Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Christian Engwer, MSc Marcel Koch

11.11.2019

Übung zur Vorlesung

Wissenschaftliches Rechnen

WS 2019/20 — Blatt 5

Abgabe: 25.11.2019, 10:00 Uhr
Code zusätzlich per e-mail an `marcel.koch@uni-muenster.de`

Achtung: Achten Sie darauf, Ihre Programme ordentlich zu formatieren und gut zu kommentieren. Die Form wird mit in die Bewertung eingehen.

Die Programmieraufgabe befasst sich mit der numerischen Lösung von Reaktions-Diffusions-Systemen. Dazu wird die eigentliche Implementierung in mehrere Schritte aufgeteilt. Zunächst wird das Newton-Verfahren zur Nullstellen Bestimmung implementiert und darauf aufbauend das Theta-Verfahren zur Zeitdiskretisierung. Mit diesen beiden Bausteinen lässt sich dann ein Löser für Reaktions-Diffusions-Systeme umsetzen.

Bemerkung 1 (Gleichungslöser)

Im praktischen Teil dieses Aufgabenblattes geht es um die Lösung nichtlinearer Gleichungssysteme in $\mathbb{R}^n$. Diese treten zum Beispiel bei impliziten Zeitschrittverfahren auf. Das Lösen eines nichtlinearen Gleichungssystems lässt sich formulieren als die Bestimmung der Nullstelle einer Funktion $r : \mathbb{R}^n \to \mathbb{R}^n$. Um verschiedene lineare oder nichtlineare Gleichungslöser auswählen zu können (Newton-Verfahren, CG-Verfahren, Gauß-Seidel-Verfahren, ...), führen wir folgendes Interface ein:

```
template <typename Vector, typename Matrix>
class Solver{
public:
  using VectorType = Vector;
  using MatrixType = Matrix;
  using FunctionType = DifferentiableFunction<VectorType,VectorType,MatrixType>;

  virtual VectorType apply (const FunctionType& r, const VectorType& z) const = 0;
};
```

Zur Repräsentation von Funktionen wie r verwendet dieses wiederum folgendes Interface für differenzierbare Funktionen, die nicht von der Zeit abhängen:

```
template <typename Domain, typename Range, typename JacobianRange>
class DifferentiableFunction{
public:
  using DomainType = typename DifferentiableFunction::DomainType;
  using RangeType = typename DifferentiableFunction::RangeType;
  using JacobianRangeType = JacobianRange;

  virtual DomainType operator()(const DomainType& x) const = 0;
  virtual JacobianRangeType evaluateJacobian (const DomainType& x) const = 0;
};
```

Definition 1 (Newton-Verfahren)

Sei $r : \mathbb{R}^n \to \mathbb{R}^n$ stetig differenzierbar. Das Newton-Verfahren liefert über die Fixpunktiteration

$$z^{l+1} := z^l - \left(J^k(z^l)\right)^{-1} r(z^l), \quad l = 0, 1, 2, \ldots \tag{1}$$

zu einem geeignet gewählten Startwert $z^0 \in \mathbb{R}^n$ eine Folge $(z^l)_{l \in \mathbb{N}}$, die gegen eine Nullstelle von r konvergiert. Genauer kann man zeigen, dass es zu jeder Nullstelle $z \in \mathbb{R}^n$ eine Umgebung $D \subset \mathbb{R}^n$ gibt, so dass das Newton-Verfahren für alle Startwerte $z^0 \in D$ quadratisch gegen z konvergiert, falls r in einer Umgebung von z Lipschitz-stetig ist und in z eine invertierbare Jakobi-Matrix $J^k(z)$ besitzt.

Aufgabe 1 (Newton-Verfahren) (4 Punkte)

Implementieren Sie das Newton-Verfahren in C++. Schreiben Sie dazu ein Klassentemplate `NewtonSolver`, welches das Interface `Solver` erfüllt.

- Das Verfahren soll für verschiedene Datentypen anwendbar sein, mit denen sich Vektoren in $\mathbb{R}^n$ und Matrizen in $\mathbb{R}^{n \times n}$ repräsentieren lassen. Zu diesem Zweck soll `NewtonSolver` zwei Template-Parameter `Vector` und `Matrix` besitzen. Setzen Sie voraus, dass `Vector` eine Methode `operator[]` für den indexbasierten Zugriff auf die Elemente des Vektors und eine Methode `size` zur Verfügung stellt (wie z.B. `std::array`). Setzen Sie ferner voraus, dass auf die Elemente und Größe einer Matrix vom Typ `Matrix` durch die doppelte Anwendung dieser Methoden zugegriffen werden kann, d.h. dass die Matrix als Vektor von Zeilenvektoren repräsentiert wird.
- Abbruchkriterium: Die Fixpunktiteration (1) soll so lange durchgeführt werden, bis die Änderungsrate $\|z^{l+1} - z^l\|$ eine zu übergebende Schranke `eps` unterschreitet.
- Beschränken Sie sich auf den Fall $n \in \{1, 2, 3\}$, für den sich die Invertierung einer regulären $n \times n$-Matrix explizit hinschreiben lässt (siehe Lineare Algebra I); behandeln Sie den Fall $n > 3$ mit einer Exception oder fangen Sie ihn mit einer Assertion ab.
- Testen Sie Ihre Implementierung in dem Sie vom Interface `DifferentiableFunction` eine Funktion ableiten, die der Funktion

  $$f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{\cos(x_1) - \sin(x_2)}{4} - x_1 \\ \frac{\cos(x_1) - 2\sin(x_2)}{4} - x_2 \end{pmatrix}$$

  entspricht und bestimmen Sie deren Nullstelle. Als Startvektor wählen Sie $x_0 = (0, 0)^T$.

Definition 2 (Theta-Verfahren)

Gegeben sei ein AWP 1. Ordnung

$$\begin{aligned} y' &= f(t, y), \qquad \text{auf} \quad I := [t^0, T], \qquad t^0, T \in \mathbb{R}_0^+, \\ y(t^0) &= y^0, \qquad\qquad\qquad\qquad\qquad\quad y^0 \in \mathbb{R}^n, \end{aligned}$$

mit $f : I \times \mathbb{R}^n \to \mathbb{R}^n$ stetig, $n \in \mathbb{N}$. Auf I sei $I_{\Delta t} := \{t^k = t^0 + k\Delta t \mid k = 0, 1, 2, \ldots \ \wedge \ t^k \leq T\}$ ein gewähltes Gitter mit zugehöriger Schrittweite $\Delta t \in \mathbb{R}^+$. Dann liefert die Iterationsvorschrift

$$y^{k+1} = y^k + \Delta t\Big((1-\theta)f(t^k, y^k) + \theta f(t^{k+1}, y^{k+1})\Big), \quad k = 0, 1, 2, \ldots \tag{2}$$

für festes $\theta \in [0, 1]$ Approximationen $y^k \approx y(t^k)$ der Lösung des AWPs. Für autonome gewöhnliche Differentialgleichungen gilt das zum impliziten Euler-Verfahren gesagte.

Bemerkung 2 (Theta-Verfahren)

Das explizite Euler-Verfahren ($\theta = 0$) ist das einzige explizite Theta-Verfahren. Für $\theta \neq 0$ ergeben sich implizite Verfahren. Die Wahl $\theta = 1$ liefert das implizite Euler-Verfahren und $\theta = 0.5$ das implizite Analogon zum Heun-Verfahren von Blatt 2, die sogenannte *Crank-Nicolson-Methode*.
Zur Anwendung eines impliziten Verfahrens sollen Sie in jedem Zeitschritt ihren Newton-Solver verwenden. Dazu wird das Lösen eines Gleichungssystems äquivalent formuliert als die Bestimmung der Nullstelle einer Funktion $r^k : \mathbb{R}^n \to \mathbb{R}^n$. Beim impliziten Euler-Verfahren z.B. ist $r^k(z) := z - y^k - \Delta t \cdot f(t^k + \Delta t, z)$.

Aufgabe 2 (Theta-Verfahren) (4 Punkte)

Implementieren Sie in C++ ein Klassentemplate `ThetaScheme` zur Durchführung des Theta-Verfahrens für den allgemeinen Fall eines nicht-autonomen AWPs 1. Ordnung.

- Das Verfahren soll für verschiedene Datentypen anwendbar sein, mit denen sich Vektoren in $\mathbb{R}^n$, Matrizen in $\mathbb{R}^{n\times n}$ und Zeiten in $\mathbb{R}_0^+$ repräsentieren lassen. Zu diesem Zweck soll `ThetaScheme` drei Template-Parameter `VectorType`, `MatrixType` und `TimeType` besitzen. Stellen Sie an `VectorType` und `MatrixType` die gleichen Vorraussetzungen wie in Aufgabe 1.

- Der Konstruktor des Klassentemplates soll einen Gleichungslöser als Objekt einer Klasse erhalten, die das Interface `Solver` aus Bemerkung 1 erfüllt. Ferner soll er die Funktion $f : I \times \mathbb{R}^n \to \mathbb{R}^n$ als Objekt einer Klasse erhalten, die folgendes Interface für zeitabhängige, örtlich differenzierbare Funktionen geeignet erfüllt:

```
template <typename Domain, typename Range, typename JacobianRange,
          typename Time = double>
class DifferentiableTimeFunction{
public:
  using DomainType = Domain;
  using RangeType = Range;
  using TimeType = Time;
  using JacobianRangeType = JacobianRange;

  virtual RangeType operator()(const TimeType& t, const DomainType& x) const = 0;

  virtual JacobianRangeType evaluateJacobian (const TimeType& t,
                const DomainType& x) const = 0;
};
```

- Mit der Methode

```
VectorType apply (const TimeType& t, const TimeType& dt,
                  const VectorType& y_old) const
```

sollen die durch die Iterationsvorschrift festgelegten Schritte des Verfahrens ausgeführt werden können. Dabei bezeichnen `t` sowie `dt` den Zeitpunkt t_k sowie die Schrittweite Δt des aktuellen Zeitschritts und `y_old` entspricht der Approximationen y_k zum alten Zeitpunkt.

- Überlegen Sie sich, wie r^k beim Theta-Verfahren aussieht und wie die Jakobi-Matrix von r^k mit der Jakobi-Matrix von f zusammenhängt.
- Verwenden Sie zur Bestimmung der Approximation y_{k+1} zum neuen Zeitpunkt die Approximation y_k zum alten Zeitpunkt als initial guess für den Gleichungslöser.
- Im Falle $\theta = 0$ soll aus Effizienzgründen der Gleichungslöser ignoriert werden und das explizite Euler-Verfahren direkt durchgeführt werden.
- Für den Test Ihrer Implementierung betrachten Sie den diffusionslosen Spezialfall des Schnakenberg-Modells

$$\begin{aligned} a' &= c_a - r_a a + s a^2 b \\ b' &= c_b - s a^2 b. \end{aligned}$$

Leiten Sie vom Interface `DifferentiableTimeFunction` eine entsprechende Klasse für die rechte Seite ab und plotten Sie die zeitliche Entwicklung der Konzentrationen a und b einmal für das explizite und einmal für das implizite Euler-Verfahren. Verwenden Sie dabei testweise die Parameter $s = r_b = c_a = c_b = 1, r_a = 2$, die Anfangswerte $a(0) = 0.5$ sowie $b(0) = 5$, die maximale Zeit $T = 50$ und die Schrittweite $\Delta t = 0.001$.

Aufgabe 3 (Turing-Modell, Computermodell mit Operator-Splitting) (8 Punkte)

Betrachten Sie das Reaktions-Diffusions-System

$$\left.\begin{aligned} \partial_t a &= D_a \Delta a + g_1(a,b) \\ \partial_t b &= D_b \Delta b + g_2(a,b) \end{aligned}\right\} \quad \text{in} \quad \Omega \subset \mathbb{R}^d \times [0,T] \tag{3a}$$

mit Neumann-Randbedingungen

$$\nabla a \cdot n = 0, \quad \nabla b \cdot n = 0 \qquad \text{auf} \quad \partial\Omega \times [0,T] \tag{3b}$$

und Anfangsbedingungen

$$a(\cdot,0) = a_0, \quad b(\cdot,0) = b_0 \qquad \text{in} \quad \Omega, \tag{3c}$$

welches Sie in der Vorlesung als Turing-Modell kennengelernt haben. Um dieses zu simulieren wollen wir ein Computermodell für (3) herleiten. Dabei beschränken wir uns auf ein rechteckiges Gebiet $\Omega = [0,L]^2 \subset \mathbb{R}^2$, welches durch ein kartesisches Gitter[1] partitioniert ist.

[1] Ein kartesisches Gitter ist ein gleichmäßiges Gitter mit uniformer Kantenlänge, d.h. es besteht aus rechteckigen Zellen mit achsenparallelen Kanten, die alle gleich lang sind.

(a) Leiten Sie mit der Linienmethode und dem zellzentrierten Finite-Volumen-Verfahren eine Semidiskretisierung im Ort her. Beide Methoden sind aus der Vorlesung bekannt. Dort haben Sie darüber hinaus das Strang-Splitting kennengelernt. Verwenden Sie dieses, um den Diffusions- und den Reaktionsanteil in der Semidiskretisierung voneinander zu splitten.

(b) Diskretisieren wir die semidiskreten Probleme nun in der Zeit, erhalten wir ein Computermodell. Für das Diffusionsproblem wollen wir das explizite Euler-Verfahren verwenden und für das Reaktionsproblem das implizite Euler-Verfahren. Implementieren Sie das resultierende Computermodell in C++.

- Auf der Vorlesungshomepage finden Sie Code zur Generierung von Anfangswerten, zur Generierung von Gitterinformationen für Finite-Volumen-Verfahren auf kartesischen Gittern und für die Datenausgabe im VTK Dateiformat ($\rightarrow$ `Paraview`).
- Verwenden Sie die Implementierung des Theta-Verfahrens. Wählen Sie dabei als Template-Parameter `VectorType`, `MatrixType` und `TimeType` geeignete Datentypen. Benutzen Sie als Gleichungssystemslöser für das implizite Euler-Verfahren die Implementierung des Newton-Verfahrens.
- Wählen Sie die Zeitschrittweite geeignet. Beachten Sie die CFL-Bedingung beim expliziten Euler-Verfahren, die eine Beschränkung der Zeitschrittweite mit sich bringt.

(c) Verwenden Sie die Anfangswerte a_0 und b_0 aus dem zur Verfügung gestellten Code und testen Sie Ihre Implementierung an dem konkreten Modell

$$
\begin{aligned}
&g_1(a,b) := 1/\varepsilon_0 \left(w_0(b)\, a + w_1(a)\, b - a^2\right), && w_0(b) := (1.0 - mb)/(1.0 - mb + \varepsilon_1),\\
&g_2(a,b) := w_0(b)\, a - b, && w_1(a) := p(q-a)/(q+a),
\end{aligned}
$$

$$
\text{mit} \qquad
\begin{aligned}
&D_a = 1.0, && D_b = 10.0, && \varepsilon_0 = 2.2, && \varepsilon_1 = 0.02,\\
&q = 0.0002, && p = 1.1, && m = 0.0007.
\end{aligned}
$$

Dieses Modell beschreibt chemische Experimente für die Belousov-Zhabotinsky Reaktion, die in [Bánsági et al., 2011][2] präsentiert werden. Die Experimente führen zu einer (eigentlich dreidimensionalen) Musterbildung, welche mit Hilfe eines Tomographen beobachtet werden kann.

[2] Bánsági, Tamás, Vladimir K. Vanag, and Irving R. Epstein. "Tomography of reaction-diffusion microemulsions reveals three-dimensional Turing patterns." Science 331.6022 (2011): 1309-1312.