

Einführung in die Objekt-Orientierte Programmierung mit MATLAB

Von Stuart McGarrity

DIE OBJEKT-ORIENTIERTE PROGRAMMIERUNG (OO) IST EINE METHODIK in der Softwareentwicklung, bei der Zusammenhänge zwischen Daten und Algorithmen in Form von Objekten dargestellt werden. Dadurch können wieder verwendbare Komponenten erstellt werden, die sich in größeren Programmen einfach nach dem Baukastenprinzip zusammensetzen lassen. Für Wissenschaftler und Ingenieure ist dies eine effektive Möglichkeit, komplexe Systeme in Teilkomponenten zu zerlegen, zu verstehen und einmal geleistete Arbeit wieder zu verwenden. Das erleichtert auch erheblich die Verwaltung komplexer Software-Entwicklungen. Objekt-orientierte Arbeitsweisen sind von großer Bedeutung für die Entwicklung und Pflege großer Anwendungen und Datenstrukturen.

In diesem Artikel wird anhand der Implementierung einer technischen Anwendung gezeigt, wie objekt-orientierte Methoden in der MATLAB-Sprache umgesetzt werden können. Die vorgestellten Beispiele beruhen auf Funktionen aus MATLAB® 7.6 aus Release 2008a.

Anwendungsbeispiel: Analyse von Daten eines Sensorengitters

Als Sensorengitter (Abb. 1) bezeichnet man Anordnungen von Sensoren, mit deren Hilfe Medien wie Luft, Wasser oder der Erdboden abgetastet werden. Sensorengitter sind oft linear aufgebaut und werden beispielsweise für Radar- und Sonaranwendungen, im Mobilfunk und für viele andere Aufgaben eingesetzt. Durch eine nach Ankunftszeiten der Signale geordnete Sammlung von Daten an unterschiedlichen Punkten im Raum lassen sich zusätzliche Informationen über das abgetastete Medium extrahieren.

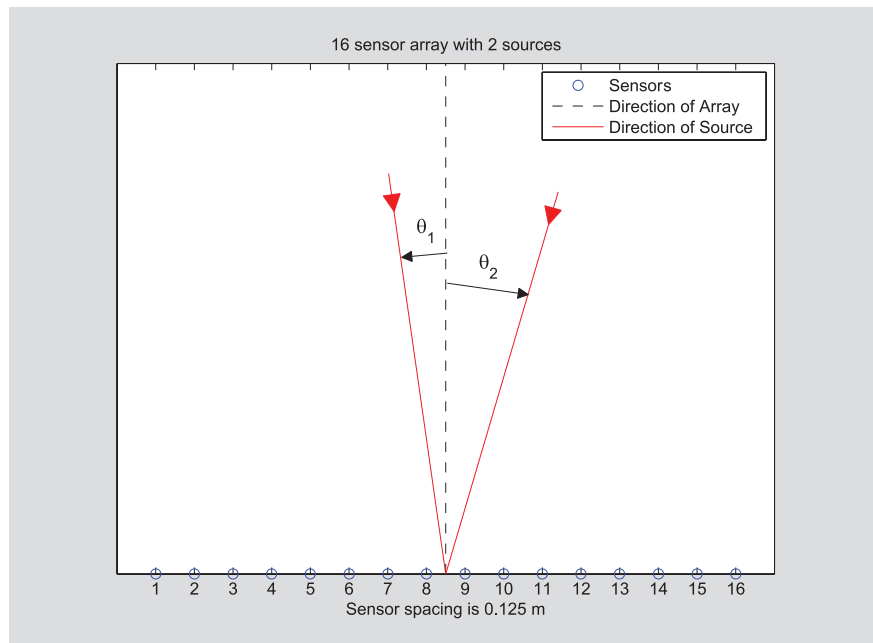


ABB. 1. Ein Sensorengitter, das zwei entfernte elektromagnetische Quellen mit unbekanntem Empfangswinkel detektiert.

In der hier geschilderten Anwendung dient ein Sensorengitter zur Ermittlung der Empfangsrichtung (Direction of Arrival, DOA) mehrerer elektromagnetischer Quellen wie etwa Funkbaken oder Radarsender. Im gezeigten Szenario sollen die Winkel θ_1 und θ_2 der beiden Quellen relativ zur Blickrichtung des Sensorengitters bestimmt werden.

Zusammenstellung der Datenelemente und Operationen

Vor Beginn der Anwendungsentwicklung muss geklärt werden, welche Datenelemente benötigt und welche Operationen an diesen ausgeführt werden. Wie bei den meisten Anwendungen muss man zur Ausführung der notwendigen Operationen eine Reihe verschiedener Arten von Daten speichern und nachverfolgen. Im gewählten Beispiel sind dies:

- Anzahl der Sensoren und Samples
- Aufgezeichnete Datensamples
- Abtastrate der Sensoren
- Sensorenabstände
- Wellenlänge(n) der entfernten Quellen
- Wellenausbreitungsgeschwindigkeit
- Name oder Beschreibung des Sensordatensatzes

Die DOAs der Quellen werden hier mit einer einfachen FFT-basierten Methode abgeschätzt. Diese Methode lässt sich in mehrere Teile herunterbrechen und so als Abfolge von Operationen implementieren. Zur Vereinfachung der Entwicklungsarbeit werden aber zunächst einige Hilfsoperationen implementiert, die:

- den Datensatz aus synthetischen Daten oder echten Messdaten erzeugen
- Werte und Parameter des Datensatzes prüfen und verändern
- die abgetasteten Daten grafisch darstellen und damit deren Interpretation und Validierung erleichtern
- das gemittelte Amplitudenquadrat der FFT des Datensatzes berechnen und es als Periodogramm darstellen
- die Spitzen des Periodogramms finden, mit deren Hilfe die DOA der Quellen abgeschätzt wird

Mit diesen Informationen lässt sich nun festlegen, welche Teile des Programms durch Klasseigenschaften dargestellt werden und welche als Klassenmethoden implementiert werden.

Darstellung von Daten durch Klasseigenschaften

Als erstes wird eine Klasse definiert, die das Sensorengitter beschreibt. Diese erste Beschreibung enthält nur die Datenelemente, die als Klasseigenschaften dargestellt werden.

In MATLAB definiert man Klassen mithilfe von Klassendefinitions-Files. Eine solche Datei enthält Codeblöcke aus Schlüsselwörtern und Endanweisungen, die die verschiedenen Merkmale der Klasse beschreiben. Das in Abbildung 2 gezeigte Definitions-File beschreibt die Klasse `sads` (kurz für Sensor Array Data Set). Alle zu ihrer Darstellung notwendigen Datenelemente sind in einem gemeinsamen Properties-Block zusammengefasst.

Erzeugung von Objekten und Zugriff auf deren Eigenschaften

Ein Objekt – oder eine Instanz der eben definierten Klasse lässt sich nun erzeugen durch die Anweisung:

```
>> s=sads;
```

Eigenschaften weist man auf die gleiche einfache Weise Werte zu wie Feldern einer Struktur, beispielsweise mit:

```
>> s.NumSensors=16;
```

Das Objekt mit allen seinen verfügbaren Eigenschaften sowie deren momentanen Werten

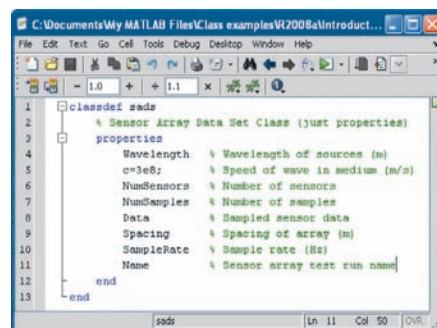


Abb. 2. Das Klassendefinitions-File `sads.m` mit Eigenschaften.

Die Sprache der Objektorientierten Programmierung

Eine Software-Anwendung kann die unterschiedlichsten Kategorien oder Gegenstände abbilden. Dies können physikalische Objekte sein wie etwa Fahrzeuge oder Organismen, virtuelle Gebilde wie Finanzmärkte, aber auch Informationen wie etwa Testergebnisse. Bei der objektorientierten Programmierung werden diese Kategorien als Klassen dargestellt. Datenelemente oder Zustände werden dabei als Klasseigenschaften behandelt, Operationen als Klassenmethoden implementiert.

Ein Objekt ist eine Instanz einer Klasse – während der Ausführung eines Programms wird das Objekt gemäß der Klassendefinition erzeugt und zeigt das für diese Klasse festgelegte Verhalten. Alle in MATLAB-Variablen gespeicherten Werte gehören zu einer bestimmten Klasse. Zu diesen Werten gehören nicht nur solche, die üblicherweise als Objekte aufgefasst werden, wie etwa Zeitreihen- oder Zustandsraum-Objekte, sondern auch simple Double-Werte.

lässt sich nun durch Eingabe seines Namens anzeigen.

```
>> s
s =
  sads
properties:
    NumSensors:    16
    NumSamples:    []
    Data:          []
    SampleRate:    []
    Spacing:       []
    Wavelength:    []
    c:              300000000
    Name:          []
list of methods
```

Alle Eigenschaften außer `NumSensors` und `c` sind noch leer. In den Funktionen `class` und `isa` sowie mit dem Befehl `whos` wird

der Datensatz nun als `sads`-Objekt identifiziert. Hierin unterscheidet er sich von einer Struktur.

```
>> class(s)
ans =
sads
```

Die Fähigkeit, die Klasse einer Variablen zu identifizieren, ist wichtig, wenn man ein Programm schreiben möchte, das den Datensatz verarbeitet. Der Anwender kann dadurch unmittelbar alle verfügbaren Datenelemente sowie die daran erlaubten Operationen ermitteln.

Fehlerkontrolle

Werden Daten durch Strukturen dargestellt, kann man diese Strukturen jederzeit um neue Feldnamen erweitern, indem man den neuen Namen definiert und ihm einen Wert zuweist. Diese Fähigkeit ist besonders praktisch in der Experimentier- oder Prototyping-Phase neuer Algorithmen. Schreibt man dabei aber einen Feldnamen einmal falsch, dann erzeugt man unbemerkt ein neues Feld, das später einen Fehler erzeugen kann, der sich nur schwer diagnostizieren lässt.

Im Gegensatz zu Strukturen kann man Objekte nicht beliebig durch Definition neuer Eigenschaftsnamen und Werte um neue Eigenschaften erweitern. MATLAB gibt einen Fehler aus, sobald man den Eigenschaftsnamen eines Objekts falsch schreibt. Diese zusätzliche Fehlerkontrolle erweist sich als nützlich, wenn ein Anwender auf das Objekt zugreift, der damit nicht so vertraut ist wie der Autor, was

insbesondere bei der Entwicklung umfangreicher Anwendungen häufig der Fall ist.

Festlegung von Zugriffsarten und Zugriffsrechten

Mit Klassen lässt sich der Zugriff auf Objekteigenschaften umfassend kontrollieren. So kann man etwa Eigenschaften dynamisch berechnen lassen, verbergen oder ihre Veränderung verbieten. Die Art des Zugriffs auf Eigenschaften wird durch Definition von Eigenschafts-Attributen im Klassendefinitions-File festgelegt.

Das Klassendefinitions-File aus Abbildung 2 lässt sich entsprechend erweitern, indem man die aufgelisteten Eigenschaften auf mehrere neue, nach eindeutigen Eigenschaftsattributen geordnete Blöcke verteilt (Abb. 3). Im vorliegenden Fall sind das die Attribute `GetAccess`, `Constant` und `Dependent`.

Die Veränderung einer Eigenschaft, hier der Lichtgeschwindigkeit `c`, verbietet man mit dem Attribut `Constant`. Da konstante Eigenschaften sich nicht ändern, kann man ganz einfach über den Klassennamen auf sie zugreifen:

```
>> sads.c
ans =
300000000
```

Den Schreibzugriff auf eine Eigenschaft verbietet man durch Setzen des Attributs `SetAccess` auf `'private'`. Durch Setzen des Attributs `GetAccess` auf `'private'` ist dagegen eine Eigenschaft ausschließlich für die auf sie angewandten Methoden sichtbar. In

diesem Beispiel gilt das für die `Wavelength`-Eigenschaft.

Namen oder Merkmale von `'private'`-Eigenschaften kann man beliebig verändern, ohne dass dies Auswirkungen auf die Nutzer des zugehörigen Objekts hat. Dieser Verkapselung oder Encapsulation genannte „Black Box“-Ansatz zur Definition eines Programmteils verhindert, dass ein Nutzer eines Objekts von einem bestimmten Detail oder Merkmal der Implementierung abhängig wird, das möglicherweise später verändert wird und damit einen Bruch im Programmcode erzeugt.

Eigenschaften, die nur auf Anforderung berechnet werden, versteht man mit dem Attribut `Dependent`. Anschließend definiert man eine Get-Methode, die beim Zugriff auf die Eigenschaft automatisch aufgerufen wird. Einzelheiten dazu sind im Kapitel „Zugriff auf Eigenschaften mit Get- und Set-Methoden“ dieses Artikels beschrieben. In der vorliegenden Anwendung sind die Eigenschaften `NumSensors` und `NumSamples` auf `Dependent` gesetzt.

Implementierung von Operationen mit Klassenmethoden

Methoden, also auf ein Objekt anwendbare Operationen, werden als Funktionsliste in einem Methoden-Block definiert. Eine Klasse kann viele verschiedene Arten von Methoden enthalten, von denen jede ihrem eigenen Zweck dient und auf ganz bestimmte Weise definiert ist. Im folgenden Abschnitt werden verschiedene Methodentypen vorgestellt.

Das Definitions-File der Klasse `sads` wird zunächst um einen Methodenblock erweitert, der zur Aufnahme aller künftig erzeugten Methoden bestimmt ist (Abb. 4).

Definition einer Konstruktor-Methode

Zuerst wird eine Konstruktor-Methode definiert, mit deren Hilfe der Anwender die Werte einiger wichtiger Parameter bereits bei der Erzeugung des Objekts festlegen kann. Konstruktor-Methoden werden häufig zur Initialisierung und Validierung von Daten eingesetzt. Das Objekt wird nun durch folgende Eingabe erzeugt:

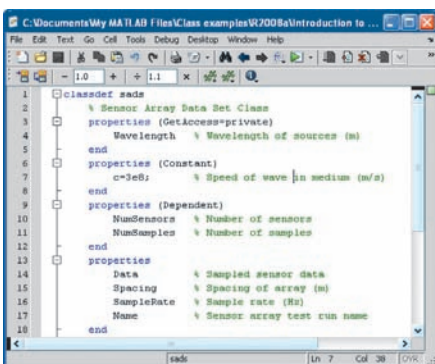


Abb. 3. Das Klassendefinitions-File `sads.m` mit seinen Eigenschaftsattributen.

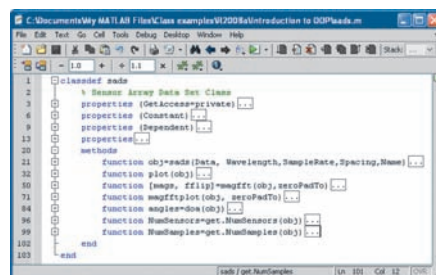


Abb. 4. Das im MATLAB Editor angezeigt Klassendefinitions-File `sads.m` mit seinen Methoden. Der Übersicht halber wurde ein Großteil des Programmcodes durch die Codefaltungs-Funktion verborgen.

```
>> s=sads(Daten, Wellenlänge,
         Abtaste, Abstand, Name);
```

Implementierung anwendungsspezifischer Methoden

Danach müssen die Methoden implementiert werden, die die für den Anwendungszweck erforderlichen Operationen am Datensatz ausführen. Die meisten Methoden nutzen das Objekt als Eingabeargument (beispielsweise `obj`) und greifen, wie in der folgenden Methode gezeigt, durch Referenzierung dieser Variablen auf die Objekteigenschaften zu (beispielsweise `obj.NumSamples`):

```
function [mags, fflip]=magfft(obj,
zpt)
mag=zeros(obj.NumSamples, zpt);
...
end
```

Die Referenzierung von Eigenschaften über Objektvariablen erfordert zwar zusätzliche Syntax, ist aber nützlich, weil sie Objektvariablen eindeutig von lokalen Funktionsvariablen – etwa `mag` im obigen Beispiel – abgrenzt.

Aufrufen von Methoden

Methoden werden auf die gleiche Weise aufgerufen wie Funktionen. Ruft man etwa die Methode zur Berechnung der DOA der Quellen auf:

```
>> angles=doa(s)
angles =
-10.1642  18.9953
```

So erhält man zwei Werte, die relativ nahe an den tatsächlichen Werten von -10° und 20° in Abbildung 1 liegen.

Zugriff auf Eigenschaften mit Get- und Set-Methoden

Durch Definition mit bestimmten Eigenschaften verknüpfter Set- und Get-Methoden können, wie bereits erwähnt, Eigenschaften validiert oder von Bedingungen abhängig gemacht werden. Die Get-Methode für die Eigenschaft `NumSensors` lautet:

```
function NumSensors=get.NumSensors(obj)
NumSensors=size(obj.Data, 2);
end
```

Get- und Set-Methoden werden beim Zugriff auf Eigenschaften automatisch aufgerufen, beispielsweise durch

```
>> N=s.NumSensors;
```

Definition von Methoden aus MATLAB-Funktionen durch Overloading

Durch Overloading lassen sich vorhandene MATLAB-Funktionen so umdefinieren, dass sie als Name in der Liste der Klassenmethoden auftauchen und auf Objekte angewandt werden können. Auch Operatoren und sogar Indexierungen können überladen werden, indem man Methoden mit entsprechenden Namen einsetzt. In diesem speziellen Fall wird die `plot`-Funktion aus MATLAB als Grafik-Methode überladen und damit eine Visualisierungsfunktion für den Datensatz geschaffen, die den meisten Anwendern vertraut ist (Abb. 5).

```
>> plot(s)
```

Diese Grafikmethode ist genau auf die vorliegende Anwendung zugeschnitten: Sie zeigt den Datensatz auf zweckdienliche Weise an und enthält alle zu seinem Verständnis benötigten Informationen. Sie wird nur an Objekten ausgeführt, für die sie explizit definiert wurde. Diese Vorgehensweise ist deutlich robuster als die Alternative, die Reihenfolge der Verzeichnisse im MATLAB-Pfad so zu manipulieren, um aus verschiedenen Funktionen gleichen Namens nur die gewünschte aufzurufen.

Weiterentwicklung der Anwendung

Die in diesem Artikel erzeugte Klasse repräsentiert den Datensatz für das Sensorengitter und enthält eine Reihe von Operationen, mit denen sich Daten analysieren lassen, insbesondere die zur Richtungsermittlung dienende Hauptoperation. Diese Klasse kann nun beispielsweise genutzt werden, um die Leistung der FFT-basierten Methode in unterschiedlichen Szenarien zu ermitteln.

Die Anwendung lässt sich mit Hilfe weiterer OO-Techniken erweitern. Man könnte beispielsweise:

- Unterklassen vorhandener Klassen mit Vererbung definieren und dabei die Definition einer übergeordneten Kategorie

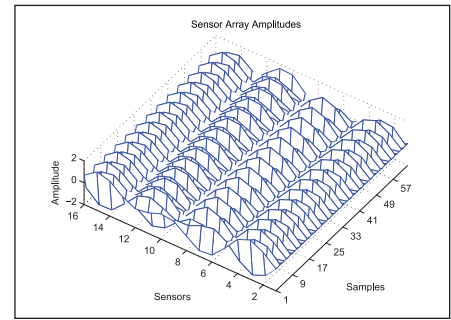


ABB. 5. Durch Overloading erzeugte, für den Datensatz des Sensorgitters optimierte Grafikmethode.

zur Definition einer spezifischeren Unterkategorie wiederverwenden.

- Statische Methoden erzeugen, mit denen sich Operationen für eine Klasse als Ganzes definieren lassen.
- Handle-Klassen mit Referenzverhalten definieren und so die Erzeugung von Datenstrukturen wie Linked Lists ermöglichen oder um mit großen Datensätzen arbeiten zu können, ohne diese kopieren zu müssen.
- Ereignisse und Listener definieren und damit Eigenschaften oder Aktionen von Objekten überwachen.

Durch Einsatz solcher Techniken lassen sich die Beziehungen und das Verhalten innerhalb der Anwendung noch umfassender definieren, wodurch sich insbesondere komplexere Programme leichter handhaben lassen.

Da die vorgestellte Anwendung mit objektorientierten Methoden erzeugt wurde, ist sie nun robust genug, um auch von anderen Anwendern genutzt und gepflegt zu werden. Sie lässt sich außerdem problemlos in eine bestehende Software-Infrastruktur integrieren. ■

➤ **Quellen**

RESSOURCEN ZUR OBJEKT-ORIENTIERTEN PROGRAMMIERUNG
www.mathworks.de/nn8/oop