Classes and Objects

**Objects and Classes**

Objects and Classes

User defined classes in Matlab

Methods

Designing a new class

– Data

– Methods

Constructor

Other methods, including operator overloading

Dispatching and Precedence

Inheritance

Abstract class for get/set/reference/assignment/dispatch

Classes and Objects

Objects are variables with an identity (the class) and functions (the methods) to operate on them.

Data may look like many different things

– given a 3 by 1 column vector. Does it represent

  • a date (day/month/year)
  • a point in 3-dimensional space
    – Cartesian coordinates
    – Cylindrical coordinates
    – Spherical coordinates
  • a quadratic polynomial
  • or something else…

And, what functions make sense on the data?

So, classes give data an identity, and associate functions to the identity (don't allow two dates to be multiplied).

Classes/Objects and Methods in Matlab

Objects are instances of the class

Example: the most common class in Matlab is double.

An instance of a double is, for example,

\[ \text{M} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} \]

Another common class in Matlab is cell.

\[ \text{C} = \{ \text{M} \}, \{ 'e77' \}, \{ \pi \}, \{ [] \}, \{ 2 \}, \{ 3 \} \]

is an instance of class cell.

Both classes have the method `size` and use identical `()` referencing methods. Only `double` (+), only `cell` has `{}`

\[ \text{size(M)} \]
\[ \text{size(C)} \]
\[ \text{M}(1,2) \]
\[ \text{C}(1,2) \]

User Defined Classes/Objects in Matlab

Associated with a class, an Object is

– a structure (ie., `struct`), with a fixed set (for that class) of fields
  – contains the data that constitutes the object
– an internal hidden "tag" which labels the structure as an object of that class, and not just a Matlab `struct`.

The structure's fields are not accessible to the user

The user interacts with the object using methods

– function m-files (for example) that reside in the `method's folder`
  – for that particular class
– Within methods m-files, the fields of the structure are accessible
– Certain referencing methods (subsref and subsasgn) allow
  the programmer to give user access to the fields if desired. Next lecture for these…

Defining a class in Matlab

Need a class for polynomial objects

– purpose: to distinguish them from ordinary row vectors

Pick a name for the class. This is the "classname".

– example: e177poly

Create a folder of the same name as the class, but beginning with the character "@". This is referred to as the `methods folder`.

– example: create folder named @e177poly

Decide on fields, and write the constructor method (an m-file) which creates objects (specific instances of the class)

– The name of the m-file should be the same as the classname.
– The constructor should be placed in the methods folder.

Decide on methods, and write function m-files which implement the methods
**e177poly data fields**

Design decision: the `struct` which contains the data that constitutes an `e177poly` object will have two fields:

- a row vector of coefficients, using the typical Matlab interpretation (i.e., the `roots` command) where the row vector represents the polynomial coefficients, from highest power down to the 0th power. Fieldname is `coeff`.
- a single character for the symbol to represent the indeterminate variable of the polynomial, only used in display purposes. Fieldname is `symbol`.

With this decision made, the constructor can be written.

- The name of the constructor file is `e177poly.m`.
- The constructor file goes in the `@e177poly` folder.

**Role of the constructor**

The Constructor must:

- Accept data as input arguments.
- Create a structure with the appropriate fields.
- Fill the fields of the structure based on the input arguments.
- Use the `class` command to "tag" the structure as an object of the `e177poly` class.

If the constructor is called with:

- no input arguments
  - it should create an "empty" or "default" instance.
- an instance of the class as the input argument
  - it should simply return the object.

**Format of the constructor**

```matlab
function h = e177poly(p)
if nargin==0
    h.coeff = [];
    h.symbol = 'x';
    h = class(h,'e177poly');
else
    if isa(p,'e177poly')
        h = p;
    elseif isa(p,'double')
        h.coeff = p;
        h.symbol = 'x';
        h = class(h,'e177poly');
    end
end
```

**Common methods in the Methods folder**

1. The constructor itself
2. The `display` method, which Matlab automatically calls if there is no semicolon at the end of an expression. Matlab passes the result to the `display` method.
3. Converter methods, which convert the object to a different representation, if appropriate.
4. The `get` method, to access properties of the object.
5. The `set` method, to change properties of the object.
6. Operator overloaded methods, such as `plus`. The expression `A+B` is actually `plus(A,B)`. If `A` and `B` are objects from the same class, and there is a `plus` method, it will be called.

**Design Decision: Methods for e177poly**

Constructor, `e177poly` converter and display methods:

- `double`, converts a polynomial to a row vector of coefficients.
- `char`, converts a polynomial to a formatted string.
- `display`, for expressions that result in an `e177poly` object.

Operator overloaded methods:

- `plus(A,B)` for `A+B`
- `minus(A,B)` for `A-B`
- `uminus(A)` for `-A`
- `umtimes(A,B)` for `*` `A*B`

Overloaded methods:

- `roots`, `polyval`, `polyder`, `polyint`

Access methods (`get`, `set`)`$e177poly$`

**Method Dispatching and Precedence (intro)**

Suppose `A` and `B` are two objects of different class. Consider the expression `SomeMethod(A,B)`

Question: Which method will be called?

- `SomeMethod.m` in the methods folder of `class(A)`, or...
- `SomeMethod.m` in the methods folder of `class(B)`? or...
- `SomeMethod.m` in the Matlab path

Answer:

- If the objects have equal precedence, then the method from `class(A)` (i.e., the first argument) will be called.
- If the objects have unequal precedence, the method associated with the object of higher precedence will be called.

Facts:

- All user-defined classes have higher precedence than Matlab built-in classes (like `double`, `char`). By default, all user-defined classes are of equal precedence.
- Commands `superiorto` and `inferiorto` are used to define precedence among user-defined classes. More next lecture.
Dispatching and Precedence

Since `e177poly` objects have higher precedence than `double` objects, the sequence of commands

\[
A = e177poly([3 -1 4]); \\
C = 2*A \quad \% \text{mtimes} \\
D = A-5 \quad \% \text{minus}
\]

call `@e177poly/mtimes.m` and `@e177poly/minus.m` methods respectively (each with two arguments).

Let's write these multi-input argument functions
- to handle non-`e177poly` objects,
- knowing at least one argument will be an `e177poly` object.

Access methods: get and set

The public properties of the object are
- Retrieved using `get`
- Modified using `set`, which can control changes that are allowed

Properties have names to identify them, so a natural syntax is

\[
\text{PropertyValue} = \text{get}(Object,\text{PropertyName}) \\
\text{set}(Object,\text{PropertyName},\text{PropertyValue})
\]

For the `e77poly` class, let's make `Symbol` a public property
- Its value is the single character symbol used to represent the indeterminate variable of the polynomial
- Since the user will be able to control its value, we won't allow binary operations (like `+` and `*`) unless the symbols match.

Writing get is pretty easy

```matlab
function value = get(p,PropName)
switch PropName
    case 'Symbol'
        value = p.symbol;
    otherwise
        error('Unknown property');
end
```

The convention (we’ll see next week) in Matlab for

value = get(Object)

is to return a struct, whose fieldnames are the public properties. Easy to add this in, using `nargin`.

Writing set is a little harder

```matlab
function set(p,PropName,value)
switch PropName
    case 'Symbol'
        if isa(value,'char') & isscalar(value)
            p.symbol = value;
        else
            error('Invalid Symbol value');
        end
    otherwise
        error('Unknown property');
end
```

Why won’t this work?

Without some trickery, the command

set(a,PropName,Value)

will not work as intended. In other words, after executing

initvalue = get(a,PropName); \\
newvalue = some_new_value; \\
set(a,PropName,newvalue)

the result of

isequal(get(a,PropName),newvalue)

is false, and the result of

isequal(get(a,PropName),initvalue)

is true.

Why? variables are passed to functions by value, changing them in the function only changes the function’s copy (recall how it actually worked – delayed copy...)

(recall) Accessing other workspaces

Two built-in Matlab commands to circumvent this by
- Allowing assignment of variables in other workspaces
- Within a function, giving access to the name of an input argument variable in the caller’s workspace

Specifically, these two functions are

`assignin`
- copy a variable from the current workspace into a different workspace, with a new, given name.

`inputname`
- Access the name (in the caller’s workspace) of an input argument variable.
Fixing `set` uses this idea

Here is one working version of `set` function:

```matlab
function set(p, PropName, value)
    switch PropName
    case 'Symbol'
        p.symbol = value; % need error check
    otherwise
        error('Unknown property');
    end
    assignin('caller', inputname(1), p);
end
```

Could also pass back an argument, or give user the option to do either...

A version of `set` allowing for either...

```matlab
function pnew = set(p, PropName, value)
    switch PropName
    case 'Symbol'
        p.symbol = value; % need error check
    otherwise
        error('Unknown property');
    end
    if nargout == 0
        assignin('caller', inputname(1), p);
    else
        pnew = p;
    end
end
```

Nearly finished: Methods for `e177poly`

- Constructor, `e177poly`
- Converter and display methods
  - `double`, converts a polynomial to a row vector of coefficients
  - `char`, converts a polynomial to a formatted string
  - `display`, for expressions that result in an `e177poly` object
- Operator overloaded methods
  - `plus(A, B)` for `A+B`
  - `minus(A, B)` for `A-B`
  - `uplus(A)` for `+A`
  - `uminus(A)` for `-A`
  - `mtimes(A, B)` for `A*B`
- Overloaded methods
  - `roots`, `polyval`, `polyder`, `polyint`
- Access methods (`get`, `set`)