AUSGEWÄHLTE KAPITEL AUS "NUMERISCHE METHODEN IN DER PHYSIK"
H. Sormann SS 2010 C

Drittes Thema: Steife Differentialgleichungen

1. Was sind steife Differentialgleichungen?
2. Genauigkeits- und Stabilitätsprobleme
3. Ein wenig Analytik
4. Implizite Rechenverfahren
5. Nicht-lineare steife Differentialgleichungen
6. Das Rosenbrock-Programmpaket
7. Das Foucault'sche Pendel
8. Ein Problem aus der Biologie: HIRES

1. Was sind steife Differentialgleichungen?

Immer wieder führt die Behandlung von physikalisch-technischen Problemen zu Anfangswertproblemen (Differentialgleichungen bzw. Systemen von Differentialgleichungen plus Anfangsbedingungen), deren numerische Auswertung mittels der im allgemeinen zuverlässigen Verfahren wie Runge-Kutta oder Fehlberg usw. massive Probleme bereitet. Sei es, daß diese Verfahren überhaupt keine vernünftigen Ergebnisse hervorbringen, sei es, daß zur Erlangung halbwegs brauchbarer Resultate unverhältnismäßig große Stützpunktzahlen (und damit Rechenzeiten) erforderlich sind.

Eine Analyse der vorliegenden Differentialgleichungen ergibt dann sehr oft, daß es sich dabei um sogenannte *steife Differentialgleichungen (stiff differential equations)* handelt.

Im folgenden sollen Sie unter anderem erfahren, was steife Differentialgleichungen sind, welche numerischen Probleme sie verursachen und welche speziellen numerischen Verfahren mit diesen Problemen fertig werden können. Vor allem aber sollen Sie auch erfahren, welche konkreten Fragestellungen aus Physik, Chemie usw. zu solchen unangenehmen Differentialgleichungen führen.

In meinem Skriptum zur Hauptvorlesung "Numerische Methoden in der Physik" gebe ich im Kapitel 8 eine Einführung in die wichtigsten numerischen Methoden zur Lösung von Anfangswertproblemen (Cauchy-Problemen). Obwohl ich im vergangenen Wintersemester dieses Kapitel nicht vorgetragen

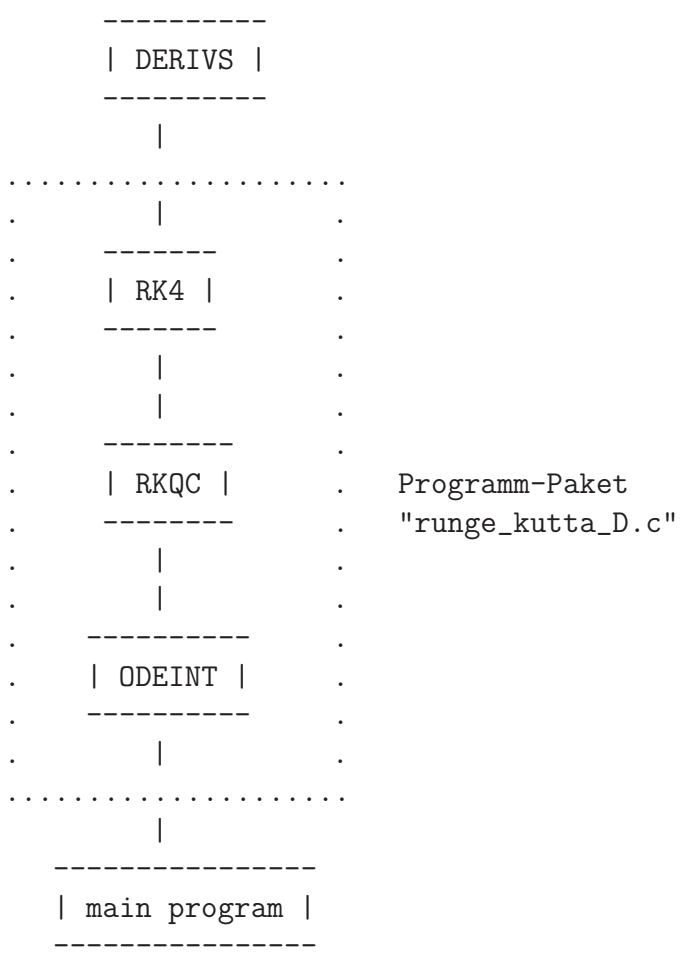
habe, sollten Sie keine Probleme haben, die wesentlichen Aspekte des *Runge-Kutta-Verfahrens* zu verstehen. Es ist für die Zwecke dieser Übung auch gar nicht erforderlich, alle Details des Numerik-Kapitels 8 zu beherrschen; es genügt, wenn Sie sich die wesentlichen Punkte (Aufstellung einer Runge-Kutta-Formel, Schrittweitensteuerung, ...) aneignen.

Abgesehen von den speziellen Programmen zur numerischen Behandlung steifer Systeme werden im Rahmen dieser Übung einige Tests mit dem *Runge-Kutta*-Verfahren durchzuführen sein. Das entsprechende C-Programm finden Sie unter

```
runge_kutta_D.c
```

auf der Website dieser Lehrveranstaltung, und einige Anmerkungen zur Verwendung dieses Programms folgen hier:

Verwendung des Runge-Kutta-Programmpaketes:
(Zitat aus: num-2006/uebung4)



Wie im Skriptum im Kapitel 8.5 beschrieben, besteht das Runge-Kutta-System aus den Elementen ODEINT, RKQC und RK4, wobei ODEINT das Programm ist, das von Ihrem Hauptprogramm aufgerufen werden muß.

Vom Benutzer ist außerdem das Programm DERIVS beizustellen, welches das gegebene Problem in Form eines Systems von n Differentialgleichungen erster Ordnung definiert:

Struktur des C-Programmes:

```
// Runge-Kutta Programm nach den Struktogrammen 26,27,28 im Skriptum,
// Ausgabe fuer WS 2002/2003.

// ACHTUNG: "lokale Extrapolation" Glg. (8.27) INAKTIV !!

#include <stdio.h>
#include <math.h>
#include "nrutil.c"


void derivs(double x, double y[], double f[])
//  Definition der f-Funktionen
{
  f[1]=.... ;   // Def. von f1(x, vec y)
  f[2]=.... ;   // Def. von f2(x, vec y)
   .
   .
}

#include "runge_kutta_D.c"    // Einbeziehung des Runge-Kutta-Programms,
                              // das Sie ueber Internet beziehen koennen.
                              // Das "Kontaktprogramm", welches Sie vom
                              // main-Programm aufrufen muessen, hat den
                              // Namen ODEINT: im folgenden eine kurze
                              // Beschreibung von Headline und Parametern:

// void odeint(double ystart[], int nvar, double x1, double x2, double eps,
//             double h1, double hmin, int nstmax, int *nwerte,
//             double xx[], double **yy)

//  Dieses Programm stellt das driver program fuer das RuKu-Programmsystem
//  dar, das aus den Prozeduren ODEINT, RKQC, RK4 und DERIVS besteht.
//  (s. Skriptum Seite 249ff).

//  Last update: 2-10-96

//  Es ist zu beachten:
//  ==========================
//  Es muss ein Unterprogramm
//         void derivs(double x, double y[], double f[])
//      existieren, welches an dem gegebenen Punkt x und mit Kenntnis
//      der Funktionswerte y[] die entsprechenden 'rechten Seiten' der
//      Differentialgleichungen f[] berechnet.
```

```
//  Bedeutung der Parameter:
//  ========================
//      Input:  ystart[]   Vektor mit den Anfangswerten des Dgl-Systems
//              nvar       Zahl der Gleichungen des Dgl-Systems
//              x1,x2      Startpunkt/Endpunkt des Integrationsintervalls
//              eps        geforderte relative Genauigkeit
//              h1         guessed value fuer die Anfangs-Schrittweite des
//                         RuKu-Prozesses
//              hmin       Minimalwert, unter den die Arbeitsschrittweite
//                         nicht sinken darf
//              nstmax     maximale Zahl von Stuetzpunkten zwischen x1 und
//                         x2, die in den Feldern xx[] bzw. yy[,]
//                         abgespeichert werden

//      Output: nwerte     Zahl der abgespeicherten Stuetzpunkte
//              xx[]       Abszissenwerte der Stuetzpunkte
//              yy[][]     Ordinatenwerte der Loesungsfunktionen:
//                            erster Index  = Index der Funktion
//                            zweiter Index = Nummer des Abszissenwertes


int main()
{
   int nvar,nstmax,nwerte,... ;
   double t0,tmax,eps,h1,hmin;
   double *ystart,*tfeld,**yfeld;

   nvar=... ;
   nstmax=... ;
   ystart=dvector(1,nvar);
   tfeld=dvector(1,nstmax);
   yfeld=dmatrix(1,nvar,1,nstmax);

   t0=... ;
   tmax=... ;

   ystart[1]=... ;
   ystart[2]=... ;
   eps=... ;
   h1=... ;           // h1=0.1   fuer Aufgabe 1
   hmin=... ;         // hmin=0.0 fuer Aufgabe 1

   odeint(ystart,nvar,t0,tmax,eps,h1,hmin,nstmax,&nwerte,tfeld,yfeld);
    .
   return (0);
}
```

Aufgabe 1

Als Einstieg ins Thema "stiff equations" sollen Sie mit Hilfe des Runge-Kutta-Programms die beiden folgenden linearen Anfangswertprobleme lösen, deren exakte[1] Lösungen bekannt sind:

Testbeispiel 1:

$$y'(t) = 3 - y(t) \qquad y(0) = 1 \qquad \text{exakt:} \quad y(t) = 3 - 2e^{-t}$$

Testbeispiel 2:

$$y'(t) = -1000y(t) + 3000 - 2000\,e^{-t} \qquad y(0) = 0$$

$$\text{exakt:} \quad y(t) = 3 - 0.998\,e^{-1000t} - 2.002\,e^{-t} \tag{1}$$

Beide Probleme sollen für den Zeitbereich $0 \leq t \leq 4$ gelöst werden.

- Berechnen Sie beide numerischen Lösungen mit einer relativen Genauigkeit von 10^{-6}.
- Stellen Sie beide Lösungen in Diagrammform dar, wobei Sie die exakte Kurve als Linie und die vom Runge-Kutta-Programm berechneten Stützpunkte als Punkte wiedergeben.
- Wie viele Stützpunkte werden zur Erreichung der geforderten Genauigkeit benötigt?

Wenn Sie diese Tests korrekt durchgeführt haben, sollten die erhaltenen Diagramme die folgenden Informationen geben:

- Test 1: es genügt eine relativ kleine Zahl von Stützpunkten, um eine numerische Lösung zu erhalten, die ausgezeichnet mit der exakten Kurve übereinstimmt.
- Test 2: Die Lösungskurve ist praktisch identisch mit der vom Test 1, mit einem wichtigen Unterschied: die Kurve startet am Punkt $y = 0$ *und "springt" innerhalb einer extrem kurzen Zeit auf den Punkt* $y = 1$. Genau hier liegen die Schwierigkeiten: zur numerischen Beschreibung dieses "Sprungs" von 0 auf 1 wird eine sehr hohe Punktdichte benötigt - was an sich keine Überraschung ist. Das Erstaunliche und Unerfreuliche ist jedoch, *daß diese hohe Punktdichte auch während des moderaten Anstiegs der Kurve von 1 auf den "Sättigungswert" 3 erhalten bleibt*, und dies trotz der im Runge-Kutta-Programm enthaltenen Schrittweiten-Optimierung!

  Die Folge dieses Verhaltens ist eine enorm große Stützpunktanzahl bei der numerischen Auswertung.

[1] Um ehrlich zu sein: die oben angegebene "exakte" Lösung des 2. Testbeispiels ist nur eine sehr gute, für unsere Zwecke ausreichende Näherung.

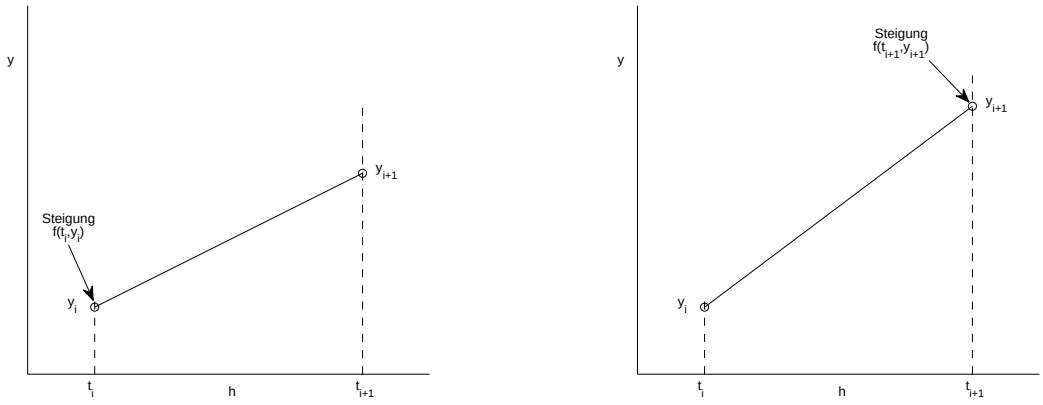


Figure 1: Geometrische Interpretation der Methode von Euler - (links) explizit und (rechts) implizit.

Das 2. Testbeispiel zeigt alle Eigenschaften einer *steifen Differentialgleichung (stiff differential equation)*. Abgesehen von mathematisch exakteren Definitionen (s.u.) kann eine solche Gleichung so charakterisiert werden:

- Eine steife Differentialgleichung erkennt man daran, daß sich ihre Lösungsfunktion [s. Glg. (1)] aus Termen zusammensetzt, die *völlig verschiedenen Zeitskalen unterliegen*.

2. Genauigkeits- und Stabilitätsprobleme

In der nun folgenden

Aufgabe 2

sollen Sie das manchmal unbefriedigende Verhalten konventioneller Programme bei der numerischen Lösung von Anfangswertproblemen genauer studieren. Dazu sollen Sie ein einfaches Programm schreiben, das auf der "Euler'schen Methode" beruht, die ich im VL-Skriptum in den Abschnitten 8.1-8.3 beschrieben habe.

Aus diesem Grund soll hier die Euler-Methode zur Lösung des Anfangswertproblems erster Ordnung

$$y'(t) = f(t, y) \qquad \text{mit} \quad y(t_0) = y_0$$

nur ganz kurz beschrieben werden. Das Programm soll keinerlei Schrittweiten-Steuerung enthalten, d.h. die vom Benutzer anfangs gewählte Schrittweite h bleibt während der gesamten Rechnung konstant.

In diesem Fall ist der Algorithmus denkbar einfach: man zerlegt den Bereich $[t_0, t_{max}]$, für den die numerische Lösung $\hat{y}(t)$ berechnet werden soll, in Subintervalle mit der Breite h, d.h. man definiert die Stützpunkte

$$t_i = t_0 + (i-1) * h \qquad (i = 1, i_{max}) \,.$$

Der Euler'sche Algorithmus hat nun die Form

$$\hat{y}_1 = y_0$$

und [s. Abb. 1 (links)]

$$\hat{y}_{i+1} = \hat{y}_i + h\, f(t_i, \hat{y}_i) \qquad (i = 1, \ldots, i_{max} - 1) \,. \tag{2}$$

Die Euler-Formel ist linear in der Schrittweite h. Der bei jedem Rechenschritt auftretende *lokale* Verfahrensfehler muß daher proportional zur zweiten Potenz von h sein, also

$$E_V^{Euler} \propto h^2 \,. \tag{3}$$

Schreiben Sie ein Euler-Programm, und werten Sie damit die beiden Testbeispiele von Aufgabe 1 aus.

Testbeispiel 1:

Verwenden Sie Ihr Euler-Programm mit verschiedenen (konstanten) Schrittweiten, z.B. $h = 0.5$, $h = 0.25$ und $h = 0.1$: Sie werden sehen, wie die numerisch erhaltenen Ergebnisse der exakten Kurve immer näher kommen; das ist natürlich klar, denn je kleiner man die Euler-Schritte macht, desto geringer wird der Verfahrensfehler.

Unabhängig von diesem mehr oder weniger großen Verfahrensfehler ist das numerische Verfahren jedoch stabil, d.h. die numerischen Werte zeigen kein unkontrolliertes, sprunghaftes Fehlerverhalten, wie es für steife Anfangswertprobleme typisch ist.

Testbeispiel 2:

Es war ein wichtiges Ergebnis der Aufgabe 1, daß steife Systeme eine sehr hohe Punktdichte (= sehr kleine Schrittweite h) benötigen, um die Lösungsfunktion erfolgreich approximieren zu können.

Sie sollen nun das Verhalten solcher Systeme untersuchen: was geschieht, wenn man mit einer kleinen Schrittweite beginnt und dann das h schrittweise vergrößert? Beschreiben Sie das Verhalten der numerischen Euler-Lösung (2) für die folgenden h-Werte:

h	Verhalten der Euler-Loesung:
0.0010	
0.0012	
0.0014	
0.0016	
0.0018	
0.0020	

Dieses "Experiment" soll Ihnen den Unterschied zwischen einem *ungenauen* und einen *instabilen* numerischen Ergebnis klar machen.
Zum Abschluß der zweiten Aufgabe testen Sie bitte mit Hilfe Ihres Euler-Programms eine Idee, die einen Ausweg aus dem Dilemma bieten könnte, in dem wir stecken. Einerseits geben nur sehr kleine Schrittweiten die Gewähr, daß der numerische Prozess stabil ist, andererseits führen diese hohen Punktdichten zu großen Rechenzeiten.

Die Idee lautet so:

(1) Die kleinen Schrittweiten sind offenbar nötig, um den raschen Funktionsanstieg von 0 auf 1 nahe bei $t = 0$ beschreiben zu können.

(2) Wäre es da nicht sinnvoll, zu einem späteren Zeitpunkt (also deutlich entfernt von der "Krisenregion" bei $t = 0$) die Schrittweite z.B. zu verdoppeln, um wenigstens ab da Stützpunkte und damit Rechenzeit einzusparen?

- Variieren Sie Ihr Euler-Programm gemäß dieser Idee, d.h. rechnen Sie für $0 \leq t < 2$ mit der Schrittweite $h = 0.0015$ und im Bereich $t > 2$ mit der doppelten Schrittweite $h = 0.0030$.
- Beschreiben Sie, was geschieht: ist diese Idee erfolgreich?

3. Ein wenig Analytik

Die Ursache für die häufige Instabilität der Euler-Methode (und auch anderer Runge-Kutta-Formeln) kann für das einfache Anfangswertproblem

$$y' = -c\,y \qquad \text{mit} \quad c > 0 \qquad \text{und} \quad y(t_0) = y_1$$

leicht gefunden werden. Die exakte Lösung des obigen Problems ist offensichtlich

$$y_{exakt}(t) = y_1\,\mathrm{e}^{-ct}$$

und geht für $t \to \infty$ asymptotisch gegen Null.

Wendet man die Formel (2) auf das obige Problem an, so ergibt sich durch den ersten Euler-Schritt

$$\hat{y}(t_0 + h) = \hat{y}_2 = y_1 - hcy_1 = y_1(1 - hc)\,.$$

Der zweite Euler-Schritt führt zu

$$\hat{y}(t_0 + 2h) = \hat{y}_3 = \hat{y}_2 - hc\hat{y}_2 = \hat{y}_2(1 - hc) = y_1(1 - hc)^2\,.$$

Für den gesamten Euler-Prozess lauten die Lösungen

$$\hat{y}_{i+1} = y_1(1 - hc)^i \qquad \text{mit} \quad i = 1, 2, \ldots \tag{4}$$

Natürlich erwartet man von der numerischen Lösung, daß sie für $i \to \infty$ wie die exakte Gleichung gegen Null geht. Offensichtlich ist dies aber nur der Fall, wenn die Bedingung

$$|1 - hc| < 1$$

erfüllt ist, was nur der Fall ist, wenn gilt:

$$h < \frac{2}{c}\,. \tag{5}$$

Wird diese Bedingung nicht erfüllt, *divergiert* der Euler-Prozess, d.h. das numerische Verfahren wird instabil.

Dieses Ergebnis ist durchaus plausibel, denn es sagt folgendes aus: wenn die (positive) Konstante c groß gegenüber 1 ist, d.h., wenn die Lösung der Differentialgleichung *sehr schnell gegen Null tendiert*, muß mit entsprechend kleinen Schrittweiten gearbeitet werden.

Ein Problem tritt erst dann auf, wenn - wie es typisch für *stiff differential equations* ist - die Lösungsfunktion aus Termen mit ganz verschiedenen Zeitskalen besteht (s. Testbeispiel 2, S. 5), weil dann die Stabilitätsbedingung *für das größte in der Lösung vorkommende c gilt*, andererseits sich aber der Zeitbereich, in dem man die Lösung haben will, am kleinsten c orientiert.

Bevor nun effiziente Lösungsmöglichkeiten behandelt werden, soll die Analyse, die zur Bedingung (5) geführt hat, auf den Fall linearer Differentialgleichungs-*Systeme* mit konstanten Koeffizienten erweitert werden:

Ein solches, aus n Gleichungen bestehendes System

$$\begin{aligned}
y^{(1)\prime} &= -c_{11}y^{(1)} - c_{12}y^{(2)} - \cdots - c_{1n}y^{(n)} \quad y^{(1)}(0) = y_1^{(1)} \\
y^{(2)\prime} &= -c_{21}y^{(1)} - c_{22}y^{(2)} - \cdots - c_{2n}y^{(n)} \quad y^{(2)}(0) = y_1^{(2)} \\
&\vdots \\
y^{(n)\prime} &= -c_{n1}y^{(1)} - c_{n2}y^{(2)} - \cdots - c_{nn}y^{(n)} \quad y^{(n)}(0) = y_1^{(n)}
\end{aligned}$$

kann kompakt in der Form

$$\mathbf{y}' = -\mathbf{C}\cdot\mathbf{y} \tag{6}$$

geschrieben werden, wobei $\mathbf{C}$ eine *positiv-definite* nxn-Matrix bedeutet. In diesem Fall lautet die Euler-Formel für den Zeitpunkt $t = (i+1)h$

$$\hat{\mathbf{y}}_{i+1} = (\mathbf{1} - h\mathbf{C})\cdot\hat{\mathbf{y}}_i \qquad \text{mit} \quad i = 1, 2, \ldots \tag{7}$$

bzw. [vgl. Glg. (4)]

$$\hat{\mathbf{y}}_{i+1} = (\mathbf{1} - h\mathbf{C})^i\cdot\mathbf{y}_1\,. \tag{8}$$

Nun lehrt die Lineare Algebra:

- *Die obige Entwicklung ist stabil, wenn der größte Eigenwert der Matrix* $(\mathbf{1} - h\mathbf{C})$ *kleiner als 1 ist.*

Diese Bedingung läßt sich leicht als die Ungleichung

$$h < \frac{2}{\lambda_{max}} \tag{9}$$

formulieren, wobei λ_{max} der größte Eigenwert der Koeffizientenmatrix $\mathbf{C}$ ist.

Dieses Verhalten soll an Hand eines "klassischen" Beispiels aus der einschlägigen Literatur[2] dokumentiert werden:

$$\begin{array}{ll} y^{(1)\prime} = 998y^{(1)} + 1998y^{(2)} & y^{(1)}(0) = 1 \\ y^{(2)\prime} = -999y^{(1)} - 1999y^{(2)} & y^{(2)}(0) = 0 \end{array} \tag{10}$$

Die exakte Lösung dieses Problems lautet

$$\begin{aligned} y^{(1)}(t) &= 2\mathrm{e}^{-t} - \mathrm{e}^{-1000t}\,, \\ y^{(2)}(t) &= -\mathrm{e}^{-t} + \mathrm{e}^{-1000t}\,. \end{aligned} \tag{11}$$

Wie man leicht ausrechnen kann, hat die Koeffizientenmatrix

$$\mathbf{C} = \begin{pmatrix} -998 & -1998 \\ 999 & 1999 \end{pmatrix}$$

die Eigenwerte $\lambda_1 = 1$ und $\lambda_2 = 1000$.

Dementsprechend lautet die Stabilitätsbedingung für dieses Problem

$$h < \frac{2}{\lambda_{max}} \qquad \text{bzw.} \quad h < 0.002 \tag{12}$$

4. Implizite Rechenverfahren

Kehren wir nun nochmals zum *linearen* System von Differentialgleichungen

$$\mathbf{y}' = -\mathbf{C} \cdot \mathbf{y} \tag{13}$$

zurück, das wir bisher mit dem *expliziten* Euler-Ansatz

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{y}}_i + h\,\hat{\mathbf{y}}'_i \tag{14}$$

berechnet haben. Dabei werden offenbar die Steigungen der Geraden, durch welche die unbekannten Lösungsfunktionen im Intervall $[t_i, t_{i+1}]$ approximiert werden, durch die Funktionswerte $\mathbf{y}'$ am linken Rand dieses Zeitintervalls berechnet [s. Abb. 1, links].

Man kann aber für diese Steigungen die Funktionswerte $\mathbf{y}'$ am rechten Rand des Zeitintervalls nehmen [s. Abb. 1, rechts], was zur Euler-Formel

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{y}}_i + h\,\hat{\mathbf{y}}'_{i+1} \tag{15}$$

führt. Man spricht in diesem Fall von einer *impliziten Euler-Formel*, weil man durch Einsetzen von Glg. (13) in (15) den Ausdruck

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{y}}_i - h\,\mathbf{C} \cdot \hat{\mathbf{y}}_{i+1}$$

[2] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Englewood Cliffs, NJ: Prentice-Hall, 1971).

erhält, in dem die Unbekannte $\hat{\mathbf{y}}_{i+1}$ sowohl links als auch rechts auftritt. Wegen der Linearität des Problems bzgl. $\mathbf{y}$ ist die Auswertung dieser impliziten Gleichung simpel: er ergibt sich

$$\hat{\mathbf{y}}_{i+1} = (\mathbf{1} + h\mathbf{C})^{-1} \cdot \hat{\mathbf{y}}_i \tag{16}$$

und schließlich

$$\hat{\mathbf{y}}_{i+1} = (\mathbf{1} + h\mathbf{C})^{-i} \cdot \mathbf{y}_1 \qquad (i = 1, 2, \ldots)\,. \tag{17}$$

In dieser Gleichung bedeutet $(\mathbf{1} + h\mathbf{C})^{-i}$ die Inverse der Matrix $(\mathbf{1} + h\mathbf{C})$, i-mal mit sich selbst multipliziert. Wie bereits oben erwähnt, ist die Voraussetzung für eine numerische Stabilität der Glg. (17) für $i \to \infty$, daß der Betrag des größten Eigenwertes der Matrix $(\mathbf{1} + h\mathbf{C})^{-1}$ kleiner als 1 ist. Nennt man die Eigenwerte von $\mathbf{C}$ λ, so gilt wegen der Positiv-Definitheit von $\mathbf{C}$ stets $\lambda > 0$, was natürlich bedeutet:

$$\textit{alle}\ \text{Eigenwerte} \quad \frac{1}{(1 + h\lambda)} < 1\,.$$

Das bedeutet, daß die *implizite Euler-Gleichung* (15) für alle Schrittweiten h stabil ist.

Aufgabe 3

- Erweitern Sie das Euler-Programm, das Sie für die Aufgabe 2 geschrieben haben, für Anwendungen auf Systeme von linearen Anfangswertproblemen vom Typus (13).

- Versuchen Sie das "Gear-Beispiel" von S. 10 [Dgl-System s. Eq. (10], exakte Lösungen s. Eq. (11)] mit dem *expliziten* Euler-Verfahren [s. Glg. (7)] zu lösen, und studieren Sie das Stabilitätsverhalten der numerisch erhaltenen Lösungsfunktionen im Bereich der "kritischen" Schrittweite $h = 0.002$.

  Produzieren Sie zu diesem Zweck 6 Diagramme, in denen Sie die numerischen Lösungen für die h-Werte: 0.0005, 0.0017, 0.0018, 0.0019, 0.0020 und 0.0021 den exakten Lösungen gegenüberstellen. Verwenden Sie für alle Diagramme dieselben Achsenabschnitte, nämlich $0 \le t \le 0.3$ für die Abszisse und $-2 \le y \le 2.5$ für die Ordinate.

- Erstellen Sie ein entsprechendes Programm, welches die *implizite* Euler-Formel (16) verwendet, und zeigen Sie, daß Stabilität für alle h gegeben ist, insbesondere im Bereich der Schrittwerte $h = 0.0020$, wo beim expliziten Euler-Verfahren große Probleme auftreten.

  Produzieren Sie zu diesem Zweck ebenfalls eine Serie von 6 Diagrammen, und zwar mit denselben Schrittweiten und denselben Achsenabschnitten wie oben.

5. Nicht-lineare steife Differentialgleichungen

Die bisherigen Beispiele haben sich auf die Lösung des einfachen Systems

$$\mathbf{y}' = -\mathbf{C} \cdot \mathbf{y}$$

von steifen Differentialgleichungen beschränkt, wobei die Koeffizientenmatrix $\mathbf{C}$ aus konstanten Koeffizienten bestand.

Im Folgenden soll nun gezeigt werden, wie ein implizites Lösungsverfahren auch für das allgemeinere System von Differentialgleichungen

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}) \tag{18}$$

funktionieren kann. In diesem Fall würde der implizite Euler-Ansatz als Verallgemeinerung des Spezialfalls (15)

$$\hat{\mathbf{y}}_{i+1} = \mathbf{y}_i + h\,\mathbf{f}(\hat{\mathbf{y}}_{i+1}) \tag{19}$$

lauten. Das Problem liegt nun darin, daß man diese Gleichung nicht - wie im linearen Fall - so ohne weiteres nach $\hat{\mathbf{y}}_{i+1}$ auflösen kann. Man muß den obigen Ansatz geeignet *linearisieren*, was dadurch geschehen kann, daß man die Vektorfunktion $\mathbf{f}$ an der Stelle $\hat{\mathbf{y}}_i$ taylor-entwickelt und diese Entwicklung nach dem linearen Term abbricht:

$$\mathbf{f}(\hat{\mathbf{y}}_{i+1}) \approx \mathbf{f}(\hat{\mathbf{y}}_i) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right)_{\hat{\mathbf{y}}_i} \cdot (\hat{\mathbf{y}}_{i+1} - \hat{\mathbf{y}}_i)\,. \tag{20}$$

Hier nennt man

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right) \equiv \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \cdot & & & \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{pmatrix} \tag{21}$$

die *Jacobi-Matrix* des aus n Gleichungen bestehenden Systems. Daraus ergibt sich die Gleichung

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{y}}_i + \left[\mathbf{1} - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right]^{-1} \cdot \mathbf{f}(\hat{\mathbf{y}}_i)\,. \tag{22}$$

Anmerkung: Beim linearen Problem (13) ergibt sich

$$\frac{\partial \mathbf{f}}{\partial \mathbf{y}} = -\mathbf{C}$$

und durch Einsetzen in Glg. (22) erhält man daraus sofort die Formel (16).

Wegen der Näherung in Glg. (20) nennt man die Formel (22) eine semi-implizite Euler-Formel. Die numerische Stabilität solcher Formeln kann nicht allgemein garantiert werden, ist aber in vielen praktischen Fällen gegeben.

Nun noch ein letztes Detail bzgl. des "allgemeinen" Differentialgleichungssystems (18): diese Allgemeinheit kann sozusagen noch erhöht werden, wenn die Komponenten des Funktionenvektors $\mathbf{f}$ nicht nur von den $y_1(t), \ldots, y_n(t)$ (und damit implizit von t) abhängen, sondern auch *explizite* zeitabhängig sind, d.h.:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)\,. \tag{23}$$

In diesem Fall muß die Jacobi-Matrix (21) noch um eine Spalte erweitert werden, d.h.

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right) \equiv \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} & \frac{\partial f_1}{\partial t} \\ \cdot & & & & \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} & \frac{\partial f_n}{\partial t} \end{pmatrix}. \tag{24}$$

Um auch in einem solchen Fall zu erreichen, daß diese Matrix quadratisch ist, *erweitert man das gegebene Differentialgleichungssystem um die (formale)* $(n+1)$*-te Gleichung*

$$y'_{n+1}(t) = 1\,,$$

wodurch sich

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right) \equiv \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} & \frac{\partial f_1}{\partial t} \\ \cdot & & & & \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} & \frac{\partial f_n}{\partial t} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \tag{25}$$

ergibt. Die entsprechende, bei jedem Rechenschritt zu invertierende Matrix [s. Glg. (22)] sieht dann wie folgt aus:

$$\mathbf{1} - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \begin{pmatrix} 1 - h\frac{\partial f_1}{\partial y_1} & -h\frac{\partial f_1}{\partial y_2} & \cdots & -h\frac{\partial f_1}{\partial y_n} & -h\frac{\partial f_1}{\partial t} \\ \cdot & & & & \\ -h\frac{\partial f_2}{\partial y_1} & 1 - h\frac{\partial f_2}{\partial y_2} & \cdots & -h\frac{\partial f_2}{\partial y_n} & -h\frac{\partial f_2}{\partial t} \\ \cdot & & & & \\ \cdot & & & & \\ -h\frac{\partial f_n}{\partial y_1} & -h\frac{\partial f_n}{\partial y_2} & \cdots & 1 - h\frac{\partial f_n}{\partial y_n} & -h\frac{\partial f_n}{\partial t} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{26}$$

6. Das Rosenbrock-Programm

Abgesehen von den ersten Tests im Rahmen dieses Übungsbeispiels (Aufgabe 1) haben Sie für die weiteren Aufgaben 2 und 3 explizite bzw. implizite Versionen der Euler-Gleichung verwendet. Vom Standpunkt der Numerik stellt die Euler-Gleichung einen *Runge-Kutta-Ansatz erster Ordnung* dar, also gewissermaßen das einfachste Mitglied der Runge-Kutta-Familie. Es ist daher klar, daß die Euler-Gleichung zu einem einfachen Algorithmus führt, aber auch zu einem nicht sehr leistungsfähigen, was seine Rechengenauigkeit bei einer gegebenen Schrittweite h betrifft.

Aus diesem Grund haben natürlich viele Autoren an der Realisierung impliziter Runge-Kutta-Ansätze höherer als erster Ordnung gearbeitet. Unter diesen Versuchen ist der von *Rosenbrock* (1963) bzw. von Kaps und Rentrop (1979) einer der erfolgreichsten: es handelt sich dabei um eine implizite Variante eines Runge-Kutta-Ansatzes vierter Ordnung.

Die mathematische Darstellung des Rosenbrock- bzw. Kaps-Rentrop-Algorithmus ist nicht schwierig, aber leider ziemlich umfangreich. Ich habe mich daher entschlossen, Ihnen das entsprechende C-Programm aus den *Numerical Recipes in C*, Ausgabe 1994, mit kleinen Adaptionen von mir, ohne theoretischen Unterbau zur Verfügung zu stellen. Die vorliegende Programmversion *rosenbrock_D.c* (D bedeutet *double*-Genauigkeit) hat den Vorteil, daß die Anbindung an ein eigenes *main*-Programm (fast) gleich erfolgt wie das entsprechende Arbeiten mit dem "normalen" Runge-Kutta-Programm vierter Ordnung, das in meinem Numerik-Skriptum im Kapitel 8 beschrieben ist (s. auch den Anfang dieses Übungsskriptums).

Für diejenigen unter Ihnen, die gerne Genaueres über den Rosenbrock-Algorithmus wissen möchten, verweise ich auf die folgenden Literatur-Stellen:

[1] Press, Teukolsky, Vetterling, and Flannery, *Numerical Recipes in C*, Cambridge Uni. Press, 1994, p. 738-742.

[2] Rosenbrock, *Some general implicit processes for the numerical solution of differential equations*, Comput. J. **5**, 329 (1963).

[3] Kaps and Rentrop, *Generalized Runge-Kutta methods of order four with stepsize control for stiff ordinary differential equations*, Numer. Math. **33**, 55 (1979).

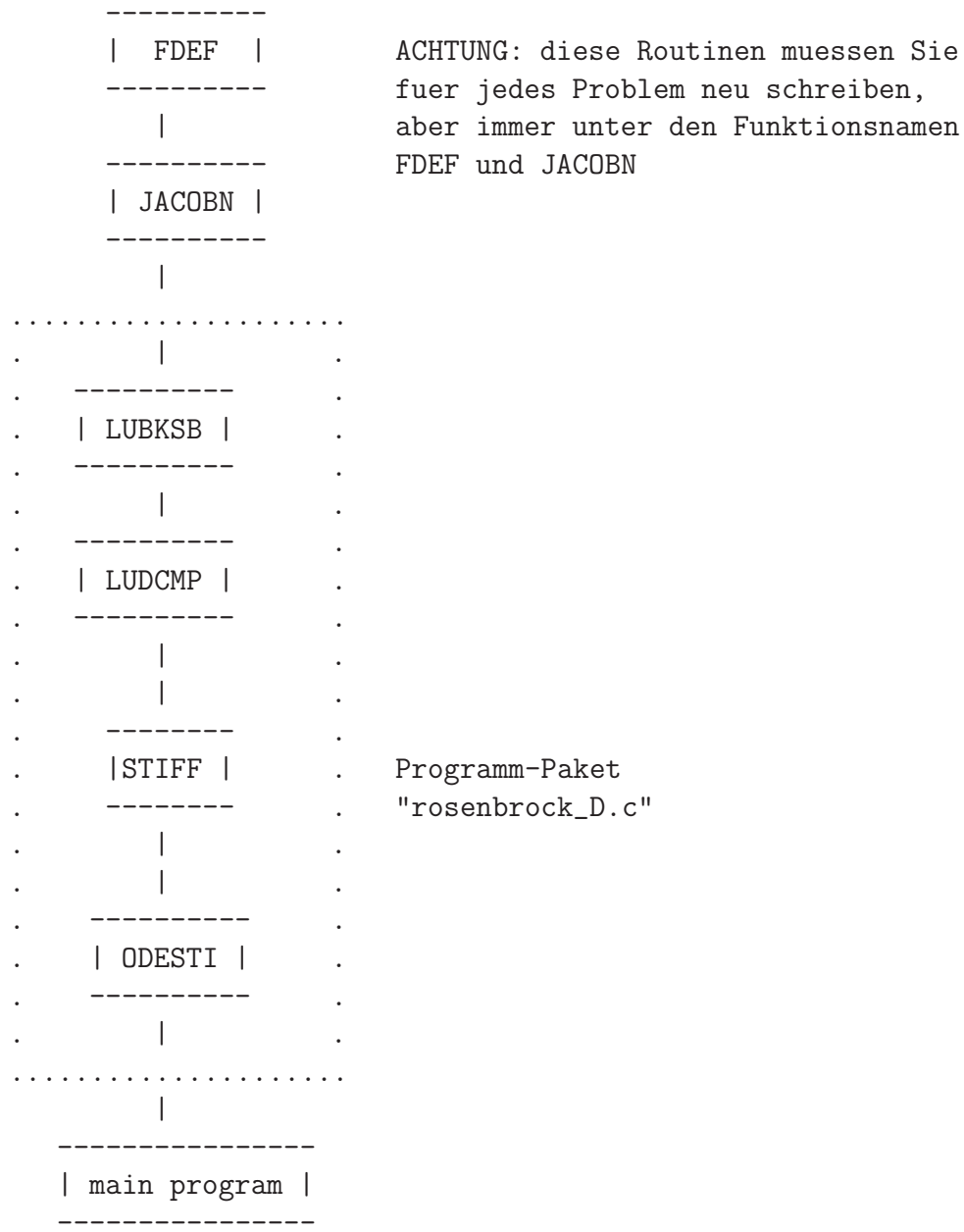
[4] Aboanber, *Stability of generalized Runge-Kutta methods for stiff kinetics coupled differential equations*, J. Phys. A: Math. Gen. **39**, 1859 (2006).

Das entsprechende C-Programm finden Sie unter

```
rosenbrock_D.c        (D bedeutet double-Version)
```

auf der Website dieser Lehrveranstaltung; zu diesem Programm, das in der Struktur und im Aufruf dem "Runge-Kutta-Programmpaket" (s. Beginn dieses Skriptums) sehr ähnlich ist, noch einige Anmerkungen:

Verwendung des Rosenbrock-Programmpaketes:



Das Rosenbrock-Programmsystem besteht aus den Elementen ODESTI, STIFF, LUDCMP und LUBKSB, wobei ODESTI das Programm ist, das von Ihrem Hauptprogramm aufgerufen werden muß.
Vom Benutzer sind außerdem die Programme FDEF und JACOBN beizustellen, welche das Problem (23) eines Systems von n *steifen* Differentialgleichungen erster Ordnung definiert.

Die Programme LUDCMP und LUBKSB sind Standardprogramme aus dem *Numerical Recipes-C*-Buch und dienen im Rosenbrock-Programm zur Invertierung der Matrix (26). Wenn es Sie interessiert, finden Sie Informationen über diese Programme in meinem Numerik-Skriptum, Kap. 2.

Struktur des C-Programmes:

```
#include <stdio.h>
#include <math.h>
#include "nrutil.c"


void fdef(double x, double y[], double f[])
// Definition der f-Funktionen des Dgls-Systems:
{
f[1] = ....;    // Def. von  f1(x, vec y)
f[2] = ....;    // Def. von  f2(x, vec y)
 .
 .
}


void jacobn(double x, double y[], double dfdx[], double **dfdy, int n)
// Definition der Jacobi-Matrix (s, Uebungsskriptum, Glg. (24)).
// n  ist die Ordnung des Gleichungssystems (nvar in der Aufrufliste
//    von ODESTI).
{

dfdx[1]= ...;      // Def. der partiellen Ableitungen der
dfdx[2]= ...;      // Funktionen f nach x:
         .
         .

dfdy[1][1]= ...;   // Def. der Jacobi-Matrix: der erste
dfdy[1][2]= ...;   // Index bezieht sich auf f, die zweite
         .         // auf y.
         .
dffy[n][n-1]= ...;
dffy[n][n]=   ...;
}

// Dazu noch einige Anmerkungen:
// (1) Wie sie aus Glg. (25) ersehen, muss die Jacobi-Matrix noch
//     um eine "Null-Zeile" erweitert werden. Das erledigt das Programm
//     intern fuer Sie, und Sie brauchen das in der Routine "jacobn"
//     NICHT zu tun.

// (2) Die Dimensionierungen der oben verwendeten Felder y, f, dfdx und
//     dfdy  werden bereits im Programm STIFF durchgefuehrt; auch darum
//     brauchen Sie sich nicht zu kuemmern.
```

```
#include "rosenbrock_D.c"  // Einbeziehung des Rosenbrock-Programms,
                           // das Sie ueber die Website dieser LV.
                           // erhalten koennen.
                           // Das "Kontaktprogramm, welches Sie von
                           // Ihrem main-Programm aufrufen muessen,
                           // hat den namen ODESTI: nun flgt eine
                           // kurze Beschreibung der Headline und
                           // der Parameter:


// void odesti(double ystart[], int nvar, double x1, double x2, double eps,
//             double h1, double hmin, int nstmax, int *nbad, int *nwerte,
//             double xx[], double **yy)


//  Bedeutung der Parameter:
//  ========================
//      Input:  ystart[]   Vektor mit den Anfangswerten des Dgl-Systems
//              nvar       Zahl der Gleichungen des Dgl-Systems
//              x1,x2      Startpunkt/Endpunkt des Integrationsintervalls
//              eps        geforderte relative Genauigkeit
//              h1         guessed value fuer die Anfangs-Schrittweite des
//                         RuKu-Prozesses
//              hmin       Minimalwert, unter den die Arbeitsschrittweite
//                         nicht sinken darf (KANN IN VIELEN FAELLEN NULL
//                         GESETZT WERDEN).
//              nstmax     maximale Zahl von Stuetzpunkten zwischen x1 und
//                         x2, die in den Feldern xx[] bzw. yy[,]
//                         abgespeichert werden

//      Output: nbad       Zahl der verworfenen Stuetzpunkte
//              nwerte     Zahl der abgespeicherten Stuetzpunkte
//              xx[]       Abszissenwerte der Stuetzpunkte
//              yy[][]     Ordinatenwerte der Loesungsfunktionen:
//                            erster Index  = Index der Funktion
//                            zweiter Index = Nummer des Abszissenwertes
```

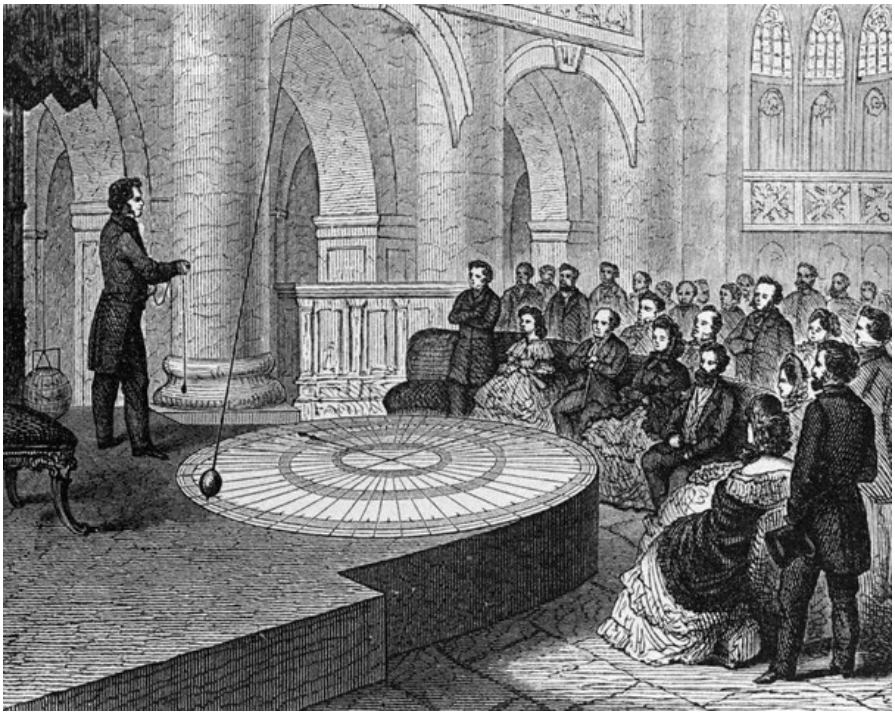
```
int main()
{
   int nvar,nstmax,nwerte,... ;
   double t0,tmax,eps,h1,hmin;
   double *ystart,*tfeld,**yfeld;

   nvar=... ;
   nstmax=... ;
   ystart=dvector(1,nvar);
   tfeld=dvector(1,nstmax);
   yfeld=dmatrix(1,nvar,1,nstmax);

   t0=... ;
   tmax=... ;

   ystart[1]=... ;
   ystart[2]=... ;
    .

   eps=... ;
   h1=... ;
   hmin=... ;

   odesti(ystart,nvar,t0,tmax,eps,h1,hmin,nstmax,&nbad,&nwerte,tfeld,yfeld);
    .
   return (0);
}
```

Noch eine Anmerkung:

Bei diesem "Rosenbrock-Programm" müssen Sie (natürlich) explizite die rechten Seiten des Differentialgleichungssystems angeben, und zwar in der Routine FDEF.

Zusätzlich müssen Sie noch - in der Routine JACOBN - die gesamte Jacobi-Matrix (24) angeben, also alle partiellen Ableitungen der Funktionen $\mathbf{f}$ nach den Funktionen $\mathbf{y}$ (Matrix dfdy) sowie alle partiellen Ableitungen der Funktionen $\mathbf{f}$ nach der unabhängigen Variablen x (Vektor dfdx); diese Variable ist bei Anfangswertproblemen meist die Zeit t.
Dies ist nicht bei allen Programmen so: Ihre KollegInnen, die eine der entsprechenden Matlab-Routinen verwenden, haben es da viel leichter: dort übernehmen die Matlab-Programme diese Aufgabe ...

ist Ihnen sicherlich zumindest dem Namen nach bekannt. Der französische Physiker Leon Foucault (1819-1868) demonstrierte mit diesem berühmten Experiment im Pariser Pantheon (1851) anschaulich die Erdrotation:



Eine 28 kg schwere Metallkugel hing an einem 67 m langen Draht, der in der Kuppel dieses Gebäudes befestigt war. Das so erhaltene - extrem gering gedämpfte - Pendel wurde in Schwingungen versetzt, wobei bekannt ist, dass die Schwingungsebene solcher Pendel äusserst raumstabil ist. Dennoch konnte unmittelbar beobachtet werden, dass die Schwingung im Lauf der Zeit seine Richtung zu ändern schien. In Wirklichkeit *bewegt sich die Erde unter dem Pendel* auf Grund der Erdrotation.

Am Äquator dreht sich die Schwingungsebene des Pendels überhaupt nicht; je weiter man sich vom Äquator entfernt, desto stärker ist die Drehung. An den geographischen Polen beträgt sie genau 360 Grad in 24 Stunden.

In dieser Aufgabe soll das Foucault'sche Experiment numerisch simuliert werden. Dazu muss das folgende System von zwei Bewegungsgleichungen des Pendelkörpers in x- und y-Richtung gelöst werden:

$$\ddot{x} = 2\Omega \sin(\lambda)\dot{y} - \frac{g}{L}x \tag{27}$$

und

$$\ddot{y} = -2\Omega \sin(\lambda)\dot{x} - \frac{g}{L}y\,. \tag{28}$$

Bedeutung der Grössen:

```
x(t)    x-Komponente der Pendelbewegung (m)
y(t)    y-Komponente der Pendelbewegung (m)

Omega   Winkelgeschwindigkeit der Erde um ihre Achse (rad/s):
        Zahlenwert = 2*pi/86400

lambda  geographische Breite (rad) des Ortes, wo das Experiment
        durchgefuehrt wird:
        Zahlenwert (fuer Paris): 49 pi/180

g       Erdbeschleunigung = 9.83 m/s^2

L       Laenge des Pendels = 67 m
```

Die Anfangsbedingungen dieses Problems lauten:

$$x(0) = \frac{L}{100} \qquad y(0) = 0 \qquad \dot{x}(0) = 0 \qquad \dot{y}(0) = 0 \tag{29}$$

Die Bewegung soll 12 Stunden (d.h. 43200 s) beobachtet werden.

Warum sind (27),(28) steife Differentialgleichungen?

Es gibt für dieses Problem keine exakte analytische Lösung, aber zumindest approximative Lösungen von Typus

$$x(t) \propto \mathrm{e}^{i\sqrt{g/L}t}\,\mathrm{e}^{-i\Omega\sin(\lambda)t}$$

und

$$y(t) \propto \mathrm{e}^{-i\sqrt{g/L}t}\,\mathrm{e}^{-i\Omega\sin(\lambda)t}\,.$$

Wie Sie sehen, stellt die Bewegung des Foucault-Pendels eine Kombination von zwei Vorgängen mit den Kreisfrequenzen (in s)

$$\sqrt{\frac{g}{L}} \approx 0.4 \qquad \text{und} \quad \Omega\sin(\lambda) \approx 5.5 \cdot 10^{-5}$$

dar: es liegt also ein Problem mit zwei sehr verschiedenen Zeitskalen vor, was für steife Differentialgleichungen typisch ist.

Der erste von Ihnen durchzuführende Rechenschritt besteht darin, das Gleichungssystem (27),(28) mit den Anfangswerten (29) *in ein äquivalentes System von 4 Differentialgleichungen erster Ordnung umzuwandeln.* Dieses System kann dann direkt von den entsprechenden C-Programmen bearbeitet werden.

Als Ergebnis jeder Auswertung erhalten Sie von dem C-Programm eine sehr grosse Zahl von Bahnpunkten der Bewegung des Pendelkörpers $\{x_i|y_i\}$ zu den (vom Programm selbständig gewählten) Zeitpunkten t_i.

Beurteilung der Qualität der numerischen Ergebnisse:

1. Tragen Sie alle erhaltenen $\{x_i|y_i\}$-Werte in ein Koordinatensystem ein. Wegen der Drehung der Erde und der konstanten Amplitude des Pendels während der gesamten Beobachtungszeit (Annahme: keine Dämpfung) sollten diese Punkte genau im Inneren bzw. auf der Peripherie eines Kreises mit dem Radius $x(0)$ liegen.

2. Ein weiteres Qualitätskriterium der numerischen Ergebnisse stellt die *Konstanz der Systemenergie* dar: wieder unter der Annahme, dass keinerlei Dämpfung stattfindet, muss die Summe aus der kinetischen plus potentiellen Pendelenergie zu jedem Zeitpunkt exakt gleich sein! Berechnen Sie daher für jeden Zeitpunkt t_i die Systemenergie

$$E = \frac{\dot{x}^2 + \dot{y}^2}{2} + g\left(L - \sqrt{L^2 - x^2 - y^2}\right) , \tag{30}$$

   uns stellen Sie die normierte Energie $E(t)/E(0)$ als Funktion der Zeit dar.

- Führen Sie diese Tests mit der Standard-C-Routine *runge_kutta_D.c* durch, die ich am Beginn dieses Übungsskriptums kurz erklärt habe. Da es sich dabei um ein *explizites* Rechenverfahren handelt, sollten Sie mit einer relativ kleinen Fehlerschranke beginnen, um brauchbare Resultate zu erzielen:

  ```
  eps = 1.e-5   .
  ```

  Lassen Sie danach das C-Programm noch zweimal laufen, und zwar mit den erhöhten Fehlerschranken

  ```
  eps = 1.e-4     bzw.    eps= 1.e-3 .
  ```

  Dokumentieren Sie, wie das Runge-Kutta-Programm mehr und mehr in Schwierigkeiten gerät.

- Nun kommt leider ein Problem: bei meinen Tests mit dem *Rosenbrock-Programm* (S. 16) musste ich feststellen, dass dieses Programm - das eigentlich auf steife Differentialgleichungen spezialisiert ist - mit dem Foucault-Problem noch mehr troubles hat als das Standard-Runge-Kutta-Programm.

  Die Ursache dafür ist mit noch nicht klar! Ich hoffe Ihnen bis zum Ende des Semesters sagen zu können, woran es liegt.

7. Ein Problem aus der Biologie: HIRES

Für die letzte Aufgabe zum Thema *stiff differential equations* habe ich Ihnen ein Problem mit realem naturwissenschaftlichen Hintergrund versprochen. Ich möchte Ihnen nun ein solches Problem vorstellen, und zwar nicht aus der Physik, sondern aus der Biologie.

Der Codename dieses Problems, HIRES, wurde von Hairer und Wanner (1996)[3] als Abkürzung von *High Irradiance RESponse* erstmals verwendet. Das Problem selbst wird allerdings schon viel länger diskutiert. Es gehört zum Gebiet der Pflanzen-Physiologie und wird in zahlreichen einschlägigen Publikationen behandelt, z.B. in einer Arbeit von E. Schäfer[4] mit dem Titel

A New Approach to explain the "High Irradiance Responses" of Photomorphogenesis on the Basis of Phytochrome.

Laienhaft übersetzt bedeutet das: die Gestaltbildung von Pflanzen unter dem Einfluß von Licht (Photomorphogenese), bei welcher Phytochrome eine Hauptrolle spielen, ist sehr stark von der Bestrahlungsstärke abhängig.

Phytochrome nennen die Biologen hochspezialisierte *Photorezeptor-Proteine*, die in Pflanzen, Algen, Bakterien und Pilzen vorkommen und z.B. das Grünwerden von Pflanzenteilen oder die Samenkeimung von Pflanzen unter Lichteinfluß steuern. Die Sache wird noch dadurch kompliziert, daß es zwei Formen von Phytrochrom gibt, nämlich die P_r-Form, die hauptsächlich auf hellrotes Licht anspricht ($\lambda \approx 660$ nm), und die P_{fr}-Form, die besonders mit dunkelrotem Licht wechselwirkt ($\lambda \approx 730$ nm)[5].

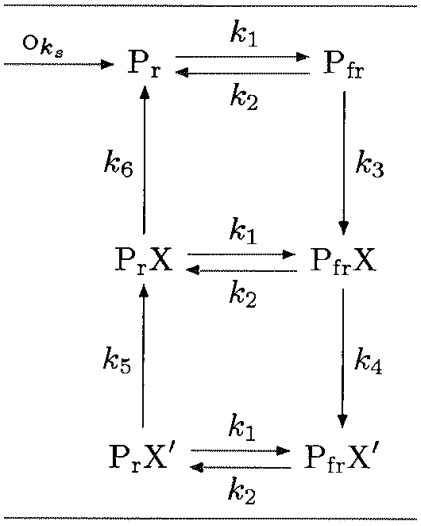
Diese beiden Konfigurationen P_r und P_{fr} gehen bei dem komplizierten Prozess, der sich in den Zellen abspielt, nicht nur mit bestimmten Wahrscheinlichkeiten (Übergangsraten) ineinander über, sondern können auch an verschiedenen Rezeptoren (X bzw. X' genannt) andocken. Auch diese komplizierten Moleküle können ineinander übergehen, manchmal in beide Richtungen, manchmal nur in eine Richtung.

Die Biologen und Biochemiker haben herausgefunden, daß das Reaktionsschema zwischen den 6 beteiligten Stoffen P_r, P_{fr}, P_rX, $P_{fr}X$, P_rX' und $P_{fr}X'$ wie folgt aussieht:

[3] E. Hairer and G. Wanner, *Solving Differential Equations II: Stiff and Differential-algebraic Problems*, Springer, 1996.

[4] E. Schäfer, J. of Math. Biology **2**, 41 (1975)

[5] r steht für engl. *red*, fr für engl. *far red*.



Für den Numeriker ist entscheidend, daß man dieses Schema ohne Probleme als System von 6 gewöhnlichen Differentialgleichungen hinschreiben kann, wie ich Ihnen nun zeigen werde.

Das obige Schema beschreibt die folgende Situation: die Substanz P_r wird in der Zelle mit einer konstanten Rate Ok_s erzeugt, und sie zerfällt ihrerseits mit einer Rate k_1. Andererseits gibt es zwei Prozesse, bei denen sich (1) die Substanz P_{fr} mit der Rate k_2 und (2) die Substanz P_rX mit der Rate k_6 in das P_r umwandelt. Die entsprechende *differentielle Bilanzgleichung* für das Phytochrom P_r kann demnach in der Form

$$\frac{\partial}{\partial t}P_r(t) = Ok_s - k_1P_r(t) + k_2P_{fr}(t) + k_6[P_rX](t)\,. \tag{31}$$

geschrieben werden und stellt die erste Differentialgleichung des HIRES-Problems dar: sie beschreibt die zeitliche Veränderung der Menge von *red*-Phytochrom $P_r(t)$.

Wenn Sie diese Interpretation verstanden haben, wird es Ihnen keine Schwierigkeiten bereiten, auch die entsprechenden Bilanzgleichungen für die übrigen 5 Beteiligten an diesem Zellprozess zu verstehen. Sie lauten (wobei ich die Zeitargumente bei den Funktionen weglasse):

$$\frac{\partial}{\partial t}P_{fr} = k_1P_r - k_2P_{fr} - k_3P_{fr}\,, \tag{32}$$

$$\frac{\partial}{\partial t}[P_rX] = -k_6[P_rX] - k_1[P_rX] + k_2[P_{fr}X] + k_5[P_rX']\,, \tag{33}$$

$$\frac{\partial}{\partial t}[P_{fr}X] = k_3P_{fr} + k_1[P_rX] - k_2[P_{fr}X] - k_4[P_{fr}X]\,, \tag{34}$$

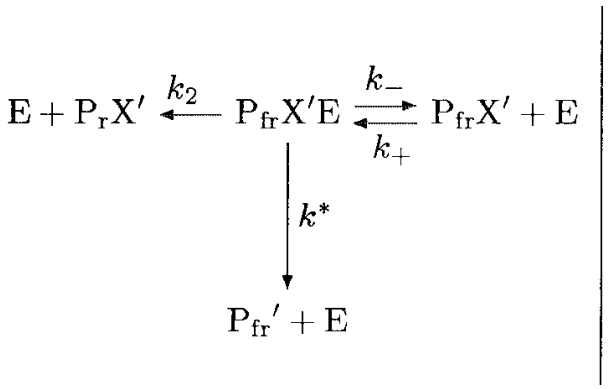
$$\frac{\partial}{\partial t}[P_rX'] = -k_5[P_rX'] - k_1[P_rX'] + k_2[P_{fr}X'] + k_2[P_{fr}X'E]\,, \tag{35}$$

$$\frac{\partial}{\partial t}[P_{fr}X'] = k_4[P_{fr}X] + k_1[P_rX'] - k_2[P_{fr}X'] + k_-[P_{fr}X'E] - k_+[P_{fr}X']E\,, \tag{36}$$

Der Zusammenhang zwischen diesen Differentialgeichungen und dem Reaktionsschema auf S. 20 sollte Ihnen nun klar sein.

Dies gilt aber nur für die "schwarzen Anteile" an den obigen Gleichungen. Wo kommen die "roten Teile" der Gleichungen (35) und (36) her?

Wie genauere Forschungen der Biochemiker ergeben haben, vollzieht sich die "Wechselwirkung" zwischen den Substanzen P_rX' und $P_{fr}X'$ nicht so einfach wie im obigen Schema beschrieben, sondern unter Mitwirkung eines Enzyms E. Dieses Enzym wirkt so wie im folgenden Reaktionsschema gezeigt:



Die richtige mathematische Umsetzung dieses Schemas ist etwas schwieriger als die bisherigen Rechnungen. Man nimmt an, daß in der Zelle eine bestimmte Menge des Enzyms E vorhanden ist, und daß dieser Stoff die oben skizzierten Prozesse ermöglicht (das Enzym wirkt als Katalysator): zum einen kommt es mit der Rate k_+ zu einer Anlagerung von $P_{fr}X'$ an das Enzym, und es bildet sich ein Molekül $P_{fr}X'E$. Die entsprechende Differentialgleichung lautet

$$\frac{\partial}{\partial t}[P_{fr}X'E] = k_+ \, [P_{fr}X'] * E \, .$$

Daß oben rechts das Produkt aus den Mengen von $P_{fr}X'$ und E steht und nicht etwa die Summe, ist darauf zurückzuführen, daß diese Anlagerung nur erfolgen kann, wenn *beide Partner existieren.*
Zusätzlich hat die Substanz $P_{fr}X'E$ auch noch drei Zerfallsmoden, und zwar mit den Raten k_2, k_- und k^*, sodaß sich insgesamt die Bilanzgleichung

$$\frac{\partial}{\partial t}[P_{fr}X'E] = k_+[P_{fr}X']E - (k_2 + k_- + k^*)[P_{fr}X'E] \qquad (37)$$

ergibt.

$P_{fr}X'E$ ist also die siebente Substanz, die am HIRES-Prozess teilnimmt; die achte ist das Enzym E, und dessen Bilanzgleichung ist ganz einfach: es gibt in der Zelle nur ein bestimmtes Quantum an diesem Stoff, d.h. jedes durch Anlagerung entstehende Molekül $P_{fr}X'E$ bedeutet ein Molekül E weniger. Es muß somit (als achte Differentialgleichung) gelten:

$$\frac{\partial}{\partial t}[E] = -\frac{\partial}{\partial t}[P_{fr}X'E] \, . \qquad (38)$$

Als letzter Diskussionspunkt bleiben nun noch die roten Teile der Differentialgleichungen (35) und (36). Auch diese Terme lassen sich aus dem obigen Reaktionsschema (mit dem Enzym E) ablesen. Sie ersehen daraus, daß das Molekül $P_{fr}X'E$ mit der Rate k_2 dissoziiert, wobei gleichzeitig eine Umwandlung des Phytochroms von der fr-Form in die r-Form erfolgt. Daraus resultiert in Glg. (35) der zusätzliche Term

$$k_2\,[P_{fr}X'E]\,.$$

Das Molekül $P_{fr}X'$ hat nun sogar zwei neue Wechselwirkungsprozesse: es kann einerseits (mit der Rate k_-) aus dem Zerfall eines $P_{fr}X'E$-Moleküls entstehen, oder sich andererseits (mit der Rate k_+) an ein Enzym-Molekül anlagern, *natürlich nur, wenn ein Enzym vorhanden ist.* Dies führt in Glg. (36) zu den zusätzlichen Termen

$$+k_-[P_{fr}X'E] - k_+[P_{fr}X']E\,.$$

Die Differentialgleichungen (31)-(38) stellen das zu lösende Problem HIRES dar.

Aufgabe 4

Nun also zu den Angaben für die 4. Aufgabe:
Achtung: für die folgenden Tests funktioniert das Rosenbrock-Programm einmwandfrei!

- Um das Arbeiten mit dem Rosenbrock-Programm auszuprobieren, ist es sinnvoll, einen einfachen Test durchzuführen. Wenden Sie deshalb dieses Programm zuerst auf das *Gear-Problem* (s. S. 10) an.

  Machen Sie die Auswertung für $t \in [0, 2.5]$ mit einer Fehler-Toleranz $eps = 0.0001$, und präsentieren Sie die Ergebnisse in Form eines Diagramms, in dem Sie sowohl die exakten Kurven von $y^{(1)}(t)$ und $y^{(2)}(t)$ als auch (als Punktmengen) die vom Rosenbrock-Programm errechneten Näherungswerte zeigen.

- Erst wenn mit diesem Test alles geklappt hat, können Sie darangehen, das etwas aufwändigere Problem HIRES numerisch zu lösen, das aus den oben definierten acht - teilweise nicht-linearen - Differentialgleichungen besteht.

  Mit den Zuordnungen

$$P_r \to y_1 \quad P_{fr} \to y_2 \quad P_rX \to y_3 \quad P_{fr}X \to y_4$$

$$P_rX' \to y_5 \quad P_{fr}X' \to y_6 \quad P_{fr}X'E \to y_7 \quad E \to y_8$$

sowie den aus der Literatur entnommenen Parametern

$k_1 = 1.71$	$k_3 = 8.32$	$k_5 = 0.035$	$k_+ = 280$
$k_2 = 0.43$	$k_4 = 0.69$	$k_6 = 8.32$	$k_- = 0.69$
$k^* = 0.69$	$Ok_s = 0.0007$		

ergibt sich daraus das folgende Differentialgleichungssystem:

$$\begin{aligned}
y'^{(1)}(t) &= -1.71y^{(1)} + 0.43y^{(2)} + 8.32y^{(3)} + 0.0007 \\
y'^{(2)}(t) &= 1.71y^{(1)} - 8.75y^{(2)} \\
y'^{(3)}(t) &= -10.03y^{(3)} + 0.43y^{(4)} + 0.035y^{(5)} \\
y'^{(4)}(t) &= 8.32y^{(2)} + 1.71y^{(3)} - 1.12y^{(4)} \\
y'^{(5)}(t) &= -1.745y^{(5)} + 0.43(y^{(6)} + y^{(7)}) \\
y'^{(6)}(t) &= -280y^{(6)}y^{(8)} + 0.69y^{(4)} + 1.71y^{(5)} - 0.43y^{(6)} + 0.69y^{(7)} \\
y'^{(7)}(t) &= 280y^{(6)}y^{(8)} - 1.81y^{(7)} \\
y'^{(8)}(t) &= -y'^{(7)}
\end{aligned} \tag{39}$$

Die Anfangswerte der 8 Funktionen lauten

$$y^{(1)}(0) = 1 \qquad y^{(2)}(0) \cdots y^{(7)}(0) = 0 \qquad y^{(8)}(0) = 0.0057 \quad , \tag{40}$$

und die numerische Auswertung soll über den Zeitbereich $0 \leq t \leq 400$ erfolgen. Als Fehlergrenze nehmen Sie $eps = 0.0005$ und als Anfangsschrittweite $hstart = 0.001$.

Erstellen Sie 8 Diagramme, in denen Sie die einzelnen numerisch ermittelten Lösungsfunktionen darstellen, und zwar die Funktionen $y^{(5)}$ und $y^{(6)}$ im Zeitbereich $0 \leq t \leq 400$, und alle anderen Funktionen im Zeitbereich $0 \leq t \leq 5$.

- Dieses Beispiel gibt noch einmal die Gelegenheit zu zeigen, wie überlegen das *implizite Runge-Kutta-Verfahren* nach Rosenbrock dem "normalen", *expliziten Runge-Kutta-Verfahren* ist, wenn es um die numerische Auswertung von *stiff differential equations* geht.

  Es macht Ihnen keine Mühe, das an Hand des HIRES-Problems zu verifizieren. Sie brauchen nur das von mir am Anfang dieses Übungsskriptums vorgestellte Runge-Kutta-Programm hernehmen und die Routine FDER, in der Sie für's Rosenbrock-Programm die Funktionen (39) definiert haben, in DERIVS umbenennen. Die Routine JACOBN benötigen Sie beim Runge-Kutta-Programm natürlich nicht. Dann ersetzen Sie den Aufruf

```
#include "rosenbrock_D.c"
```

durch

```
#include "runge_kutta_D.c"
```

beziehungsweise (in Ihrem *main*-Programm) die Aufrufzeile

```
odesti(ystart,nvar,t0,tmax,eps,h1,hmin,nstmax,&nbad,&nwerte,tfeld,yfeld);
```

durch

```
odeint(ystart,nvar,t0,tmax,eps,h1,hmin,nstmax,&nwerte,tfeld,yfeld);
```

und die Sache ist erledigt.

Vergleichen Sie, wieviele Stützpunkte das Runge-Kutta-Programm im Vergleich zum Rosenbrock-Programm für dieselbe Genauigkeit und dieselbe Anfangs-Schrittweite benötigt.