Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. M. Ohlberger, Martin Drohmann, Stefan Girke

10.Mai 2012

Arbeitsblatt zum Praktikum

Scientific Computing

SS 2012 — Blatt 2

Abgabe: 22.Mai 2012 per Email

Ziel dieses Aufgabenblattes ist es in kurzer Zeit in die objekt-orientierte Programmierung mit Matlab einzusteigen. Einige Programmteile der RB Software „RBmatlab“ sind objekt-orientiert und ein grundlegendes Verständnis der Begrifflichkeit ist daher für die weiteren Praktikumssitzungen wichtig. Außerdem vereinfacht die objekt–orientierte Programmierung das Austauschen von Code–Teilen und somit später die Zusammenarbeit / Arbeitsteilung mit anderen Programmierern. Aus diesem Grund enthalten die meisten modernen Programmiersprachen, wie z.B. C++ objekt-orientierte Sprachelemente.

Anwesenheits-Aufgabe 2.1
Erweitern Sie das M-File `exercise2_1_model.m` um eine Methode `plot_detailed_data`, welche die RB Funktionen, die mit der Methode `gen_detailed_data` generiert wurden, visualisiert. Lesen Sie sich hierzu zunächst den Hilfe-Text zur globalen Funktion `plot_detailed_data` durch, und passen Sie die Argumente und Rückgabewerte ihrer eigenen Implementierung an die der globalen Funktion an.

Aufgabe 2.2 (Wiederholungsfragen)
Bitte beantworten Sie kurz die folgenden Fragen:

- Erklären Sie die Begriffe „Objekt“, „Klasse“, „Vererbung“.
- Erklären Sie den Unterschied zwischen einer statischen und einer dynamischen Methode.
- Erklären Sie, wofür abstrakte Klassen nützlich sind.

Aufgabe 2.3 (Auto–Beispiel)
Betrachten Sie die im Praktikum vorgestellten Klassen, die vom Interface `IAuto` abgeleitet wurden.
Schreiben Sie eine weitere Klasse für ihr Lieblings–Auto, und dazu ein kleines Skript, welches eine Instanz Ihrer eigenen Auto–Klasse erzeugt, und mindestens die Methode `fahren` auf diese Instanz anwendet.

Aufgabe 2.4 (Wiederverwendung eines Basis–Generierungs–Algorithmus)
Stellen Sie sich folgende Situation vor: Jemand hat einen Basis–Generierungs–Algorithmus geschrieben, und Sie möchten Ihr parametrisiertes detailliertes Modell aus Aufgabe 0.2 bzw. 1.1 mit diesem Algorithmus testen. Ihnen wurde mitgeteilt, dass sie zur Verwendung des Algorithmus' eine Klasse implementieren müssen, die das folgenden Interface erfüllen muss: `Exercise2.IDetailedModel`.

(a) Implementieren Sie diese Klasse. Dafür können Sie die Vorlage im Package `Exercise2` verwenden. Gehen Sie folgendermaßen vor:

- Verwenden Sie den Befehl `show_abstract_members`, um die zu implementierenden Methoden und Eigenschaften anzuzeigen.
- Lesen Sie die Beschreibung des Hilfe–Textes für die zu implementierenden Methoden und Eigenschaften, entweder mit den Matlab-Befehlen `help` oder `doc` z.B. durch Eingabe von `help IDetailedModel.mu_names` im Matlab-Fenster, oder mit dem Programm `htdoc`, z.B. durch `htdoc('IDetailedModel')`.
- Implementieren sie die fehlenden Methoden und Eigenschaften.
- Testen Sie die implementierten Klassen mit Hilfe des Skriptes `exercise2.m`

(b) Die Klassen `IReducedData` und `IReducedModel` haben eine abstrakte `copy` Methode. Diese ist in den Vorlagen bereits implementiert. Versuchen Sie heraus zu finden, was die Aufgabe dieser Methode ist. Hierzu müssen Sie die Funktionalität der Klasse `handle` (Hilfe mit `doc handle`) verstehen, von der alle Interface–Klassen abgeleitet sind.

(c*) Falls Sie zusätzliche Methoden und Eigenschaften in Ihren Klassen angelegt haben, machen Sie sich Gedanken über die Kapselung dieser Methoden, und fügen Sie entsprechende Zugriffsberechtigungen hinzu.

(d*) Überlegen Sie, welche der Klassen und Methoden Ihrer Meinung nach so komplex sind, dass die Funktion sich auf den ersten Blick nicht sofort erschließt und dokumentieren Sie diese. Überprüfen Sie das Ergebnis mit den Matlab-Befehlen `help` oder `doc`.

Aufgabe 2.4 (Parabolische Differentialgleichungen)
In dieser Aufgabe soll die Lösung einer zeitabhängigen Wärmeleitungsgleichung approximiert werden. Dazu betrachten wir folgendes Problem:
Sei $\Omega = [0,1] \subseteq \mathbb{R}^1$ und $T \in \mathbb{R}$ mit $T > 0$. Gesucht ist die Lösung $u : \Omega \times (0,T) \to \mathbb{R}$ der zeitabhängigen Wärmeleitungsgleichung

$$\partial_t u(x,t) - \partial_{xx} u(x,t) = 0 \qquad \forall (x,t) \in \Omega \times (0,T), \tag{1}$$

die folgende Randbedingungen erfüllt

$$u(x,t) = 0 \qquad \forall (x,t) \in \partial\Omega \times (0,T), \tag{2}$$
$$u(x,0) = u_0(x) \qquad x \in \Omega. \tag{3}$$

Dabei sei

$$u_0(x) = \begin{cases} 1, & 0.4 < x < 0.6 \\ 0, & \text{sonst} \end{cases}$$

eine Funktion, die die Anfangswerte beschreibt.
Wählen Sie zum Lösen der Differentialgleichung ein Finites-Differenzen-Verfahren /ein explizites Eulerverfahren, indem Sie wie folgt vorgehen:
Zunächst wählen wir eine Diskretisierung für die Zeit: Für das Zeitintervall $[0,T]$ definieren wir eine Zerlegung der Feinheit $\Delta t = \frac{T}{N}$, durch $t^i := i\Delta t$, $i = 0, \ldots, N$.
Nun wählen wir eine Diskretisierung für den Ort: Für das Intervall $[0,1]$ definieren wir eine Zerlegung der Feinheit $\Delta x = \frac{1}{M}$, durch $x^i := i\Delta x$, $i = 0, \ldots, M$.
Mit u_i^n bezeichnen wir nun eine diskrete Lösung unserer Differentialgleichung zu einem festen Zeitpunkt t^n auf dem Intervall x^i.
Mit u^n bezeichnen wir analog eine diskrete Lösung zu einem festen Zeitpunkt t^n.
Zunächst projiziert man die Anfangswerte u_0 auf eine diskrete Funktion, z.B. $u_i^0 := u_0(x^i)$.
Nun wendet man iterativ das explizite Eulerverfahren zum Berechnen des nächsten Zeitschrittes an, d.h. man berechnet für gegebenes u^n die nächste diskrete Lösung u^{n+1}, gegeben durch

$$\frac{u^{n+1} - u^n}{\Delta t} + \partial_{xx} u^n = 0.$$

Implementieren Sie das Verfahren (nur detaillierte Simulation)!

Tipp: Sie können sich entweder an der Implementierung von Aufgabenblatt 1 orientieren oder einen eigenen Ansatz wählen.