

Introduction à la programmation objet avec Matlab

Jacques Grelet IR US191 IMAGO
Plouzané



1982 -1986

Electronicien Océanographe

FOCAL Atlantique Equatorial

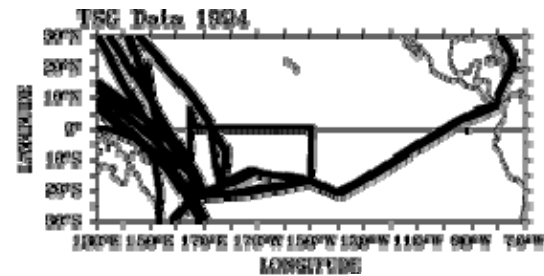


1986 – 1994 Nouméa

SST: Sea Surface Technician



- Programme SURTOPAC
- Réseaux navires marchands
 - XBT
 - Thermosalinographe SST-SSS



1995 -2001
Afrique de l'Ouest
PICOLO - PIRATA



2001 – 2016

IRD Bretagne– campus IFREMER

US191 IMAGO – laboratoire mesures physiques



Contexte de travail

- 82 campagnes océanographiques
- 1747 jours en mer
- Plusieurs casquettes à bord
 - Chef de mission - organisation des campagnes
 - Réalisation (quart) - mise en œuvre du matériel
 - Traitement des données
 - Electronicien sur l'Antea
- Langages programmation
 - Basic HP, C / C++, Java, Matlab ,Perl, PHP, Python, Golang
- Développement de boite à outils
- Productivité
 - Softs de traitement et visualisation
 - TSG-QC
 - dataGUI
 - Toolbox
 - Oceano
 - Datagui.netcdf

Présentation de Matlab

Les plus:

- Logiciel scientifique: éditeur Mathworks
- Langage interprété, objet
- Environnement de développement très complet
- Nombreuses toolbox
 - Payantes: map, images, signal, compilateur, ...
 - Libres: m_map, xUnit, ...
- Réalisation d'interfaces graphiques riches, **interactives**
- Multiplateformes
- Une aide très bien faite,
- Partage de codes: matlab central, github
- Aide en ligne: [stackoverflow](https://stackoverflow.com), developpez.com

Les moins:

- Propriétaire, coût des licences élevées: éducation, individuelles, groupes, jetons
- Performance: interprété, mémoire
- Function is obsolete and will be discontinued. function will be removed in a future release of MATLAB

Help

Object-Oriented Programming

Documentation

Search Help

CONTENTS

Close

< Documentation Home

< MATLAB

< Advanced Software Development

Object-Oriented Programming

Object-Oriented Design with MATLAB

Class Syntax Guide

Sample Class Implementations

Class Definition

Class Customization

Class Editing

Class Metadata

Calling External Functions

Calling Web Services

Toolbox Distribution

Exception Handling

Testing Frameworks

Source Control Integration

Performance and Memory

System Commands

MATLAB API for Other Languages

Object-Oriented Programming

Use object-oriented programming techniques in MATLAB®

MATLAB enables you to use object-oriented techniques in your program design. Object-oriented techniques can simplify programming tasks that involve specialized data structures or large numbers of functions that interact with special kinds of data.

MATLAB classes support function and operator overloading, controlled access to properties and methods, reference and value semantics, and events and listeners.

Example and Syntax Overview

Create a Simple Class

Developing Classes — Typical Workflow

Class to Represent Structured Data

Class Components

Representative Class Code

Comparison of MATLAB and Other OO Languages

Object-Oriented Design with MATLAB

Object-oriented concepts related to MATLAB programming.

Class Syntax Guide

Syntax for defining MATLAB classes and class components

Sample Class Implementations

MATLAB classes showing programming patterns and techniques

Class Definition

Implementation of MATLAB classes

Class Customization

Customize behavior of object indexing, array formation, display, and the save and load operations.

Class Editing

Edit and debug class definitions

Class Metadata

Get detailed information about classes from class metadata

Was this topic helpful?

Yes

No

Structure

```
% structure
field = 'temp';
value = [10, 20, 30];
field2 = 'psal';
value2 = [33, 34, 35];
s = struct(field,value,field2,value2)
var = 'dens';
s.(var) = [20,21,22];
s.psali = 3
vars = {'temp', 'psal'}
for var = vars
    s.(char(var))
end
save('test.mat', s)
load('test.mat')
```

[Visualiser le code sous GitHub](#)

[Executer avec Tutorialpoint.com](#)

Create a Simple Class

```
classdef myFirstClass
    %MESURE Cette class devrait remplacer une structure
    %  usage: m = myFirstClass
    %  m.field = 'temp'
    %  m.value=[10 20 30]
    properties
        field
        value
    end

    methods
        function self = myFirstClass()
            end
    end
end
```

end

[myFirstClass.m](#)

[run_myFirstClass.m](#)

Important: si vous modifiez votre classe, faire un « clear class » ou « clear all » dans la console

Property Set Methods

```
classdef myFirstClass2
    properties
        field
        value
    end

    methods
        function self = myFirstClass2()
            end
        function self = set.field(self,f)
            if ischar(f)
                self.field = f;
            else
                error('myFirstClass2:set.field: not a string');
            end
        end
    end
end

end
```

[run_myFirstClass2.m](#)

Property Get Methods

```
classdef myFirstClass2b
    properties
        field
        value
    end

    methods
        function self = myFirstClass2b() % constructor
        end
        function self = set.field(self,f)
            if ischar(f)
                self.field = f;
            else
                error('myFirstClass2:set.field: not a string');
            end
        end
        function str = get.field(self)
            str = sprintf('property field: %s\n', self.field);
        end

    end % end of public function

end % end of myFirstClass2b class
```

[run_myFirstClass2b.m](#)

containers.Map class

```
%% containers.Map (à partir de R2008b)
mapObj = containers.Map(keySet,valueSet)
mapObj = containers.Map('KeyType',kType,'ValueType',vType)

theKeys =    {'temp', 'psal', 'oxy', 'dens'};
theValues = {[10 20 30], [33 34 35],[120 130 140], [ 22 23 24]};
m = containers.Map(theKeys, theValues)

keys(m)
values(m)
m('flu2') = [0.6 0.7 0.8];
for k = keys(m)
    k = char(k);
    disp(k), disp(m(k))
end
```

[run_containersMap.m](#)

[Hashtable](#) (table hachage)

myFirstClassContainer

```
classdef myFirstClassContainer
    properties
        map % containers.Map
    end

    methods
        function self = myFirstClassContainer(theKeys, theValues) % constructor
            self.map = containers.Map(theKeys, theValues);
        end

        end % end of public function

end % myFirstClassContainer
```

[myFirstClassContainer.m](#)

[run_myFirstClassContainers.m](#)

mySecondClassContainer

```
classdef mySecondClassContainer < containers.Map
    properties
    end

    methods
        function self = mySecondClassContainer(theKeys, theValues)
            % call containers.Map constructeur
            self@containers.Map(theKeys, theValues)
        end

        end % end of public function

    end % end of mySecondClassContainer class
```

[mySecondClassContainer.m](#)

[run_mySecondClassContainers.m](#)

mySecondClassContainer is a handle class now, inherit from containers.Map

Class Definition and File Organization

- **Attributes for Class Members**
 - [Class Attributes](#)
 - [Method Attributes](#)
 - [Property Attributes](#)
 - [Event Attributes](#)
- **Kinds of Classes :**
 - [Comparison of Handle and Value Classes](#)
- **Class Hierarchies**
 - [Inheritance](#) : [interface](#), [abstract](#),
- **[Namespaces](#)**
 - [dataGUI.netcdf](#)
- **[Class Files and Folders](#)**
 - Example: [datagui](#)

Value Class

```
theKeys = {'temp', 'psal', 'oxy', 'dens'};
theValues = {[10 20 30], [33 34 35],[120 130 140], [ 22 23 24]};
m = mySecondClassContainer(theKeys, theValues)
a=m
a =
  Map with properties:
    Count: 4
    KeyType: char
    ValueType: any
m('flu2') = [0.6 0.7 0.8]
m =
  Map with properties:
    Count: 5
    KeyType: char
    ValueType: any
a =
  Map with properties:
    Count: 5
    KeyType: char
    ValueType: any
```

Class handle

```
theKeys = {'temp', 'psal', 'oxy', 'dens'};
theValues = {[10 20 30], [33 34 35],[120 130 140], [ 22 23 24]};
m = mySecondClassContainer(theKeys, theValues)
a=m
a =
  Map with properties:
    Count: 4
    KeyType: char
    ValueType: any
m('flu2') = [0.6 0.7 0.8]
m =
  Map with properties:
    Count: 5
    KeyType: char
    ValueType: any
a =
  Map with properties:
    Count: 5
    KeyType: char
    ValueType: any
```


Nested functions

```
function parent
    disp('This is the parent function')
    nestedfx

    function nestedfx
        disp('This is the nested function')
    end
end
```

- Les fonctions imbriquées peuvent utiliser des variables qui ne sont pas passées explicitement comme arguments d'entrée. Elles partagent le même espace de travail ([workspace](#)) que la fonction parent.
- Dans une fonction parent, vous pouvez créer un [handle](#) vers une fonction imbriquée qui contient les données nécessaires pour exécuter la fonction imbriquée.

Passage de paramètres

- Fonctions:

```
[out1, out2,...] = function(var1, var2,...)
```

- Setappdata/getappdata:

```
tsg = getappdata( hMainFig, 'tsg_data');  
setappdata( hMainFig, 'tsg_data', tsg);
```

- Propriété « userdata » des objets graphiques:

```
% The callback to detect the mouse motion can be set to on  
hMainFig = figure;  
set( hMainFig, 'UserData', 'ButtonMotionOn');
```

- guihandles/guidata:

```
guidata(object_handle,data)  
data = guidata(object_handle)
```

- findobj:

```
% enable toolbar menu pushtool  
hdl_pushtool = findobj('-regexp','Tag', 'PUSHTOOL_');  
set(hdl_pushtool, 'Enable', 'on');
```

Anonymous Functions

Syntax:

```
h = @(arglist)anonymous_function
```

Exemple:

```
sqr = @(n) n.^2;
```

```
x = sqr(3)
```

Callback Function Syntax

Specify a Function Handle:

Function handles provide a way to represent a function as a variable. The function must be a local or nested function.

```
b = uicontrol('Style','pushbutton','Callback',@pushbutton_callback);
```

```
function pushbutton_callback(src,event)
    display('Button pressed');
end
```

Specify a Cell Array:

```
plot(x,y,'ButtonDownFcn',{@lineCallback,'--','*'})
```

```
function lineCallback(src,evt,arg1,arg2)
    src.Color = 'red';
    src.LineStyle = arg1;
    src.Marker = arg2;
end
```

Callback Function Syntax (2)

Specify an Anonymous Function:

Specify an anonymous function when you want a UI component to execute a function that does not support the two arguments that are required for function handles and cell arrays.

```
e = uicontrol('Style','pushbutton','Callback',@(src,event)myfun(x));
```

Specify a Character Vector Containing MATLAB Commands (Not Recommended):

You can specify a character vector when you want to execute a few simple commands, but the callback can become difficult to manage if it contains more than a few commands

```
e = uicontrol('Style','pushbutton','String','Plot line','Callback',  
    'plot(rand(20,3))');
```


Lecture de fichier, vectorisation

Fichiers:

```
# file_type = ascii
*END*
 322.500000      28.8567  0.000e+00
 322.500012      28.8573  0.000e+00
 322.500023      28.8579  0.000e+00
...
```

[Code:](#)

```
fid = fopen(fileName);
[A,count] = fscanf( fid, '%g', [columns,inf] );
nb = count / columns;
disp( [num2str(nb),' records'] );
fclose(fid);    % close file
A=A'; % transpose matrix
% read first 10 lines
JulianDay = data(1,1:10)
```

Les expressions régulières: regex

```
% extract software version
% ex: * Software Version Seasave V 7.16
s = regexp(str, '^*\s*Software Version Seasave.*(<version>\d+\.\d+)', 'names');
if ~isempty(s)
    self.Seasave_Version = s.version
    continue
end

% extract NMEA Longitude
% ex: * NMEA Longitude = 023 00.01 W
match = regexp( str, '^*\s*NMEA
    Longitude\s*=\s*(?<degree>\d+)\s+(?<minute>\d+\.\d+)\s*(?<hemi>\w{1})', 'names ');
if ~isempty(match)
    deg  = match.degree;
    min  = match.minute;
    hemi  = match.hemi;
    self.Longnum = str2double(DegMin_2_Dec(sprintf('%s%s',deg, min), hemi));
    continue
end
```

Tester ses regex: [regexcoach](https://regexcoach.com/)

Les expressions régulières: regex

```
> str = 'Pressure [dbar]';  
> toks = regexp( str, '^(?<name>.*)?\s+\[(?<unit>.*)\]$', 'names');
```

```
toks =
```

```
struct with fields:
```

```
    name: 'Pressure'
```

```
    unit: 'dbar'
```

```
> tr = 'Date Time Latitude Temp';
```

```
> dr = regexp( str, ' ','split')
```

```
hdr =
```

```
1×4 cell array
```

```
    'Date'    'Time'    'Latitude'    'Temp'
```

MATLAB > R2014b

- **MATLAB < R2014b**

```
> f = figure  
f = 1
```

- **MATLAB > R2014b**

```
>> f = figure  
f = Figure (1) with properties:  
    Number: 1  
    Name: ''  
    Color: [0.9400 0.9400 0.9400]  
    Position: [680 558 560 420]  
    Units: 'pixels'
```

```
% get the figure number, starting in R2014b figure became an object  
if verLessThan('matlab','8.4')  
    figureNumber = num2str(f);  
else  
    figureNumber = num2str(f.Number);  
End  
cmd = ['print -f' ,figureNumber, ' -d', printer, ' ', fileName extension];  
eval(cmd);
```

Astuces (à compléter)

Commentaires multilignes:

```
%{  
...  
Block of COMMENTS HERE  
...  
...  
%}
```

Pour rédiger du code:

Mettre un point d'arrêt.

Lancer l'exécution en mode run.

Ecrire le code puis l'évaluer avec F9

Méthode display():

Supprimer le ; en fin de ligne appel la méthode display sur l'objet

Tests unitaires

- Le but est de tester chaque élément du code avant et après chaque modifications ou ajout de fonctionnalités
- Méthode agile:
 - écrire les tests avant le développement du code métier
- Frameworks
 - [matlab-xunit](#)
 - [unit-test](#)(intégré à Matlab R2013b)
- Exemple: dataGUI [+tests](#)
 - > runxunit datagui.tests
 - > [runtests](#)

Tests unitaires

- Offre une meilleure vision de l'interaction d'un objet avec le mode réel (quand on rédige les tests)
- Facilite le cycle de développement : modification -> tests -> correction -> tests

Quelques exemples

- Paquetage dataGUI
 - datagui.dynaload: [code](#)
 - Descripteur : fichiers Excel/csv/json
 - Structure NetCDF indépendante du code de l'application
 - datagui.netcdf : représentation d'un fichier NetCDF en mémoire
 - datagui.ncload: charge les variables dans l'espace de travail
 - datagui.ncdump: émule la commande ncdump Unidata
 - Datagui: [exemples](#)
 - Netcdf et formulaire: [github](#)
 - Lecture de fichier Excel [classe excelRead.m](#)
 - Lecture de fichier ODV: [classe odcReaderWriter](#)
- TSG-QC
 - TSG-QC: [code sous subversion](#)
- dataGUI
 - dataGUI: [version 1](#)

Liens utiles

- MATLAB [Object-Oriented Programming](#)
- MATLAB Files Exchange: [Matlab Central](#)
- Aide en ligne: [stackoverflow](#), [developpez.com](#)
- [Undocumented Matlab](#) (360 articles)
- Exécuter du code MATLAB/SCILAB en ligne: [Coding ground](#)
- GitHub : <https://github.com/jgrelet>
- [Blog](#) du l'US191 IMAGO
 - [TSG-QC](#)
 - dataGUI
 -
- MATLAB Tools for Oceanographic Analysis: [SEA-MAT](#)