Nsp/Premia Manual

Jean-Philippe Chancelier Jérôme Lelong

November 29, 2017

1 How Premia is embedded into Nsp

For long, the only way to use Premia was from the command line. With the growing of Premia every year, the need of real graphical user interface has become more and more pressing. The idea of embedding the Premia library in a Matlab-like Scientific Software has come up quite naturally. Unlike a standalone graphical user interface, embedding Premia into Matlab-like Scientific Software provides two ways of accessing the library either through the scripting language or using the graphical capabilities of the software. The possibility of accessing the Premia functions directly at the interpreter level makes it possible to make Premia interact with other toolboxes. Since the license of Premia gives right to freely distribute the version of Premia two year older that the current release, it was important that the scientific software used can be freely obtained and has extensive graphical feature. Nsp fulfilled all these conditions.

The inheritance system of Nsp enables to easily add new objects in the interpreter. This is how we introduced a new type named *PremiaModel*, through which the wide range of pricing problems described in Premia and their corresponding pricing methods are made available from Nsp. The results obtained in a given problem can be used in any post-treatment routines as any other standard data.

For practitioners, the daily valuation of a complex portfolio is a burning issue. Given a bunch of pricing problems to be solved, which are implemented in Premia, how can we make the most of Nsp and the Premia toolbox? First, we needed a way to describe a pricing problem in a way that is understandable by Nsp so that it can create the correct instance of the *PremiaModel* class. We implemented the *load* and *save* methods for such an instance relying on the XDR library (eXternal Data Representation). This way, any *PremiaModel* object can be saved to a file in a format which is independent of the computer architecture; these files can be reloaded later by any Nsp process. Then, a bunch of pricing problems can be represented by a list of files created either from the scripting language or using the graphical interface. Let us give an example of how to create such a file. To save the pricing of an American call option in the one dimensional Heston model using a finite difference method, one can use the following instructions

Examples

```
P = premia_create()
P.set_asset[str="equity"]
P.set_model[str="Heston1dim"]
P.set_option[str="PutAmer"]
```

```
P.set_method[str="FD_Fem_Achdou"]
save('fic', P)
```

Creating an instance of the *PremiaModel* class and setting its parameters are very intuitive. The object saved in the file `fic` can reloaded using the command `load('fic')`.

2 A scripting approach

The *PremiaModel* toolbox can be loaded into *Nsp* by running `exec loader.sce` in the directory `nsp` of *Premia*.

2.1 General functions

In this section, arguments written between square brackets are optional. All the indices start from `1`.

premia_init ()

Description

Create the *PremiaModel* type.

premia_create ()

Description

Create an instance of a *PremiaModel* object and return its address.

premia_get_assets ()

Description

Return a the names of the different asset types handled by Premia.

premia_get_models ([asset=str])

Parameters

str is a string parameter. It must be one of the values returned by `premia_get_assets`. This argument is optional and describes the type of underlying assets. When no `asset` argument is passed, the default value is `asset=equity`.

Description

Return the list of models available for the given asset type.

premia_get_families ([n], [asset=str])

Parameters

n is the index of the family considered. It starts counting at 1.

str is a string parameter. It must be one of the values returned by `premia_get_assets`. This argument is optional and describes the type of underlying assets. When no `asset` argument is passed, the default value is `asset=equity`.

Description

Return the name of the `n`$-th$ family available for the underlying asset type given by the optional argument `asset=str`. When the argument `n` is not given, the function returns the names of the different families available for the given asset type.

premia_get_family (family, [model], [asset=str])

Parameters

family is the index of the family to be considered within the list returned by premia_get_families.

model is the index of the model to be considered within the list returned by premia_get_models.

str is a string parameter. It must be one of the values returned by `premia_get_assets`. This argument is optional and describes the type of underlying assets. When no `asset` argument is passed, the default value is `asset=equity`.

Description

Return the list of options belonging the `family`$-th$ family in the `model`$-th$ model for the asset described by `str`. When no `model` argument is given, the names of all the options available in `family`$-th$ family are returned. If no option of the given family is compatible with the model, an empty string matrix is returned.

premia_get_methods (family, option, model, [asset=str])

Parameters

family is the index of the family to be considered. This index can be guessed using premia_get_families.

option is the index of the option to be considered within the list of options available in the given family for the model defined by `model`.

model is the index of the model to be considered. This index can be guessed using premia_get_models.

str is a string parameter. It must be one of the values returned by `premia_get_assets`. This argument is optional and describes the type of underlying assets. When no `asset` argument is passed, the default value is `asset=equity`.

Description

Return the names of the methods available to carry out the pricing problem defined by the arguments. The return value is a string matrix.

load[fic]

Parameters

fic is the name of a file.

Description

Load the *PremiaModel* object stored in the file `fic`. The file must have been created by the method save.

2.2 PremiaModel's methods

The following methods can be applied to a *PremiaModel* object `M` returned by the function premia_create using the syntax `M.method[]` for instance. Note that unlike functions, parameters are passed to methods within square brackets and not braces.
The symbol | separates complementary parameters. This is actually a concise way of describing two or possibly more ways of calling a method. The use of curly braces to group parameters means that these parameters must either be all used together or all ignored. Note that the curly braces must not be typed into Nsp.

get_asset[]

Description

Return the name of the underlying asset.

get_model[]

Description

Return the name of the model.

get_models[]

Description

Return the names of the models available for the already set asset type.

get_option[]

Description

Return the name of the option.

get_family[n]

Description

Return the names of the options of the `n`-th family which can priced in the already set model. When no option is available in that family, an empty string matrix is returned.

get_method[]

Description

Return the name of the method.

get_methods[]

Description

Return the names of the methods available to solve the Premia problem.

get_model_values[]

Description

Return the list of parameters of the model.

get_option_values[]

Description

Return the list of parameters of the option.

get_method_values[]

Description

Return the list of parameters of the method. When the method does not need any parameter, an empty list is returned.

set_asset[str=asset]

Parameters

asset is a string parameter. It must be one of the values returned by premia_get_assets

Description

Set the underlying asset.

set_model[n | str=model]

Parameters

n is the index of the model to be set in the list returned by get_models[ ], ie within the list of compatible models.

model is a string parameter. It must be one of the values returned by premia_get_models called with the asset type of the object or get_models[ ].

Examples

```
M=premia_create();
M.set_asset[str="credit"];
models=premia_get_models(asset=M.get_asset[])
M.set_model[models(2)];
```

Equivalently, the list of available models for the already set asset type can be obtained by the method get_models[ ].

```
M=premia_create();
M.set_asset[str="credit"];
models=M.get_models[];
M.set_model[models(2)];
```

Description

Set the model.

set_option[{n_family, n_option} | str=option]

Parameters

n_family is the index of the family in which the option is picked up. The index is computed within the restricted list of families compatible with the model already set.

n_option is an integer parameter. It is the index within the `n_family`$-th$ family of the option to be set. Note that the index is taken in the restricted list of options belonging to the given family which are compatible with the model

option is a string parameter. It must be one of the values returned by premia_get_families called with the parameters corresponding to the object.

Description

Set the option using either the two parameters `n_option` and `n_family` or the unique parameter `str=option`. When the two parameter form is used, the option set is the `n_option`

set_method[n | str=method]

Parameters

n is an integer parameter. This is the index of the method to be chosen in the restricted list of pricing methods available for the given asset, model ans option.

method is a string parameter. It must be one of the values returned by premia_get_methods called with the parameters corresponding to the object.

Description

Set the method using either the parameter `n` or `str=method`. The list of available methods can be obtained using the method get_methods[ ].

set_model_values[L]

Parameters

L is a list parameter. It must have the same structure as the one returned by get_model_values[ ].

Description

Set the different parameters of the model. The list `L` must define all the parameters. There is no way to specify just a few of them, instead you can get the list returned by get_model_values[ ], modify a few values and set it back using set_model_values[ ].

set_option_values[L]

Parameters

L is a list parameter. It must have the same structure as the one returned by get_option_values[ ].

Description

Set the different parameters of the option. The list `L` must define all the parameters. There is no way to specify just a few of them, instead you can get the list returned by get_option_values[ ], modify a few values and set it back using set_option_values[ ].

set_method_values[L]

Parameters

L is a list parameter. It must have the same structure as the one returned by get_method_values[ ].

Description

Set the different parameters of the method. The list `L` must define all the parameters. There is no way to specify just a few of them, instead you can get the list returned by get_method_values[ ], modify a few values and set it back using set_method_values[ ].

compute[]

Description

Compute the quantities of interest (price and delta generally). This is a void method. See get_method_results[ ] to withdraw the results of the computation.

get_compute_err[]

Description

If the computation run by compute[ ] did not end successfully, this method returns the corresponding error message. When the computation ended successfully, an empty string matrix is returned.

get_method_results[]

Description

Return a list containing the values computed by the compute[ ] method. If the compute[ ] method has not been called yet, an empty list is returned.

get_method_results_iter[]

Description

Returns two arguments. The first one is a list of 2 or 3 matrices depending on whether the iteration was performed on one or two parameters. The first matrix (resp. first two matrices) contains the values taken by the parameter (the two parameters) on which we iterated. The last matrix contains the prices for the values of the parameter(s) given by the first (resp. first two) matrices. The second returned argument is a string matrix containing the names of the variables over which the iteration was performed.

is_with_iter[]

Description

Return a boolean telling if we have iterated on some parameters.

get_help[type]

Parameters

type is a string parameter. It can be one of the keywords "model", "option" or "method".

Description

Display the PDF help corresponding to the object referred to by the `type` variable.

save[]

Description

Save a *PremiaModel* object to a file in a format that can be easily reloaded into Nsp (see function load). The format of the file is independent of the architecture.

3 The graphical interface

To access the graphical interface, one must execute the file `interface.sci` located in the directory `nsp`.

3.1 Basic usage

premia ()

Description

Launch the graphical interface and returns a *PremiaModel* object linked to the interface, which means that this object is modified each time you change a parameter in the interface.

Examples

```
P = premia ()
// .. do some changes in the interface
P.get_method[] //returns the name of the method
```

Once all the different parameters are properly set, the computation is launched with the button ‘‘Compute’’.

3.2 Advanced usage

The interface has three menus : File, Help, Tools, explained below.

File menu The File menu provides the ability to save or reload a *PremiaModel* object from a file. This is equivalent to using load or save.

Help menu The Help menu provides a fast and easy access to Premia's scientific documentation. You can directly access the help concerning the model, the option or the pricing method. This help is displayed in an external PDF viewer.

Tools menu The Tools menu is intended to be used with parameter iteration. It is possible to use a vector for a parameter which is normally real valued and its means that the prices must be computed for the different values specified in the vector. The different results can be withdrawn using the method get_method_results_iter[ ] and are stored internally so that the values can be plotted using the function Draw from the Tools menu. If a second set of computations is carried out iterating on an other parameter, the size of the two iterations must be the same because the new results are concatenated to the previous ones and plotted on the same graph. This is particularly useful to compare different methods for instance.
The function Clear Data clears the variable in which the results of the iterations are stored.
The function Clear Graph clears the graph which can be redrawn using the function Redraw as far as the Clear Data function has not been called.
Finally, it is possible to add a legend to the graph using the Legend function. The format of the legend is the following leg1@leg2@leg3... where leg1, leg2 and ... are strings. The position of the legend can also be specified 'ul' (upper left), 'ur' (upper right), ' dl' (lower left) or 'dr' (lower right).

Index

compute, 7

get_asset, 4
get_compute_err, 7
get_family, 4
get_help, 8
get_method, 4
get_method_results, 7
get_method_results_iter, 7
get_method_values, 5
get_methods, 4
get_model, 4
get_model_values, 5
get_models, 4
get_option, 4
get_option_values, 5

is_with_iter, 8

load, 3

premia, 8
premia_create, 2
premia_get_assets, 2
premia_get_families, 2
premia_get_family, 3
premia_get_methods, 3
premia_get_models, 2
premia_init, 2

save, 8
set_asset, 5
set_method, 6
set_method_values, 7
set_model, 5
set_model_values, 6
set_option, 6
set_option_values, 7