《数值分析》之

常微分方程数值方法

徐岩

中国科学技术大学数学系

yxu@ustc.edu.cn

http://staff.ustc.edu.cn/~yxu/

Taylor级数方法的回顾

- 对于ODE数值解的Taylor级数方法，为了进行程序设计，需要事先进行一些分析，确定求导的最高阶数
- 例如，若想对一般的ODE初值问题

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

应用四阶Taylor级数方法，那么就需要进行逐次微分，得到y'', y'''和$y^{(4)}$

- Runge-Kutta方法也是采用了Taylor级数展开，但是它通过对$f(x, y)$的恰当组合，避免了上述困难

二阶Runge-Kutta方法

- $y(x+h)$的Taylor级数展开：

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \frac{h^3}{3!}y'''(x) + \cdots$$

- 根据微分方程，我们得到

$$\begin{aligned} y'(x) &= f \\ y''(x) &= f_x + f_y y' = f_x + f_y f \\ y'''(x) &= f_{xx} + f_{xy} f + (f_x + f_y f) f_y + f(f_{yx} + f_{yy} f) \\ &\vdots \end{aligned}$$

这里下标表示偏导数

- $y(x+h)$的Taylor展开到前三项，可以写为

$$\begin{aligned} y(x+h) &= y + hf + \frac{1}{2}h^2(f_x + ff_y) + \mathcal{O}(h^3) \\ &= y + \frac{1}{2}hf + \frac{1}{2}h[f + hf_x + hff_y] + \mathcal{O}(h^3) \end{aligned}$$

- 应用两变量函数的Taylor展开，可以消去前几项中的偏导数。事实上，

$$f(x+h, y+hf) = f + hf_x + hff_y + \mathcal{O}(h^2)$$

从而有

$$y(x+h) = y + \frac{1}{2}hf + \frac{1}{2}hf(x+h, y+hf) + \mathcal{O}(h^3)$$

Heun公式

- 从而步进求解的公式可以写为

$$y(x+h) = y(x) + \frac{h}{2}f(x,y) + \frac{h}{2}f(x+h, y+hf(x,y))$$

或等价地写为

$$y(x+h) = y(x) + \frac{1}{2}(F_1 + F_2)$$

其中

$$F_1 = hf(x,y), \qquad F_2 = hf(x+h, y+F_1)$$

- 上述公式称为二阶Runge-Kutta方法,也称Heun公式

二阶Runge-Kutta方法的一般化

- 一般来说，二阶Runge-Kutta公式具有形式

$$y(x+h) = y + w_1 hf + w_2 hf(x+\alpha h, y+\beta hf) + \mathcal{O}(h^3)$$

其中w_1, w_2, α和β是需要配置的参数

- 借助于两变量函数的Taylor展开，上式可改写为

$$y(x+h) = y + w_1 hf + w_2 h[f + \alpha h f_x + \beta h f f_y] + \mathcal{O}(h^3)$$

因此结合$y(x+h)$真正的Taylor展开式

$$y(x+h) = y + hf + \frac{1}{2}h^2(f_x + f f_y) + \mathcal{O}(h^3)$$

可得对参数强加的条件：

$$w_1 + w_2 = 1, w_2\alpha = w_2\beta = \frac{1}{2} \Longrightarrow \alpha = \beta = \frac{1}{2w_2} = \frac{1}{2(1-w_1)}$$

各种版本的二阶Runge-Kutta方法

- $w_1 = w_2 = 1/2$, $\alpha = \beta = 1$对应的就是Heun方法
- $w_1 = 0$, $w_2 = 1$, $\alpha = \beta = 1/2$对应的就是修正的Euler方法：

$$y(x+h) = y(x) + F_2$$

其中

$$F_1 = hf(x, y)$$
$$F_2 = hf\left(x + h/2, y + F_1/2\right)$$

- 二阶以及其它阶的Runge-Kutta方法都具有一个显著特点，那就是只需要计算$f(x, y)$在一些点的值，而不需要计算f_x, f_y等导数的表达式以及取值

四阶Runge-Kutta方法

- 推导高阶Runge-Kutta方法的过程是相当令人乏味的，但是推导出来的结果一般是相当简明的
- 经典的四阶Runge-Kutta方法是

$$y(x+h) = y(x) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)$$

其中

$$\begin{aligned} F_1 &= hf(x, y) \\ F_2 &= hf\left(x + h/2, y + F_1/2\right) \\ F_3 &= hf\left(x + h/2, y + F_2/2\right) \\ F_4 &= hf\left(x + h, y + F_3\right) \end{aligned}$$

- 这个方法被称为四阶方法，是因为它应用了Taylor展开中直到四阶的所有项。误差为$\mathcal{O}(h^5)$

下面给出一个例子，用以验证方法的有效性

- 在区间$[1,3]$上，采用步长$h=1/128$的四阶Runge-Kutta方法求解下列初值问题：

$$\begin{cases} y' = \frac{xy-y^2}{x^2} \\ y(1) = 2 \end{cases}$$

- 其精确解为

$$y(x) = \frac{x}{\ln x + 1/2}$$

- 结果“ode_runge_kutta_4.pdf”

```
M = 256; k = 0; h = 1.0 / 128; t = 1.0; x = 2.0; sol = Table[{t, x}, {n, 0, M}];
f[tt_, xx_] = (tt - xx) * xx / tt^2; u[a_] = 2 a / (1 + 2 Log[a]);
e = Abs[u[t] - x]; Print[k, "\t", t, "\t", x, "\t", e];
For[k = 1, k <= M, k++, F1 = h * f[t, x]; F2 = h * f[t + h / 2, x + F1 / 2]; F3 = h * f[t + h / 2, x + F2 / 2];
 F4 = h * f[t + h, x + F3]; x0 = x; x = x + (F1 + 2 * F2 + 2 * F3 + F4) / 6; t = t + h; e = Abs[u[t] - x];
 Print[k, "\t", t, "\t", x, "\t", e]; sol[[k]] = {t, x}]
```

0	1.	2.	0.
1	1.00781	1.98473	2.59039×10^{-11}
2	1.01563	1.97016	4.87406×10^{-11}
3	1.02344	1.95623	6.88727×10^{-11}
4	1.03125	1.94293	8.66187×10^{-11}
5	1.03906	1.9302	1.02256×10^{-10}
6	1.04688	1.91802	1.1603×10^{-10}
7	1.05469	1.90637	1.28154×10^{-10}
8	1.0625	1.89521	1.38818×10^{-10}
9	1.07031	1.88452	1.48185×10^{-10}
10	1.07813	1.87427	1.56404×10^{-10}
11	1.08594	1.86445	1.63602×10^{-10}
12	1.09375	1.85503	1.69893×10^{-10}
13	1.10156	1.846	1.75379×10^{-10}
14	1.10938	1.83733	1.80147×10^{-10}
15	1.11719	1.82901	1.84276×10^{-10}
16	1.125	1.82103	1.87837×10^{-10}
17	1.13281	1.81336	1.90892×10^{-10}
18	1.14063	1.806	1.93495×10^{-10}
19	1.14844	1.79892	1.95695×10^{-10}
20	1.15625	1.79213	1.97536×10^{-10}
21	1.16406	1.7856	1.99057×10^{-10}
22	1.17188	1.77933	2.00292×10^{-10}
23	1.17969	1.7733	2.01273×10^{-10}
24	1.1875	1.76751	2.02026×10^{-10}
25	1.19531	1.76194	2.02578×10^{-10}
26	1.20313	1.75659	2.02948×10^{-10}
27	1.21094	1.75144	2.03158×10^{-10}
28	1.21875	1.7465	2.03225×10^{-10}
29	1.22656	1.74174	2.03164×10^{-10}
30	1.23438	1.73717	2.02991×10^{-10}
31	1.24219	1.73278	2.02717×10^{-10}

32	1.25	1.72856	2.02355×10^{-10}
33	1.25781	1.72451	2.01914×10^{-10}
34	1.26563	1.72061	2.01403×10^{-10}
35	1.27344	1.71687	2.00832×10^{-10}
36	1.28125	1.71328	2.00207×10^{-10}
37	1.28906	1.70982	1.99534×10^{-10}
38	1.29688	1.70651	1.98821×10^{-10}
39	1.30469	1.70333	1.98072×10^{-10}
40	1.3125	1.70028	1.97293×10^{-10}
41	1.32031	1.69735	1.96487×10^{-10}
42	1.32813	1.69454	1.95659×10^{-10}
43	1.33594	1.69185	1.94812×10^{-10}
44	1.34375	1.68927	1.93949×10^{-10}
45	1.35156	1.68679	1.93074×10^{-10}
46	1.35938	1.68443	1.92188×10^{-10}
47	1.36719	1.68216	1.91295×10^{-10}
48	1.375	1.68	1.90395×10^{-10}
49	1.38281	1.67793	1.89492×10^{-10}
50	1.39063	1.67595	1.88586×10^{-10}
51	1.39844	1.67406	1.8768×10^{-10}
52	1.40625	1.67226	1.86774×10^{-10}
53	1.41406	1.67055	1.8587×10^{-10}
54	1.42188	1.66891	1.84968×10^{-10}
55	1.42969	1.66736	1.84071×10^{-10}
56	1.4375	1.66588	1.83177×10^{-10}
57	1.44531	1.66448	1.8229×10^{-10}
58	1.45313	1.66315	1.81408×10^{-10}
59	1.46094	1.6619	1.80532×10^{-10}
60	1.46875	1.66071	1.79664×10^{-10}
61	1.47656	1.65959	1.78803×10^{-10}
62	1.48438	1.65853	1.77951×10^{-10}
63	1.49219	1.65754	1.77106×10^{-10}
64	1.5	1.65661	1.7627×10^{-10}
65	1.50781	1.65574	1.75443×10^{-10}
66	1.51563	1.65492	1.74625×10^{-10}
67	1.52344	1.65417	1.73816×10^{-10}
68	1.53125	1.65347	1.73016×10^{-10}

69	1.53906	1.65282	1.72226×10^{-10}
70	1.54688	1.65223	1.71445×10^{-10}
71	1.55469	1.65168	1.70674×10^{-10}
72	1.5625	1.65119	1.69912×10^{-10}
73	1.57031	1.65075	1.6916×10^{-10}
74	1.57813	1.65035	1.68418×10^{-10}
75	1.58594	1.65	1.67685×10^{-10}
76	1.59375	1.64969	1.66962×10^{-10}
77	1.60156	1.64943	1.66249×10^{-10}
78	1.60938	1.64921	1.65545×10^{-10}
79	1.61719	1.64903	1.64851×10^{-10}
80	1.625	1.6489	1.64166×10^{-10}
81	1.63281	1.6488	1.6349×10^{-10}
82	1.64063	1.64874	1.62824×10^{-10}
83	1.64844	1.64872	1.62167×10^{-10}
84	1.65625	1.64874	1.61519×10^{-10}
85	1.66406	1.64879	1.6088×10^{-10}
86	1.67188	1.64888	1.6025×10^{-10}
87	1.67969	1.649	1.59629×10^{-10}
88	1.6875	1.64916	1.59017×10^{-10}
89	1.69531	1.64935	1.58413×10^{-10}
90	1.70313	1.64957	1.57818×10^{-10}
91	1.71094	1.64983	1.57231×10^{-10}
92	1.71875	1.65011	1.56652×10^{-10}
93	1.72656	1.65042	1.56082×10^{-10}
94	1.73438	1.65077	1.55519×10^{-10}
95	1.74219	1.65114	1.54965×10^{-10}
96	1.75	1.65154	1.54419×10^{-10}
97	1.75781	1.65197	1.5388×10^{-10}
98	1.76563	1.65243	1.53348×10^{-10}
99	1.77344	1.65291	1.52825×10^{-10}
100	1.78125	1.65342	1.52308×10^{-10}
101	1.78906	1.65395	1.51799×10^{-10}
102	1.79688	1.65451	1.51298×10^{-10}
103	1.80469	1.65509	1.50803×10^{-10}
104	1.8125	1.65569	1.50316×10^{-10}
105	1.82031	1.65632	1.49835×10^{-10}

106	1.82813	1.65697	1.4936×10^{-10}
107	1.83594	1.65765	1.48893×10^{-10}
108	1.84375	1.65834	1.48432×10^{-10}
109	1.85156	1.65906	1.47977×10^{-10}
110	1.85938	1.6598	1.47529×10^{-10}
111	1.86719	1.66056	1.47087×10^{-10}
112	1.875	1.66134	1.46652×10^{-10}
113	1.88281	1.66214	1.46222×10^{-10}
114	1.89063	1.66295	1.45798×10^{-10}
115	1.89844	1.66379	1.4538×10^{-10}
116	1.90625	1.66465	1.44968×10^{-10}
117	1.91406	1.66552	1.44561×10^{-10}
118	1.92188	1.66641	1.4416×10^{-10}
119	1.92969	1.66732	1.43765×10^{-10}
120	1.9375	1.66825	1.43375×10^{-10}
121	1.94531	1.66919	1.4299×10^{-10}
122	1.95313	1.67015	1.42611×10^{-10}
123	1.96094	1.67113	1.42236×10^{-10}
124	1.96875	1.67212	1.41867×10^{-10}
125	1.97656	1.67313	1.41502×10^{-10}
126	1.98438	1.67415	1.41143×10^{-10}
127	1.99219	1.67519	1.40788×10^{-10}
128	2.	1.67624	1.40438×10^{-10}
129	2.00781	1.67731	1.40093×10^{-10}
130	2.01563	1.67839	1.39752×10^{-10}
131	2.02344	1.67948	1.39416×10^{-10}
132	2.03125	1.68059	1.39084×10^{-10}
133	2.03906	1.68171	1.38757×10^{-10}
134	2.04688	1.68285	1.38434×10^{-10}
135	2.05469	1.684	1.38115×10^{-10}
136	2.0625	1.68516	1.378×10^{-10}
137	2.07031	1.68633	1.3749×10^{-10}
138	2.07813	1.68752	1.37184×10^{-10}
139	2.08594	1.68872	1.36881×10^{-10}
140	2.09375	1.68993	1.36583×10^{-10}
141	2.10156	1.69115	1.36288×10^{-10}
142	2.10938	1.69239	1.35997×10^{-10}

143	2.11719	1.69363	1.3571×10^{-10}
144	2.125	1.69489	1.35427×10^{-10}
145	2.13281	1.69615	1.35147×10^{-10}
146	2.14063	1.69743	1.3487×10^{-10}
147	2.14844	1.69872	1.34598×10^{-10}
148	2.15625	1.70002	1.34329×10^{-10}
149	2.16406	1.70132	1.34063×10^{-10}
150	2.17188	1.70264	1.338×10^{-10}
151	2.17969	1.70397	1.33541×10^{-10}
152	2.1875	1.70531	1.33285×10^{-10}
153	2.19531	1.70666	1.33032×10^{-10}
154	2.20313	1.70801	1.32783×10^{-10}
155	2.21094	1.70938	1.32537×10^{-10}
156	2.21875	1.71075	1.32293×10^{-10}
157	2.22656	1.71214	1.32053×10^{-10}
158	2.23438	1.71353	1.31815×10^{-10}
159	2.24219	1.71493	1.31581×10^{-10}
160	2.25	1.71634	1.3135×10^{-10}
161	2.25781	1.71776	1.31121×10^{-10}
162	2.26563	1.71918	1.30895×10^{-10}
163	2.27344	1.72062	1.30672×10^{-10}
164	2.28125	1.72206	1.30452×10^{-10}
165	2.28906	1.72351	1.30234×10^{-10}
166	2.29688	1.72496	1.30019×10^{-10}
167	2.30469	1.72643	1.29807×10^{-10}
168	2.3125	1.7279	1.29597×10^{-10}
169	2.32031	1.72938	1.2939×10^{-10}
170	2.32813	1.73087	1.29185×10^{-10}
171	2.33594	1.73236	1.28983×10^{-10}
172	2.34375	1.73386	1.28783×10^{-10}
173	2.35156	1.73537	1.28585×10^{-10}
174	2.35938	1.73688	1.2839×10^{-10}
175	2.36719	1.7384	1.28197×10^{-10}
176	2.375	1.73993	1.28007×10^{-10}
177	2.38281	1.74146	1.27819×10^{-10}
178	2.39063	1.743	1.27633×10^{-10}
179	2.39844	1.74455	1.27449×10^{-10}

180	2.40625	1.7461	1.27268×10^{-10}
181	2.41406	1.74766	1.27088×10^{-10}
182	2.42188	1.74922	1.2691×10^{-10}
183	2.42969	1.75079	1.26735×10^{-10}
184	2.4375	1.75237	1.26562×10^{-10}
185	2.44531	1.75395	1.26391×10^{-10}
186	2.45313	1.75554	1.26222×10^{-10}
187	2.46094	1.75713	1.26055×10^{-10}
188	2.46875	1.75873	1.2589×10^{-10}
189	2.47656	1.76033	1.25727×10^{-10}
190	2.48438	1.76194	1.25565×10^{-10}
191	2.49219	1.76356	1.25405×10^{-10}
192	2.5	1.76517	1.25248×10^{-10}
193	2.50781	1.7668	1.25092×10^{-10}
194	2.51563	1.76843	1.24938×10^{-10}
195	2.52344	1.77006	1.24786×10^{-10}
196	2.53125	1.7717	1.24636×10^{-10}
197	2.53906	1.77334	1.24487×10^{-10}
198	2.54688	1.77499	1.2434×10^{-10}
199	2.55469	1.77664	1.24195×10^{-10}
200	2.5625	1.7783	1.24051×10^{-10}
201	2.57031	1.77996	1.23909×10^{-10}
202	2.57813	1.78163	1.23768×10^{-10}
203	2.58594	1.7833	1.2363×10^{-10}
204	2.59375	1.78497	1.23493×10^{-10}
205	2.60156	1.78665	1.23357×10^{-10}
206	2.60938	1.78833	1.23223×10^{-10}
207	2.61719	1.79002	1.2309×10^{-10}
208	2.625	1.79171	1.22959×10^{-10}
209	2.63281	1.7934	1.22829×10^{-10}
210	2.64063	1.7951	1.22701×10^{-10}
211	2.64844	1.79681	1.22574×10^{-10}
212	2.65625	1.79851	1.22449×10^{-10}
213	2.66406	1.80022	1.22325×10^{-10}
214	2.67188	1.80194	1.22203×10^{-10}
215	2.67969	1.80365	1.22082×10^{-10}
216	2.6875	1.80537	1.21962×10^{-10}

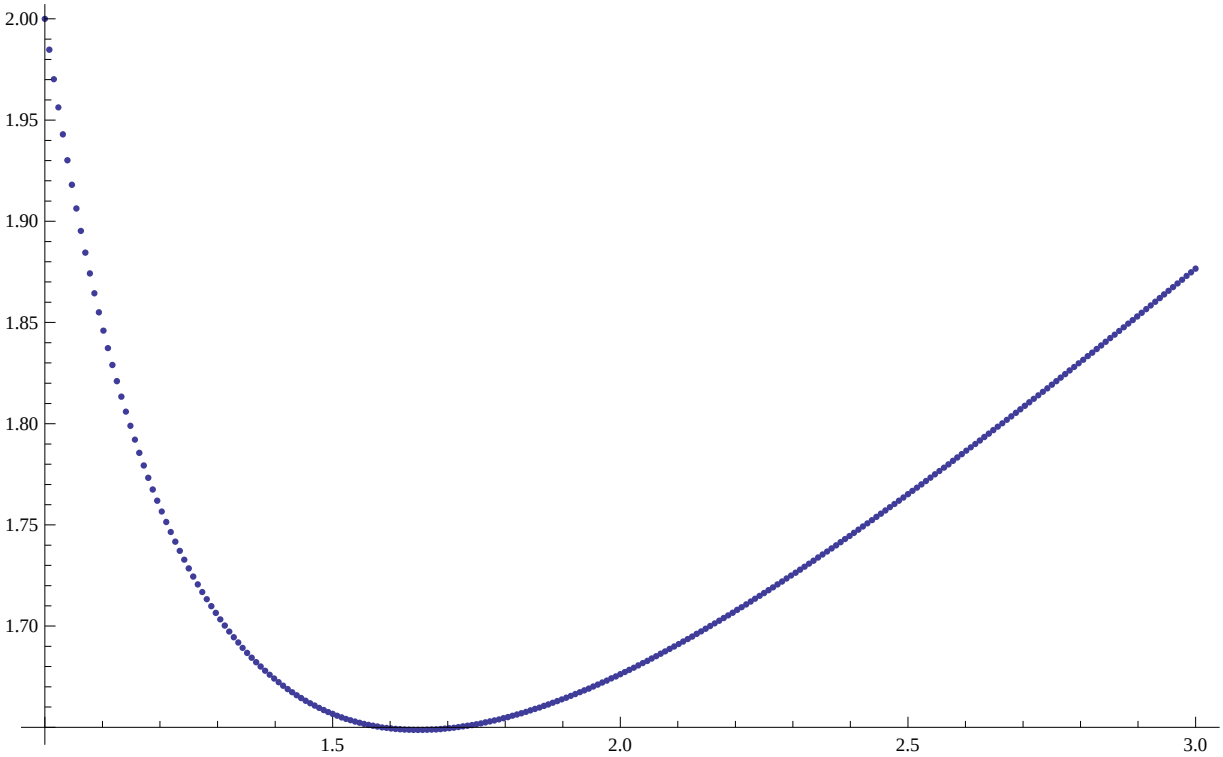
217	2.69531	1.8071	1.21844×10^{-10}
218	2.70313	1.80883	1.21727×10^{-10}
219	2.71094	1.81056	1.21612×10^{-10}
220	2.71875	1.81229	1.21497×10^{-10}
221	2.72656	1.81403	1.21384×10^{-10}
222	2.73438	1.81577	1.21272×10^{-10}
223	2.74219	1.81752	1.21161×10^{-10}
224	2.75	1.81926	1.21052×10^{-10}
225	2.75781	1.82101	1.20944×10^{-10}
226	2.76563	1.82277	1.20837×10^{-10}
227	2.77344	1.82452	1.20731×10^{-10}
228	2.78125	1.82628	1.20627×10^{-10}
229	2.78906	1.82805	1.20523×10^{-10}
230	2.79688	1.82981	1.20421×10^{-10}
231	2.80469	1.83158	1.2032×10^{-10}
232	2.8125	1.83335	1.2022×10^{-10}
233	2.82031	1.83513	1.20121×10^{-10}
234	2.82813	1.83691	1.20023×10^{-10}
235	2.83594	1.83869	1.19927×10^{-10}
236	2.84375	1.84047	1.19831×10^{-10}
237	2.85156	1.84225	1.19737×10^{-10}
238	2.85938	1.84404	1.19644×10^{-10}
239	2.86719	1.84583	1.19551×10^{-10}
240	2.875	1.84762	1.1946×10^{-10}
241	2.88281	1.84942	1.1937×10^{-10}
242	2.89063	1.85122	1.1928×10^{-10}
243	2.89844	1.85302	1.19192×10^{-10}
244	2.90625	1.85482	1.19105×10^{-10}
245	2.91406	1.85663	1.19018×10^{-10}
246	2.92188	1.85843	1.18933×10^{-10}
247	2.92969	1.86024	1.18848×10^{-10}
248	2.9375	1.86205	1.18765×10^{-10}
249	2.94531	1.86387	1.18682×10^{-10}
250	2.95313	1.86568	1.18601×10^{-10}
251	2.96094	1.8675	1.1852×10^{-10}
252	2.96875	1.86932	1.1844×10^{-10}
253	2.97656	1.87115	1.18361×10^{-10}

254 2.98438 1.87297 1.18283×10^{-10}

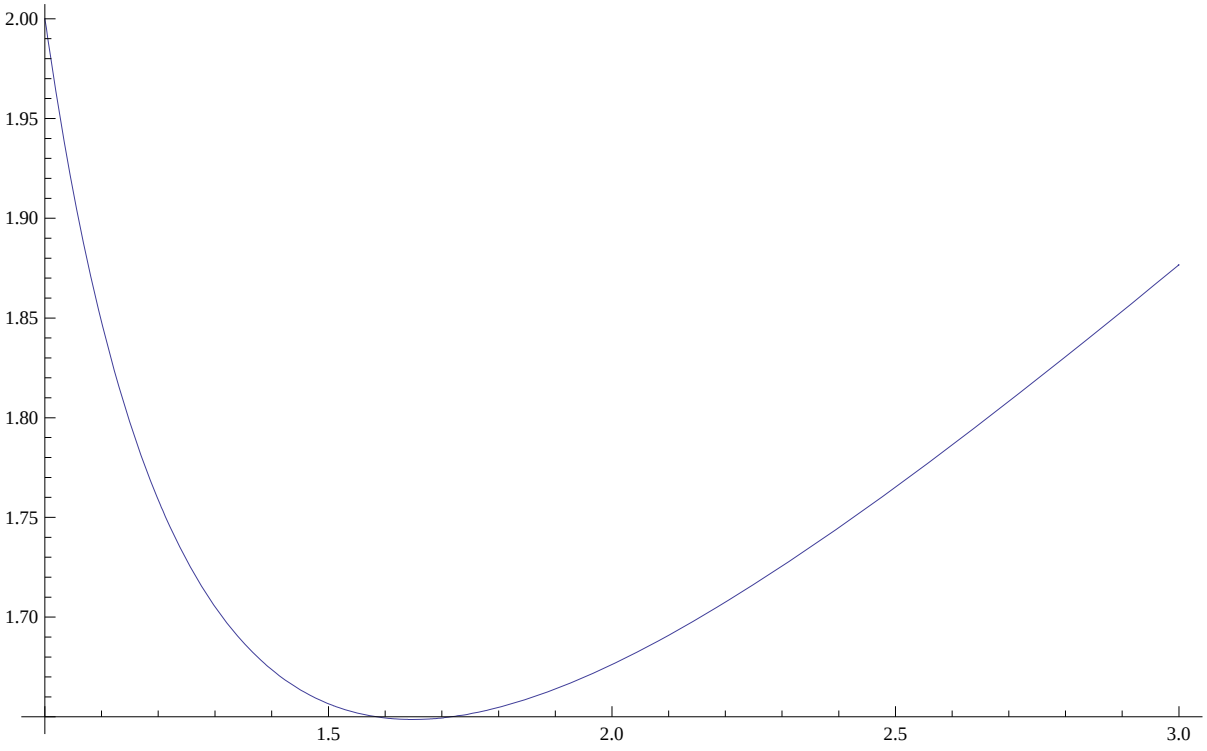
255 2.99219 1.8748 1.18206×10^{-10}

256 3. 1.87663 1.18129×10^{-10}

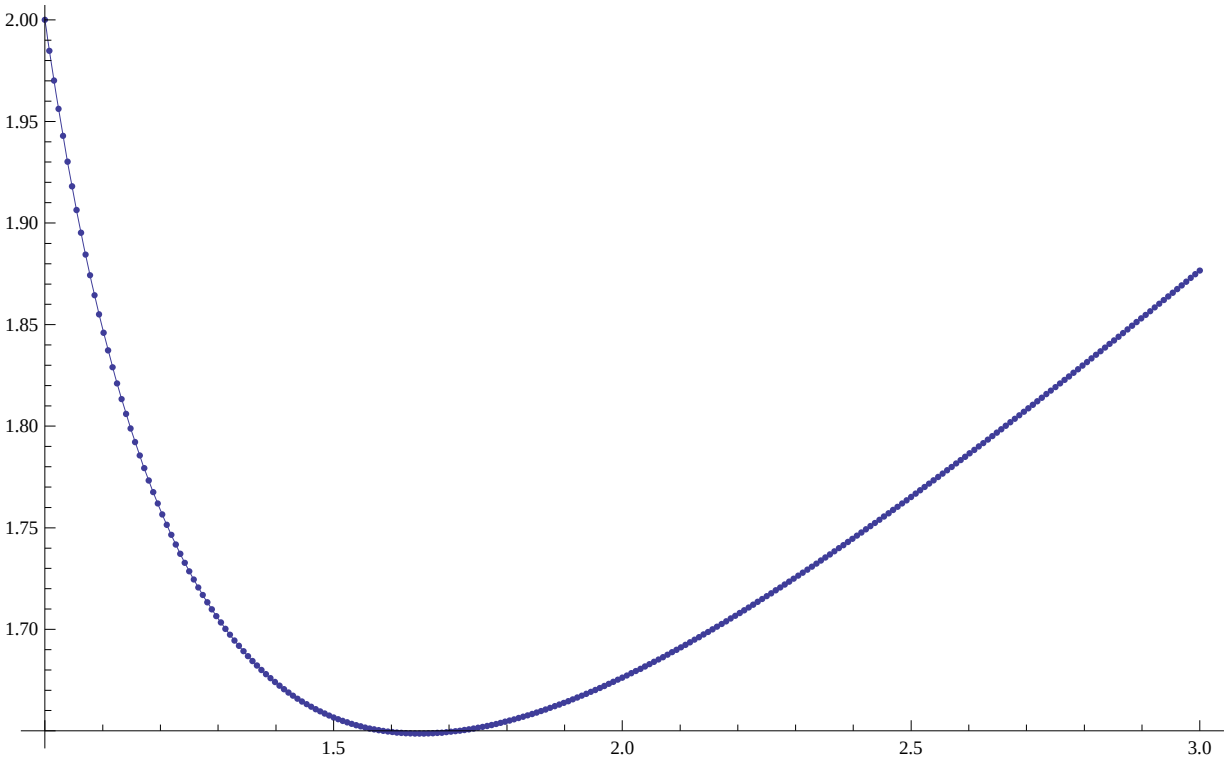
```
fig1 = ListPlot[sol]
```




```
fig2 = Plot[2 t / (1 + 2 * Log[t]), {t, 1, 3}]
```



```
Show[fig1, fig2]
```




```
DSolve[{xx'[tt] == tt^{-2} (tt * xx[tt] - xx[tt]^2), xx[1] == 2}, xx[tt], tt]
```

$$\left\{\left\{\text{xx}[\text{tt}] \to \frac{2\,\text{tt}}{1 + 2\,\text{Log}[\text{tt}]}\right\}\right\}$$

其它的低阶Runge-Kutta公式

- Runge-Kutta方法的构造设想比较清晰，但是其中系数所满足的非线性方程相当复杂，而求解这个非线性方程组更困难。通常这组方程的解并不唯一，因此可以有几种不同的同阶Runge-Kutta方法
- 高阶Runge-Kutta方法通常用于要求高精度的常微分方程数值求解
- 常用的三阶Runge-Kutta方法有三个：
  1. 第一个：

$$y(x+h) = y(x) + \frac{1}{6}(F_1 + 4F_2 + F_3)$$

其中

$$\begin{aligned} F_1 &= hf(x, y) \\ F_2 &= hf(x + h/2, y + F_1/2) \\ F_3 &= hf(x + h, y - F_1 + 2F_2) \end{aligned}$$

- (续)
  2. 第二个

$$y(x+h) = y(x) + \frac{1}{4}(F_1 + 3F_3)$$

其中

$$F_1 = hf(x, y)$$
$$F_2 = hf(x + h/3, y + F_1/3)$$
$$F_3 = hf(x + 2h/3, y + 2F_2/3)$$

  3. 第三个

$$y(x+h) = y(x) + \frac{1}{9}(2F_1 + 3F_2 + 4F_3)$$

其中

$$F_1 = hf(x, y)$$
$$F_2 = hf(x + h/2, y + F_1/2)$$
$$F_3 = hf(x + 3h/4, y + 3F_2/4)$$

其它的四阶Runge-Kutta公式

1. $$y(x+h) = y(x) + \frac{1}{8}(F_1 + 3F_2 + 3F_3 + F_4)$$

其中

$$\begin{aligned} F_1 &= hf(x, y) \\ F_2 &= hf(x + h/3, y + F_1/3) \\ F_3 &= hf(x + 2h/3, y + F_1/3 + F_2) \\ F_4 &= hf(x + h, y + F_1 - F_2 + F_3) \end{aligned}$$

2. 第二个: Runge-Kutta-Gill方法

$$y(x+h) = y(x) + \frac{1}{6}\Big(F_1 + (2-\sqrt{2})F_2 + (2+\sqrt{2})F_3 + F_4\Big)$$

其中

$$\begin{aligned}
F_1 &= hf(x,y)\\
F_2 &= hf(x+h/2, y+F_1/2)\\
F_3 &= hf\left(x+h/2, y+\frac{\sqrt{2}-1}{2}F_1 + (1-\sqrt{2}/2)F_2\right)\\
F_4 &= hf\left(x+h, y-\frac{\sqrt{2}}{2}F_2 - (1+\sqrt{2}/2)F_3\right)
\end{aligned}$$

误差

- 在Runge-Kutta方法的第一步，$y(x_0+h)$的值是由算法计算出来的。另一方面，存在一个我们不知道的正确解$y^*(x_0+h)$. 那么在该步中的局部截断误差是

$$y^*(x_0+h)-y(x_0+h)$$

- Runge-Kutta方法的理论指出这个截断误差对小的h值具有类似于Ch^{n+1}的性态，其中n是方法的阶，C是一个与h无关但与x_0和函数y^*有关的数

局部截断误差的估计

- 以四阶Runge-Kutta方法为例。为估计Ch^5, 我们假定当x从x_0变到x_0+h时C不变
- 设v是在x_0+h上的近似解的值，它是从x_0出发取长度为h的步长一步得到的；设u 是在x_0+h上的另一种数值近似解，它是从x_0出发取长度为$h/2$步长进行两步计算得到的。u, v都是可计算的(computable).
- 从而有

$$y^*(x_0+h) = v + Ch^5$$
$$y^*(x_0+h) = u + 2C(h/2)^5$$

两式相减，得到

$$\text{局部截断误差} = Ch^5 = \frac{u-v}{1-2^{-4}} \approx u - v$$

自适应策略

- 在Runge-Kutta方法的计算机实现中，为了保证近似的截断误差处于特定的容限之下，我们可以通过计算$|u-v|$的值来帮助判断
- 若$|u-v|$超出特定的容限，可以减小步长(通常减半)来改善局部截断误差
- 另一方面，若局部截断误差远远小于特定的容限，那么可以使步长加倍

高阶Runge-Kutta方法的不足

- 同阶的Runge-Kutta方法会有不同的版本，因此Runge-Kutta方法是一族方法
- 在各阶Runge-Kutta方法中，需要的函数求值数目相对于阶数的增加要变得更迅速，如下表所示

函数求值数	1	2	3	4	5	6	7	8
Runge-Kutta方法的阶	1	2	3	4	4	5	6	6

- 因此高阶Runge-Kutta方法并不比经典的四阶Runge-Kutta方法具有更大的吸引力

Runge-Kutta-Fehlberg方法

- 为了尝试在Runge-Kutta方法中设计一个自动调整步长的方法，Fehlberg仔细研究了一个具有五次函数求值的四阶方法和一个具有六次函数求值的五阶方法，通过巧妙选取方法的参数，得到了一个只需要六次函数求值的自适应Runge-Kutta方法
- 方法的基本思路就是让四阶方法的五次函数求值包含在五阶方法的六次函数求值中，从而把本来的11次函数求值减少为只有六次。同时应用两种方法得到结果的差做为对局部截断误差的估计

RKF方法中的六次函数求值

- 六次函数求值:

$$
\begin{aligned}
F_1 &= hf(x, y) \\
F_2 &= hf(x + h/4, y + F_1/4) \\
F_3 &= hf\left(x + \frac{3}{8}h, y + \frac{3}{32}F_1 + \frac{9}{32}F_2\right) \\
F_4 &= hf\left(x + \frac{12}{13}h, y + \frac{1932}{2197}F_1 - \frac{7200}{2197}F_2 + \frac{7296}{2197}F_3\right) \\
F_5 &= hf\left(x + h, y + \frac{439}{216}F_1 - 8F_2 + \frac{3680}{513}F_3 - \frac{845}{4104}F_4\right) \\
F_6 &= hf\left(x + \frac{1}{2}h, y - \frac{8}{27}F_1 + 2F_2 - \frac{3544}{2565}F_3 \right. \\
&\qquad \left. + \frac{1859}{4104}F_4 - \frac{11}{40}F_5\right)
\end{aligned}
$$

RKF方法中的四阶和五阶方法

- 五阶方法

$$y(x+h) = y(x) + \sum_{i=1}^{6} a_i F_i$$
$$= y(x) + \frac{16}{135}F_1 + \frac{6656}{12825}F_3 + \frac{28561}{56430}F_4 - \frac{9}{50}F_5 + \frac{2}{55}F_6$$

注意其中F_2项的系数为零

- 四阶方法

$$\bar{y}(x+h) = y(x) + \sum_{i=1}^{6} b_i F_i$$
$$= y(x) + \frac{25}{216}F_1 + \frac{1408}{2565}F_3 + \frac{2197}{4104}F_4 - \frac{1}{5}F_5$$

注意其中F_2和F_6项的系数为零

RKF45 vs RKF54

- 如果以四阶方法计算的结果作为最终结果，那么称为RKF45方法
- 反之，如果以五阶方法计算的结果作为最终结果，那么称为RKF54方法
- 一般认为RKF54方法较好，但也存在精度被减少的情形，因此两种方法孰优孰劣，没有定论

自适应步长选择的策略

假设要构造一个自适应算法，试图使局部截断误差e的值限定在预先给定的容限δ之内

- 第一种策略：假设$y(x) - \bar{y}(x)$估计的局部截断误差为Ch^5。当加倍步长导致$C(2h)^5 < \delta/4$时，对步长进行加倍是合理的。因此当局部截断误差小于$\delta/128$时，对步长进行加倍。另一方面，按自然的方法对步长进行减半，即一旦误差大于δ，就对步长进行减半
- 第二种策略：在每一步计算后，采用如下通用公式确定下一步的步长：

$$h \leftarrow 0.9h\left(\frac{\delta}{|e|}\right)^{\frac{1}{1+p}}$$

其中p是一对Runge-Kutta方法中第一个公式的阶。这个公式既可能增加步长，也可能减小步长

RKF方法的推广

- 由一对阶分别为p和q的Runge-Kutta方法构成，其中两者共享高阶方法的函数求值，这种方法称为嵌入式Runge-Kutta方法(embedded Runge-Kutta procedures). 一般采用$q = p + 1$
- 通常是用来求解非刚性初值问题的有效方法
- 现在已推导出了大量带有复杂系数的高阶Runge-Kutta方法
- 除非采用的是自适应格式，经典的四阶Runge-Kutta方法仍是最常用的

程序作业

- 应用RKF45或RKF54方法，设计实现自适应方法，求解如下常微分方程初值问题：

$$\begin{cases} y' = e^{yx} + \cos(y - x), \\ y(1) = 3 \end{cases}$$

- 初值步长取为$h = 0.01$. 在自适应方法中步长的选取采用第二种策略。
- 在解溢出前终止。
- 程序的输出：
  1. 解的范围: $[1, ?]$
  2. 提示输入一个介于上述范围的值，应用简单的两点线性插值计算出对应的函数值。