Zeitliche Dynamik Fortsetzung

Zeitabhängige Probleme können für unsere Beispiele in zwei Fälle klassifiziert werden (eigentlich Spezialfälle)

Lineare Probleme:

1) $\underline{\dot{u}} = \underline{\underline{A}} \cdot \underline{u}$ (indem $\underline{u} = \begin{pmatrix} \underline{v} \\ \underline{p} \end{pmatrix}$, keinen anderen Problem $\underline{\dot{v}} = \underline{\underline{B}} \cdot \underline{v} + \underline{f}$)

Matrix, Vektor

2) mit zeitabhängiger Matrix $\underline{\underline{A}}(t)$

$\underline{\dot{u}} = \underline{\underline{A}}(t) \cdot \underline{u}$ dabei kann es effizienter sein im Falle z. B. Beispiel:

$\underline{\underline{A}}(t) = \underline{\underline{A}}_{const} + F(t)\, \underline{\underline{A}}_{dyn}$

3) $\underline{\dot{u}} = f(\underline{u}, t)$ Allgemeiner Form, deshalb schwierig in Vektorisierte Fassung zu bringen.

Beispiele für die Umsetzung

1. Beispiel zeitabhängige Schrödingergl

$$i\hbar\, \partial_t \psi(\underline{r}, t) = \left(-\frac{\hbar^2}{2m} \Delta_r + V_{coul}(\underline{r})\right) \psi(\underline{r}, t) + f(\underline{r})\, dE(t)$$

Wir diskretisieren

$\psi(\underline{r}, t)$ z. B. auf Gitter $\longrightarrow \underline{\psi}$

$\left(-\frac{\hbar^2}{2m} \Delta_r + V_{coul}(\underline{r})\right) \longrightarrow \underline{\underline{A}}_{const}$

$$\underline{v} = \begin{pmatrix} \psi \\ 1 \end{pmatrix}$$

$$\partial_t \underline{v} = \begin{pmatrix} \underline{\underline{A}}_{const} & 0 \\ 0 & 0 \end{pmatrix} \cdot \underline{v} + E(t) \cdot \begin{pmatrix} 0 & 0 \\ 0 & \underline{f} \end{pmatrix} \cdot \underline{v}$$

$\Rightarrow$ Fall 2. (sehr ähnlich)
bzw. Fall 1 falls $\underline{E}(t) = 0$

2. Beispiel Bloch-Gleichungen

$$\partial_t \rho_{12} = -i(\varepsilon_1 - \varepsilon_2)\rho_{12} - i\underline{E} d_{21}(\rho_{22} - \rho_{11})$$

$$\partial_t \rho_{11}/\rho_{22} = \mp 4\,\mathrm{Im}(\underline{E} \cdot d_{21} \rho_{12})$$

$$\underline{V} = (\rho_{12}, \rho_{21}, \rho_{11}, \rho_{22})$$

(komplex konjugierte Größen nicht so leicht zu vektorisieren)

$$\partial_t \underline{V} = \begin{pmatrix} -i(\varepsilon_1 - \varepsilon_2) & & -i\underline{E} d_{21} & i\underline{E} d_{21} \\ & \pm i(\varepsilon_1 - \varepsilon_2) & +i\underline{E} d_{12} & -i\underline{E} d_{12} \\ -i\underline{E} d_{21} & i\underline{E} d_{21} & 0 & \\ i\underline{E} d_{21} & -i\underline{E} d_{12} & & 0 \end{pmatrix}$$

$\Rightarrow \partial_t \underline{V} = \underline{\underline{A}} \cdot \underline{V}$ $\Rightarrow$ wieder Fall (1)

Achtung hier gibt es ein Problem $\varepsilon_2 - \varepsilon_1 \gg \underline{E} d_{21}$
daher wird der Zeitsolver sehr kleine Schritte benötigen, um die Oszillation aufzulösen!

Trick: $\underline{V} \to \underline{\tilde{V}} = (\tilde{\rho}_{12}, \tilde{\rho}_{21}, \rho_{11}, \rho_{22})$

mit $\tilde{\rho}_{12} = \rho_{12} e^{i\omega t}$, Rotating Frame

$$\Rightarrow \partial_t \tilde{\rho}_{12} = -i\Delta \rho_{12} - i\tilde{E} d_{21}(\rho_{22} - \rho_{11})$$

mit $\Delta = (\varepsilon_1 - \varepsilon_2) + \omega_L$

Andeutet die vektorisierte Form des Problems allerdings nicht.

3. Beispiel Halbleiter-Bloch-Gleichungen

Bewegungsgl.:

$$\partial_t P_{ll} = -\frac{i}{\hbar}\underbrace{(\varepsilon_{v,ll} - \varepsilon_{c,ll} + i\gamma)}_{\text{linear}} P_{ll} + \frac{i}{\hbar}\left(\sum_{q\neq 0} V(q)\,(f_{c,k+q} - f_{v,k+q})\,P_q\right)$$

Produkt aus zwei Größen in $\underline{v}$

$\Leftarrow\!\!/\!\!\Rightarrow$ also nicht $\underline{\underline{A}}\cdot\underline{v}$ darstellbar

allgemeiner Fall 3, gesucht $f(\underline{v},t)$

$\Rightarrow$ per Hand programmieren.

z.B. $\sum_{q=0} V(q)\,(f_{c,k+q} - f_{\dots})$

Filtry Hedry

Schwierig bei Parallelisierung, nicht alle $f_{\dots}$ sind auf dem richtigen Prozessor.

Nächster Schritt: Algorithmen finden für den 3. Fall.

Bemerkung: Bei PDGL gibt es noch weitere Verfahren.

Lösungsverfahren.

Es gibt viele Verfahren, hier diskutieren wir die beliebte Familie der Runge-Kutta Algorithmen (Achtung: Hier, nicht der spezielle RK 4. Ordnung.)

Allgemein ist unser Problem

$$\underline{u} = \underline{f}(\underline{u}, t)$$

Wir diskretisieren die Zeit:

$$a = t_0 < t_1 < \dots < t_n = b$$

$\underline{u}(t_0)$ sei gegeben.

Gesucht sind jetzt Runge-Kutta-Formeln der Form

mit $\underline{u}(t_n) = \underline{u}_n$

$$\underline{u}_{n+1} = \underline{u}_n + h \sum_{i=1}^{s} b_i \underline{k}_i$$

h ist die Schrittweite $h = t_{n+1} - t_n$

$$\underline{k}_1 = \underline{f}(t_n, \underline{u}_n)$$

$$\underline{k}_i = \underline{f}\left(t_n + c_i h, \underline{u}_n + h \sum_{j=1}^{i-1} a_{ij} \underline{k}_j\right)$$

$i = 2, \dots, s$

Das Schema verwendet die Ableitung an verschiedenen Punkten für die neue Approximation

Abschätzung des Fehlers

$$\underline{u}(t_n + h) - \underline{u}_{n+1} = \sum_{i=1}^{\infty} h^i \sum_{j=1}^{n_i} \alpha_j^{(i)} D_j^{(i)}$$

($n_i = \{1, 1, 2, 4, 9, 20, \dots\}$)

Taylor-Entwicklung für glatte Fkt.

Koeffizienten, die hängen von a_{ij} und b_i

Summe von partiellen Ableitungen.

Die Ordnung kann jetzt abgeschätzt werden über die Koeffizienten.

Sind alle $\alpha_j^{(i)} = 0$ für $i \le n$,

dann ist es ein Schema n-Ordnung.

Ein Verfahren wird über die Koeffizienten

c_i, b_i und a_{ij} gesetzt.

Für jede Ordnung gibt es verschiedene Koeffizienten kombinierbar, die andere Vor- und Nachteile bzgl. Anzahl an Rechenschritte, Stabilität, Schrittweite, etc. haben.

Beispiel Bogacki - Shampine (Appl. Math. Lett 2(4), 321-325 (1988)

c_1 0		$= a_{21}$
c_2 $\frac{1}{2}$	$\frac{1}{2}$	
c_3 $\frac{3}{4}$	0 a_{31}	$\frac{3}{4}$ a_{32}
	$\frac{2}{9}$	$\frac{1}{3}$ $\frac{4}{9}$
	b_1	$\dots$ b_3

Verfahren 3. Ordnung.

Verfahren 3. Ordnung haben ein lokalen Fehler von h^3. Aufsummiert ist es h^2 (1. Ordnung Verfahren Euler mit h, sehr viel schneller $\Rightarrow$ hohe Diskretisierung.)

Vorteil von Bogacki-Shampine es rechnet auch Verfahren 2. Ordnung aus!

$\frac{7}{24}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$

Das erlaubt den Fehler abzuschätzen (gleich ja anders mit h (Potenz)) und Schrittweite h beim Rechnen anzupassen!
(embedded Runge Kutta)

Der Fehler ist dann:

$$e_{n+1} = u_{n+1}^{(\text{Methode 1})} - u_{n+1}^{(\text{Methode 2})} = h \sum_{i=1}^{5} (b_i^{(\text{Methode 1})} - b_i^{(\text{Methode 2})}) k_i$$

Das ist $O(h^p)$

Der Fehler muss dann in bestimmter Toleranz sein. Falls nicht wird die Schrittweite verringert.

Beispiele

Euler

0	0
	1

Heun Method

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

Original Runge Kutta

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

Embedded Methoden

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

Heun und Euler

Fehlberg RK1(2)

0			
$\frac{1}{2}$	$\frac{1}{2}$		
1	$\frac{1}{256}$	$\frac{255}{256}$	
	$\frac{1}{256}$	$\frac{255}{256}$	0
	$\frac{1}{512}$	$\frac{255}{256}$	$\frac{1}{512}$