



# Introduction to Object-Oriented Programming in MATLAB®

By Stuart McGarrity

**Object-oriented programming (OO)** applies to software development the standard science and engineering practice of identifying patterns and defining a classification system describing those patterns. Classification systems and design patterns allow engineers and scientists to make sense of complex systems and to reuse efforts by others. By applying classification systems and design patterns to programming, the OO approach improves your ability to manage software complexity—particularly important when developing and maintaining large applications and data structures.

This article demonstrates the use of object-oriented techniques in the MATLAB® language to implement a typical technical application. The examples use features available in MATLAB 7.6, part of release 2008a.

## Product Used

- MATLAB®

## Application Example: Analyzing Sensor Array Data

A sensor array (Figure 1) is a collection of sensors, often arranged in a line, that is used to sample a medium such as air, water, or the ground for radar, sonar, cellular communications, and other applications. By collecting time samples from multiple points in space, you can extract additional information from the medium being sampled.

Our application uses a sensor array to determine the direction of arrival (DOA) of multiple distant electromagnetic sources, such as radio beacons and radar transmitters. In this particular scenario, we will attempt to estimate the angles  $\theta_1$  and  $\theta_2$  of the two sources relative to the direction in which the sensor array is pointing.

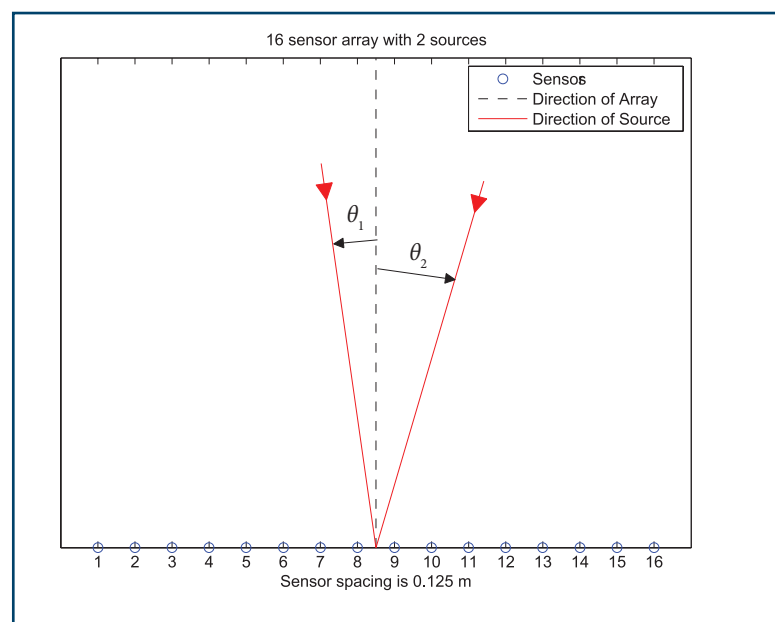


Figure 1. A sensor array detecting two distant electromagnetic sources at unknown angles.

## The Language of Object-Oriented Programming

When creating software applications, the categories or things that you could represent include physical objects, such as a car or an organism; a virtual entity, such as a financial market; or information, such as a set of test results. In object-oriented programming, these categories are represented as classes. Data elements, or state, are represented as class *properties* and operations are implemented as class *methods*.

An object is an *instance* of a class - when a program executes, the object is created based on its class and behaves in the way defined by the class. The values stored in MATLAB variables all belong to a class. These values include not only what you might normally consider objects, such as a time series or state space object, but also simple doubles.

## Classes in MATLAB

In MATLAB, the class of a variable is displayed in the output of the `whos` command, together with other variable characteristics. Examples include `double`, `char`, `int8`, `struct`, and `timeseries`.

```
>> a=1;
>> str='Hello';
>> whos
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	double	
str	1x5	10	char	

## Reviewing Data Items and Operations

We will begin by reviewing the data items that we need to represent and the operations that we need to implement. As with most applications, many pieces of data must be stored and tracked to perform the required operations. We need to represent the following data:

- Numbers of sensors and samples
- Sampled sensor data
- Sensor sample rate
- Sensor spacing
- Wavelength of distant sources
- Speed of wave
- Sensor data set name or description

We will use a simple FFT-based technique to estimate the DOA of the sources. This

technique can be broken down into parts and implemented as a collection of operations. A small number of utility operations needs to be implemented to help with development work. For example, we must:

- Create the data set from synthetic data or acquired live data
- Inspect and modify data set values and parameters
- Plot the sample data to help with interpretation and validation
- Calculate and plot the power spectrum of the data set (by simple magnitude squared of the FFT method)
- Find the peaks of the power spectrum to estimate direction of arrival of the sources

We can now determine what to represent with class properties and what to implement with class methods.

## Representing Data with Class Properties

We begin by defining a class to describe the sensor array. This initial representation contains only the data items, representing them as class properties.

You define a class in MATLAB with a *class definition file*, which contains blocks of code, denoted by keywords and end statements that describe different aspects of the class. The definition file shown in Figure 2 describes a class `sads` (for sensor array data set), with all the data items that we need to represent listed in a properties block.

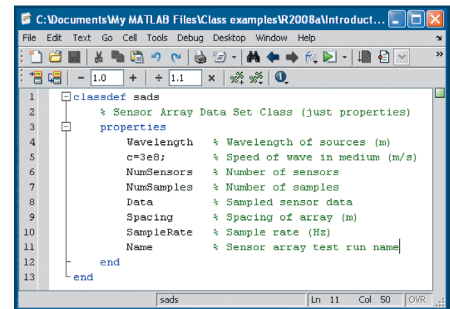


Figure 2. Class definition file `sads.m` with properties.

## Creating an Object and Accessing Properties

To create an object or instance of the class that we defined, we use the statement

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

To set the value of a property, we specify its name just like fields of a structure with

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

We can display the object, seeing all the available properties and current values, by typing its name.

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

All the properties except `NumSensors` and `c` are still empty. By double-clicking the object in the workspace browser, we can inspect or edit the properties using the variable editor just as if they were fields in a structure (Figure 3).

The data set can now be identified as a `sads` object using the `class` and `isa` functions and the `whos` command, something that is not possible with structures.

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

The ability to identify the class of a variable is important to users who create code to operate on the data set, as it lets them determine the available data items to be accessed and operations that can be legally performed.

## Error Checking

If you use structures to represent your data, you could add a new field name at any time simply by specifying a new field name and assigning it a value. This capability is particularly convenient when you are experimenting with and prototyping algorithms. However, if you misspell a field name, a new field will be added silently, which might cause an error later that is difficult to diagnose.

Unlike structures, you cannot arbitrarily add a new property to an object simply by specifying a new property name and assigning it a value. If you misspell an object property name, MATLAB immediately issues an error. This additional level of error checking is useful when

the object is being accessed by users who are less familiar with it than the author, common during the development of a large application.

## Controlling Access to Data

Classes give you great control over property access. For example, they let you prohibit modification of a property, hide a property, or cause it to be calculated dynamically. You control access to properties by specifying property attributes in the class definition file.

We expand on the class definition file in Figure 2 by dividing the current list of properties into multiple property blocks, each with unique property attributes: `GetAccess`, `Constant`, and `Dependent` (Figure 4).

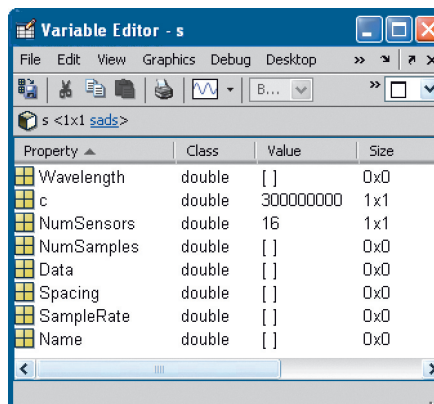
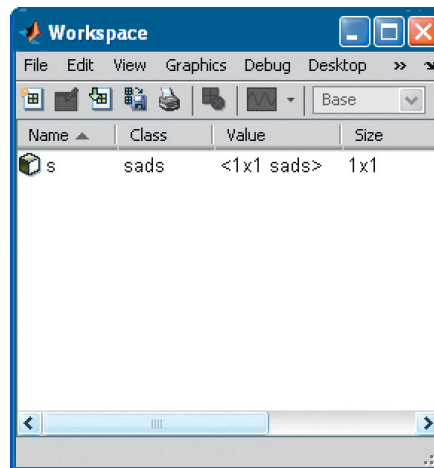


Figure 3. Sensor array object listed in workspace browser (top) and open in variable editor (bottom).

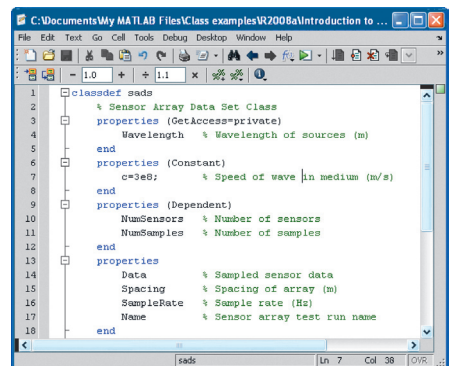


Figure 4. Class definition file `sads.m` with property attributes.

You prohibit modification of a property by setting the `Constant` attribute. In our example, we will set the speed of light property `c` to be constant. Because constant properties do not change, they can be accessed simply by referencing the class name.

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

You make a property read-only by setting the `SetAccess` attribute to private. You can make a property visible only to the methods operating on it by setting the `GetAccess` attribute to private, as we will do with the `Wavelength` property.

You can freely change the names or characteristics of a private property without affecting users of the object. This “black box” approach to defining a piece of software, known as *encapsulation*, prevents the user of the object from becoming dependent on an implementation detail or characteristic that could change and break their code.

You specify that a property is calculated only when asked for by setting its `Dependent` attribute. You then specify a get method that is automatically called when the property is accessed. See the “Accessing Properties with Get and Set Methods” section of this article for details on how to specify class methods. In our application, we set the `NumSensors` and `NumSamples` properties to be dependent.

## Implementing Operations with Class Methods

Methods, or the operations that can be performed on the object, are specified as a list of functions in a methods block. A class can contain many types of methods, each fulfilling a different purpose, each specified differently. The following section describes a number of these types of methods.

We will add a methods block to the `sads` definition file and add each new method inside this block (Figure 5).

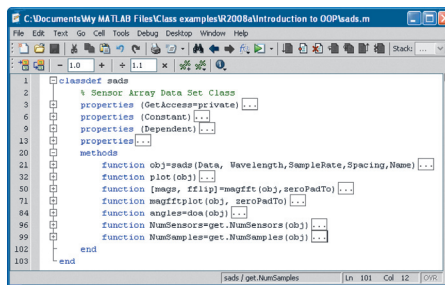


Figure 5. Class definition file `sads.m` with methods, displayed in the MATLAB editor. For ease of viewing, the code-folding feature is used to hide much of the code.

## Specifying a Constructor Method

In our example, we will specify a constructor method that lets the user provide parameters to be used in the creation of the object. The constructor method often performs data initialization and validation. The object is now created with

```
>>s=sads(Data, Wavelength,  
        SampleRate, Spacing, Name);
```

## Implementing Application-Specific Methods

We will add several methods to implement application-specific operations to be performed on the data set. Most methods take the object as an input argument (for example, `obj`) and access the object properties by referencing this variable (for example, `obj.NumSamples`), as in this method:

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

Although it requires additional syntax, referencing properties via the object variable can help differentiate them from local function variables, like `mag` above.

## Calling Methods

Methods are called just like functions, with the object(s) passed in as one of the arguments. We call the method that plots the power spectrum, (Figure 6) passing in additional required arguments.

```
>>magfftplot(s,128)
```

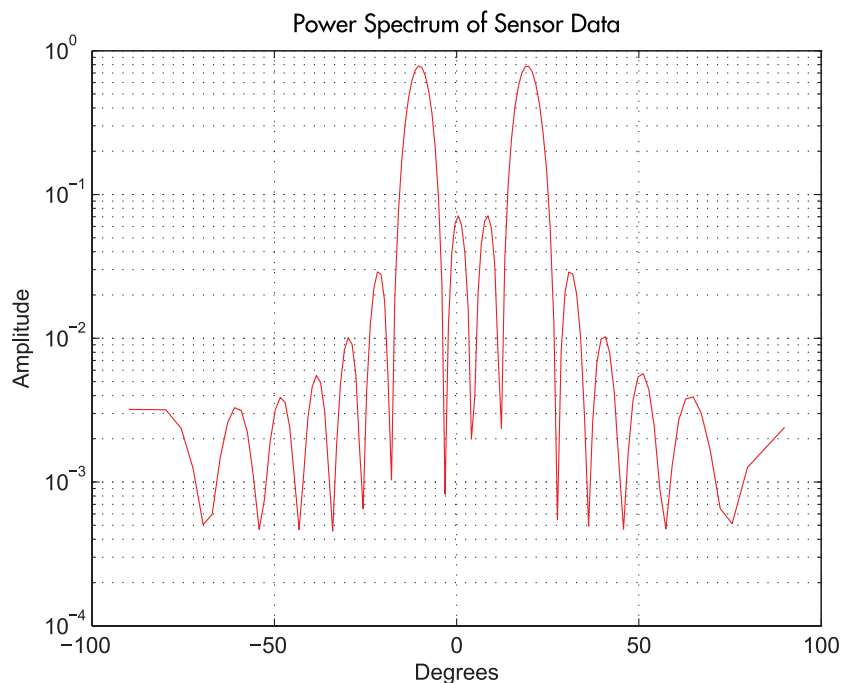


Figure 6. Plot of power spectrum against angle.

Next, we call the method that performs the main operation of estimating the sources' DOA.

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

The DOA angles match the location of the peaks in Figure 2 and approximate the true locations of the sources shown in Figure 1, which are  $-10^\circ$  and  $20^\circ$ .

### Accessing Properties with Get and Set Methods

You can validate properties or implement dependent properties, as mentioned earlier, by specifying associated set and get methods. Here is the get method for the `NumSensors` property.

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

Get and set methods are called automatically when properties are accessed, for example with

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

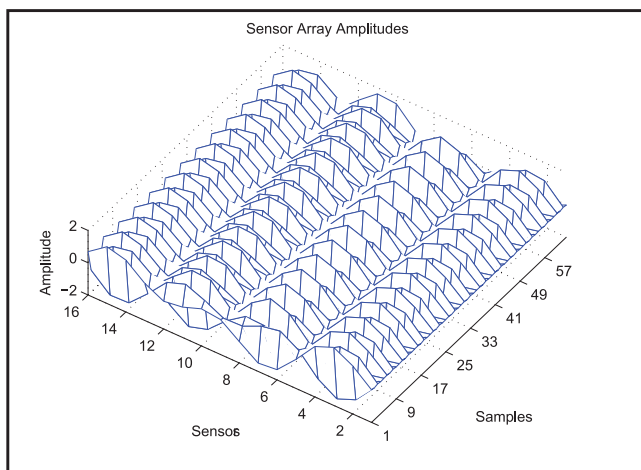


Figure 7. Overloaded plot method specialized for the sensor array data set.

### Specifying Methods for Existing MATLAB Functions with Overloading

*Overloading* lets you redefine existing MATLAB functions to work on your object by providing a function with that name in your list of methods. You can also overload operators and even indexing by using methods with special names. In our application we will include an overloaded plot method, providing a function to visualize the data set that is familiar to many MATLAB users (Figure 7).

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

This customized plot method represents the information in the most appropriate way for this data set, annotating it with all available information. This plot method is executed only on objects for which it has been defined, a much more robust approach than manipulating the order of directories in the path to control which of the multiple functions with the same name are called.

### Developing the Application Further

The class that we created in this example represents our sensor array data set and provides several operations that we can use to analyze the data, including the main direction-finding operation. We can use this class to evaluate the performance of the FFT-based technique in different scenarios.

We could expand the application using additional OO techniques. For example, we could:

- Define sub-classes of existing classes (reusing a definition of a broader category to define a more specific sub-category) with inheritance
- Specify static methods, letting us define an operation for the class as a whole
- Use handle classes with reference behavior, enabling us to make data structures like linked lists or work with a large data set without copying it
- Define events and listeners, letting us monitor object properties or actions

These techniques enhance our ability to manage complexity by enabling us to further define relationships and behavior in the application.

Because it was built using OO techniques, the application is now robust enough for others to use and maintain and can be integrated with related applications throughout an organization. ■

## Object-Oriented Programming Glossary

**Class.** A category or set of objects

**Class definition file.** A MATLAB file defining the behavior of a class

**Method.** An operation that can be carried out on an object

**Object.** An instance of a class, existing in a workspace

**Property.** An item of data or state associated with an object

## For More Information

- Video tutorial giving overview of class development in MATLAB  
[www.mathworks.com/oop\\_video](http://www.mathworks.com/oop_video)
- Wireless communications application implemented with object-oriented techniques in MATLAB including comparison with equivalent C++ code  
[www.mathworks.com/oop\\_wireless](http://www.mathworks.com/oop_wireless)
- Red Black Tree object-oriented example comparing MATLAB code to C++, Java and other languages  
[www.mathworks.com/oop\\_redblack](http://www.mathworks.com/oop_redblack)
- Documentation on getting started with classes in MATLAB  
[www.mathworks.com/oop\\_doc](http://www.mathworks.com/oop_doc)
- Sensor application code  
[www.mathworks.com/dlcode1\\_0308](http://www.mathworks.com/dlcode1_0308)
- Object-oriented programming resources  
[www.mathworks.com/products/matlab/object\\_oriented\\_programming.html](http://www.mathworks.com/products/matlab/object_oriented_programming.html)

## Resources

**VISIT**  
[www.mathworks.com](http://www.mathworks.com)

**TECHNICAL SUPPORT**  
[www.mathworks.com/support](http://www.mathworks.com/support)

**ONLINE USER COMMUNITY**  
[www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)

**DEMOS**  
[www.mathworks.com/demos](http://www.mathworks.com/demos)

**TRAINING SERVICES**  
[www.mathworks.com/training](http://www.mathworks.com/training)

**THIRD-PARTY PRODUCTS AND SERVICES**  
[www.mathworks.com/connections](http://www.mathworks.com/connections)

**Worldwide CONTACTS**  
[www.mathworks.com/contact](http://www.mathworks.com/contact)

**E-MAIL**  
[info@mathworks.com](mailto:info@mathworks.com)

© 2009 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

91560V01 02/09