1. Häufige Fehler und Tipps

Im Feedback stand, dass ihr manchmal mühe habt mit MATLAB. Bitte schreibt mir per Mail, wenn ihr etwas genaue erklärt haben wollt. (Wir schauen in der Übung die Rolle von *polyfit* in der Serie 4).
Es gab keine grosse Probleme Bei Serie 5. Falls man etwas induktiv zeigen will, sollte man explizit auch die Induktion aufschreiben, um die volle Punktzahl zu kriegen.

Tipps:

1. Bei 2.a) schreibe die DGL für den Position Vektor einzeln für die x und y Koordinate auf.
2. Schreibe dann die beiden DGLs zweiter Ordnung in jeweils 2 DGLs erster Ordnung um.

2. Role of *polyfit*

Wenn man über die Konvergenz-Ordnung/Rate reden, betrachten wir Ungleichungen der Form

$$||Q[I] - I|| \leq CN^{\alpha}, \quad \text{for N big enough.}$$

Üblicherweise (in den Serien) benutzen wir $\log\log$-plots, dann kriegen wir die Ungleichung

$$\log(||Q[I] - I||) \leq \log(C) + \alpha \log(N).$$

Jetzt sollte man sehen, dass es einfach ein lineares Polynom in variable $\log(N)$ ($\alpha x + \log(C)$).
So wenn wir wenden *polyfit* auf $\log(N)$ und $\log(||Q[I] - I||)$ an, und kriegen das Array $[\alpha, \log(C)]$.

3. Runge-Kutta methoden

Die **Runge-Kutta-Methoden** sind bei weitem die beliebtesten und leistungsfähigsten (general) numerischen Methoden zur Integration gewöhnlicher Differentialgleichungen. Die Idee hinter den Runge-Kutta-Methoden ist es, f bei sorgfältig ausgewählten Stellen auszuwerten um eine Approximation zu konstruieren, die so genau ist wie eine *high-order* Taylorexpansion, ohne irgendwelche Ableitungen von f auszuwerten.
Wir definieren ein s-stufiges Runge-Kutta Verfahren als

$$x^{j+1} = x^j + h \sum_{i=1}^{s} b_i k_i,$$

wobei die Stufen $k_i = f(t_j + c_i h, y_j + h \sum_{l=1}^{s} a_{i,l} k_l)$ sind.
Die Methoden die wir bisher gesehen haben sind alle *one-step* Runge-Kutta Methoden. Die generelle n*-step* Methoden sind von der Form.

$$\sum_{j=0}^{n} \alpha_j x^{k+j} = \Delta t \sum_{j=0}^{n} \beta_j f(t_{k+1}, x^{k+j}),$$

wobei α_j, β_j reelle Konstanten sind und $\alpha_n \neq 0$.
Falls jetzt $\beta_n = 0$, dann erhält man x^{k+n} explizit von den vorherigen x^j und $f(t_j, x^j)$. Man nennt dann die Methode (drum roll please)... **explizit**. Falls $\beta_n \neq 0$ nennet man die Methode **implizit**.

Beispiel 1.

(i) Die four-step Adams-Bashforth Methode

$$x^{k+4} = x^{k+3} + \frac{(\Delta t)}{24}\left[55f\left(t_{k+3}, x^{k+3}\right) - 59f\left(t_{k+2}, x^{k+2}\right) + 37f\left(t_{k+1}, x^{k+1}\right) - 9f\left(t_k, x^k\right)\right]$$

ist ein Beispiel von einer expliziten four-step Methode;

(ii) Die two-step Adams-Moulton Methode

$$x^{k+2} = x^{k+1} + \frac{(\Delta t)}{12}\left[5f\left(t_{k+2}, x^{k+2}\right) + 8f\left(t_{k+1}, x^{k+1}\right) + f\left(t_k, x^k\right)\right]$$

ist ein Beispiel von einer impliziten two-step Methode.

4. Butcher-Tableau (BT)

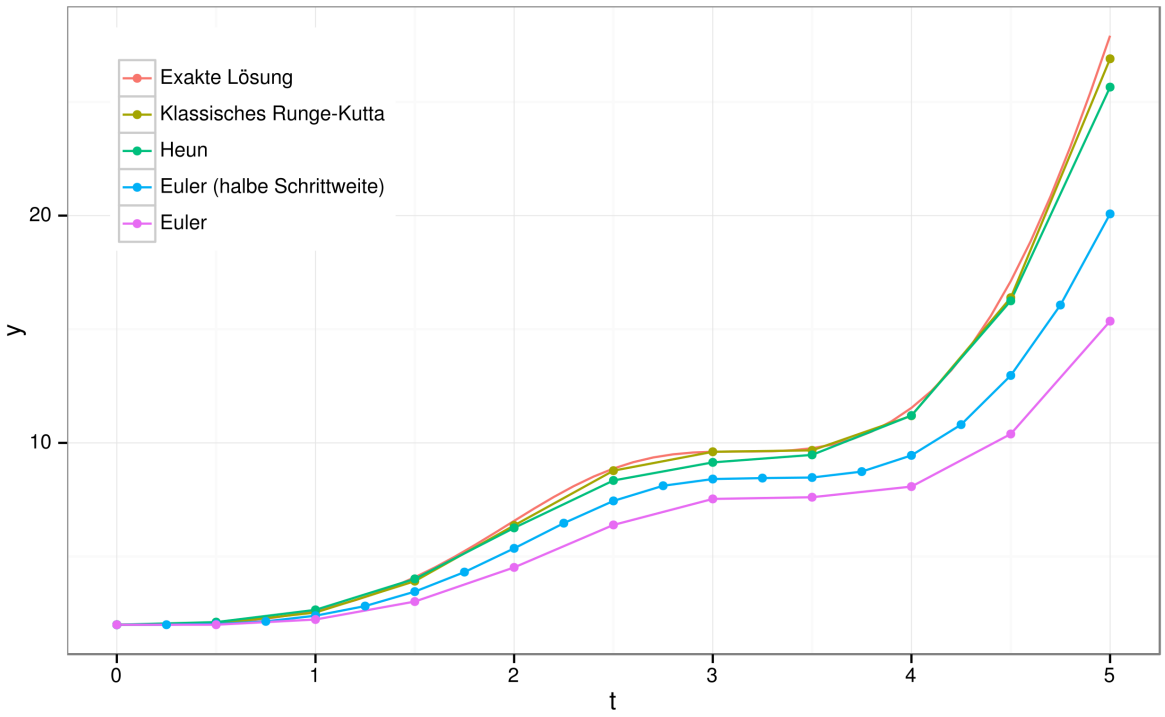
Um effizient mit Runge-Kutta (RK) Methoden arbeiten zu können benutzen wir die sogenannten Butcher-Tableau um die RKs zu klassifizieren. Sie sind von der Form

$\mathbf{c}$	A
	$\mathbf{b}^T$

Dann kann man einfach sehen, dass bei explizite RK Methoden A eine untere Dreiecksmatrix ist.
Das bekannteste ist die klassische RK-methode ("The RK4 method")

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Unten seht ihr ein Vergleich der RK4 Methode für die DGL $y' = sin(t)^2 y$.


Exakte Lösung
Klassisches Runge-Kutta
Heun
Euler (halbe Schrittweite)
Euler
20
10
y
0
1
2
3
4
5
t