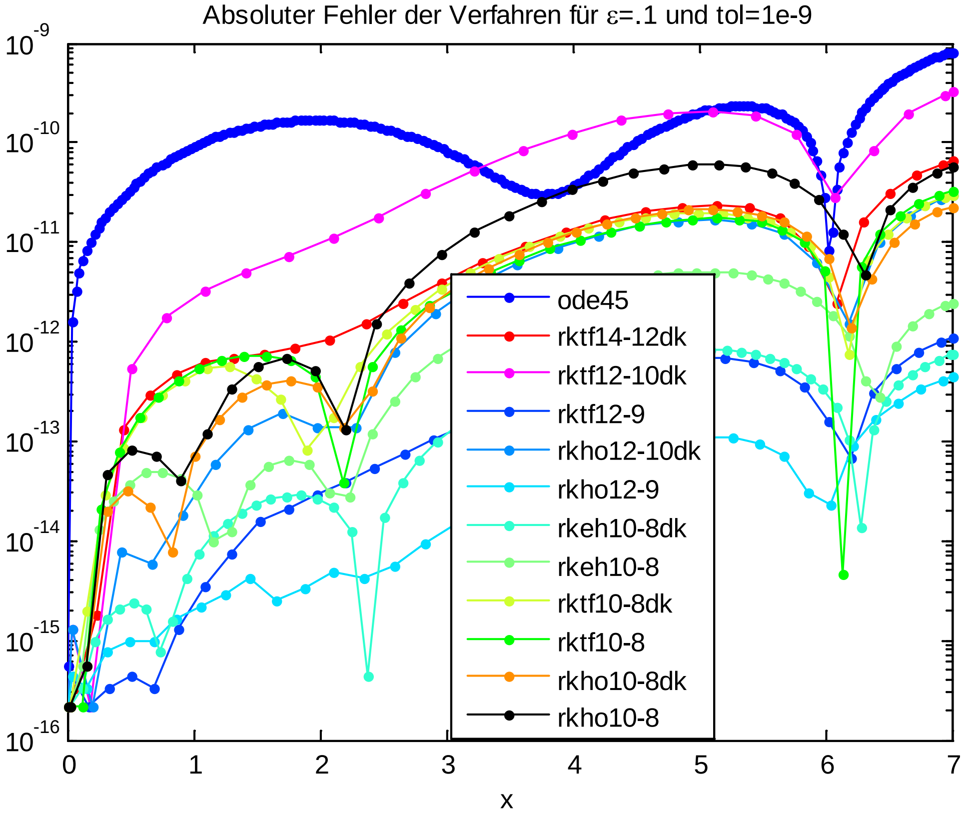
d) Tol=1e-9

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_\infty$
ode45	1	202	1207	9.9837e-010
rktf14_12dk	0.21	30	1016	7.5989e-011
rktf12_10dk	0.098	24	576	4.0505e-010
rktf12_9	0.15	36	1016	1.3930e-012
rkho12_10dk	0.091	28	676	3.6946e-011
rkho12_9	0.13	36	1016	4.8672e-013
rkeh10_8dk	0.11	62	1038	1.2173e-012
rkeh10_8	0.11	50	981	6.3441e-012
rktf10_8dk	0.075	40	664	3.8916e-011
rktf10_8	0.10	40	781	4.2069e-011
rkho10_8dk	0.077	38	630	3.4551e-011
rkho10_8	0.081	35	681	9.0964e-011

e) Tol=1e-11

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_\infty$
ode45	1	551	3301	7.4851e-012
rktf14_12dk	0.21	43	1576	4.9821e-013
rktf12_10dk	0.11	36	876	1.4791e-012
rktf12_9	0.17	61	1741	3.3862e-015
rkho12_10dk	0.11	44	1151	3.8761e-014
rkho12_9	0.17	62	1770	9.5729e-014
rkeh10_8dk	0.17	114	1922	5.6261e-014
rkeh10_8	0.16	87	1721	5.3264e-014
rktf10_8dk	0.11	73	1225	6.8917e-014
rktf10_8	0.13	73	1441	8.9651e-014
rkho10_8dk	0.10	70	1174	7.7979e-014
rkho10_8	0.11	64	1261	1.8900e-013

Für diese Rechnung mit tol=1e-9 hier mal eine grafische Ausgabe des Fehlers der Verfahren:



Ergebnisse für $\varepsilon = .9$:

f) Tol=1e-6

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_{\infty}$
ode45	1	113	685	0.0015508
rktf14_12dk	0.36	47	2031	2.4795e-006
rktf12_10dk	0.17	38	1176	0.00035095
rktf12_9	0.26	54	1915	1.8168e-007
rkho12_10dk	0.16	39	1226	6.1702e-007
rkho12_9	0.22	50	1828	1.2344e-007
rkeh10_8dk	0.15	63	1327	2.2209e-007
rkeh10_8	0.17	59	1481	3.4255e-007
rktf10_8dk	0.15	58	1242	3.3355e-006
rktf10_8	0.19	59	1541	5.9304e-006
rkho10_8dk	0.12	46	970	1.2999e-006
rkho10_8	0.15	50	1261	9.6718e-007

Tol=1e-9

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_\infty$
ode45	1	501	3001	3.1454e-007
rktf14_12dk	0.35	77	3361	4.2357e-010
rktf12_10dk	0.20	64	2076	4.6095e-009
rktf12_9	0.34	110	3858	2.7292e-011
rkho12_10dk	0.18	68	2101	8.6583e-010
rkho12_9	0.31	100	3597	2.3716e-011
rkeh10_8dk	0.23	139	2789	2.0726e-010
rkeh10_8	0.25	121	2961	1.1112e-009
rktf10_8dk	0.23	130	2704	1.9393e-009
rktf10_8	0.25	124	3021	7.7367e-009
rkho10_8dk	0.18	100	2109	6.6880e-010
rkho10_8	0.20	102	2441	5.8256e-010

g) Tol=1e-11

Name	Relative Zeit	Anzahl Punkte	Anz. DGL-Aufr.	Fehler $\|y-\tilde{y}\|_\infty$
ode45	1	1373	8233	1.2485e-009
rktf14_12dk	0.32	108	4761	3.2401e-011
rktf12_10dk	0.20	100	3301	3.4394e-011
rktf12_9	0.38	185	6613	5.3071e-011
rkho12_10dk	0.18	104	3251	2.1057e-011
rkho12_9	0.33	167	5888	6.2217e-011
rkeh10_8dk	0.25	246	4540	3.5182e-011
rkeh10_8	0.27	215	4921	5.8726e-011
rktf10_8dk	0.23	233	4030	7.4994e-012
rktf10_8	0.28	224	5121	8.6207e-011
rkho10_8dk	0.20	177	3588	6.9047e-012
rkho10_8	0.24	182	4201	2.0802e-011

7. Zusammenfassung

Die Tests zeigen, dass die jeweiligen ‚dk'-Versionen der RKV dem entsprechenden ‚aufgebetteten' Verfahren aus [6] und auch ode45 meist überlegen sind. Dies zeigen auch noch weitere hier nicht veröffentlichte Testrechnungen. Die Aussage gilt sowohl für den numerischen Aufwand und der damit verbundener Zeitersparnis, als auch für die Einhaltung der geforderten Rechengenauigkeit. Überraschenderweise hat sie das RKV mit der Ordnung 14 nicht als das Beste herausgestellt. Dies kann seine Ursache darin haben, dass die Konstante c nicht optimal gewählt wurde. Wenn man sich das Gebiet der A-Stabilität ansieht, fällt auf, dass dieses Verfahren einen relativ kleinen Bereich hat. Dies könnte eine weitere Ursache dafür sein.

Literaturverzeichnis:

[1] M. Fritsche: Ein Programm zur automatischen Erzeugung der Konsistenzgleichungen bei Runge-Kutta-Verfahren, *Reports on Numerical Mathematics*
http://www.minet.uni-jena.de/Math-Net/reports/sources/2004/04-02report.ps

[2] E. Hairer, A Runge-Kutta Method of Order 10, J. Inst. Maths Applics (1978) 21, 47-59.

[3] T. E. Hull, W. H. Enright, B. M. Fellen and A. E. Sedgwick, SIAM Journal on Numerical Analysis, Vol. 9, No. 4 (Dec., 1972), pp. 603-637

[4] http://users.minet.uni-jena.de/~kaiserd/rungekutta

[5] http://www.dm.uniba.it/~testset/report/plei.pdf

[6] **Unter:** ‚Higher order Runge-Kutta schemes' von:
http://www.peterstone.name/Maplepgs/RKcoeff.html

[7] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10b_1.pdf

[8] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10b(8)_1.pdf

[9] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10e_1.pdf

[10] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10e(8)_1.pdf

[11] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10f_1.pdf

[12] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK10/RKcoeff10f(8)_1.pdf

[13] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK12/RKcoeff12a_1.pdf

[14] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK12/RKcoeff12a(9)_1.pdf

[15] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK12/RKcoeff12e_1.pdf

[16] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK12/RKcoeff12e_1.pdf

[17] http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK14/RKcoeff14a_1.pdf