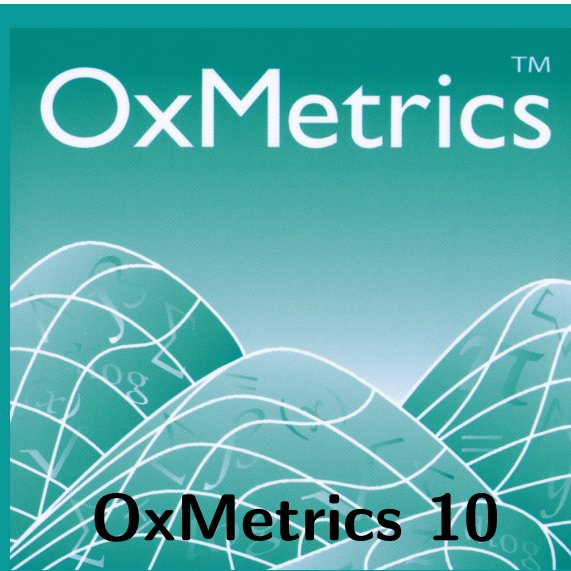
Jurgen A. Doornik

Developer's manual for Ox™ 10



www.oxrun.dev, www.oxlang.dev, doornik.com

Ox^{TM} 10

Developer's manual

First published 1998

Revised in 2012,2018,2021

Trademark notice

All Companies and products referred to in this manual are either trademarks or registered trademarks of their associated Companies.

Contents

Listings

Chapter D1

Introduction

There are several possible reasons why it can be useful to interface Ox with another language, e.g.

- to access a library that is already available, possibly using a C layer in between;
- to implement a small section of code that would be more efficiently executed in another language;
- embed Ox functionality in another system, such as Excel;
- to provide a user interface for existing code.

An easy way to create an interface for an Ox program is using OxPack and the OxMetrics front-end. This only requires coding in Ox, and is documented first in Chapter D2.

Ox is an open system to which you can add functions written in other languages. It is also possible to control Ox from another programming environment such as Visual C++. The interface to the Ox dynamic link library is based on the C language. Most, if not all, other languages provide a mechanism to interface in a manner similar to the C language. For example, under Windows, the Ox interface is very similar to interfacing with the Windows API.

Extending Ox requires an understanding of the innards of Ox, a decent knowledge of C, as well as the right tools. In addition, extending Ox is simpler on some platforms than others. Thus, it is unavoidable that writing Ox extensions is somewhat more complex than writing plain Ox code. However, there could be reasons for extending Ox, e.g. when you need the speed of raw C or FORTRAN code, or to add a user-friendly interface in Java, say. Another case is to write a wrapper to use a function from another library.

When you port some Ox code to C for reasons of speed, make sure that the function takes up a significant part of the time it takes to run the program and that it actually will be a faster in C than in Ox! This chapter gives many examples that could provide a start for your coding effort.

When you write your own C functions to link to Ox, memory management inside the C code is your responsibility. So care is required: any errors can bring down the Ox program, or, worse, lead to erroneous outcomes.

The main platforms for Ox are Windows, Linux and macOS, and the foreign lan-

guage interface is available on each platform. A dynamic link library has the `.so` extension under Linux and macOS, while under Windows it is `.dll`.

Chapter D6 documents the C functions available to interface with Ox. This includes the C mathematical functions exported by the Ox DLL: any program could use Ox as a function library by making direct calls to the Ox DLL.

Note that from Ox 9 onwards, the default version is 64 bit.

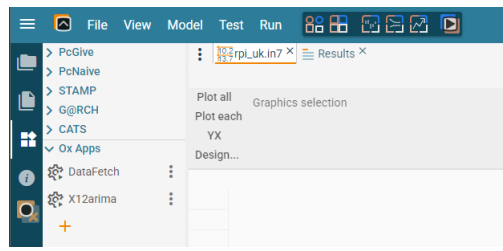
Chapter D2

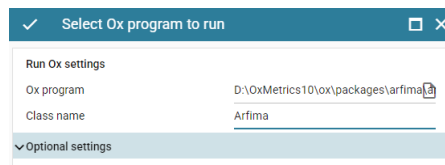
Modelbase and OxApp

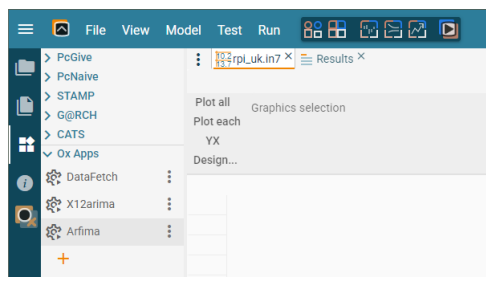
D2.1 Introduction

OxApp allows for interactive use of a `Modelbase`-derived class in cooperation with OxMetrics. This can be achieved solely by adding Ox code. In particular, it is possible to create dialogs, and define menu entries.

The following three captures shows how to add the Arfima package the Ox Apps menus:



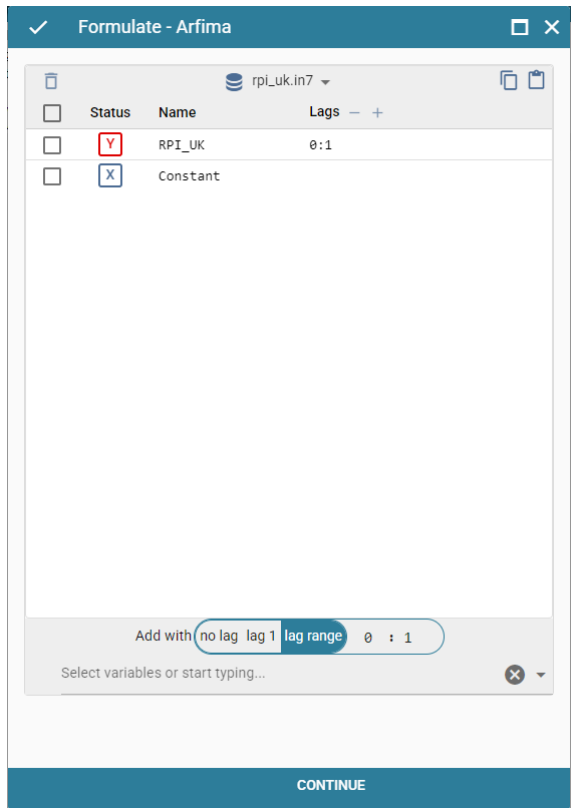




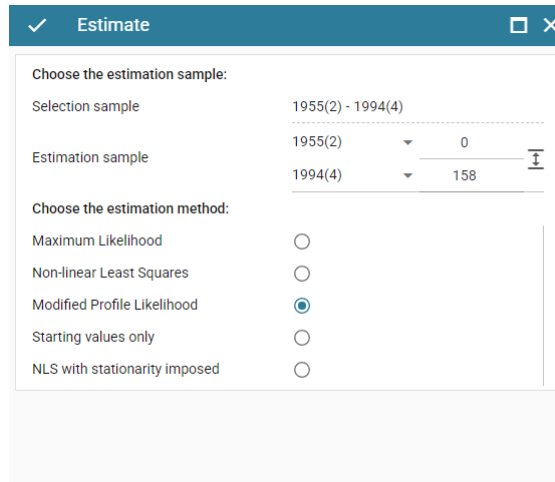
Now it can be started from within OxMetrics. (If OxMetrics does not know where it is installed, it will prompt you to specify the OxMetrics10 root folder.) The items on

the Model and Test menu are specified by the package through the `SendMenu` function.

- Model/Formulate
  This brings up the Model Formulation dialog:



  The Arfima package uses `OxAppQuery` to show the `OP_FORMULATE` dialog. Modelbase uses `SendVarStatus()` in the package to determine the type of variables available to build the model, and `SendSpecials()` to add special variables in the dialog (here they are: Constant, Trend and Seasonal, CSeasonal).
- Model/Model Settings
  The model settings determine the remaining model specification. When the user selects Model Settings on the model menu (or automatically after successful formulation), OxApp calls `ReceiveMenuChoice("OP_SETTINGS")`.
- Model/Estimate



When the user clicks on Continue, the `Estimate` function is called.

- Model/Options
  Options refer to settings which may be less frequently changed. When OxApp calls `ReceiveMenuChoice("OP_OPTIONS")`, the default `Modelbase` implementation allows for the maximization options to be set.
- Test menu
  The menu entries are determined from the return value of `SendMenu("Test")`. The package can again use dialogs to allow the user to choose options.

D2.2 Examples

The following sample code is in `ox/samples/oxapp`:

- `OxAppDialogs`
  Provides an example of all dialogs that can be used in the OxApp application.
- `BprobitEx`
  Provides an example of a Modelbased package that implements an estimation procedure (Probit), and a test menu.
- `OxAppModel`
  Provides an example of an OxApp application which uses a file instead of an OxMetrics database, as well as a descriptive example that uses it's own dialog for variable selection.
- Dialog controls are unchanged except for `CTL_DATEINDEX` and `CTL_DATERANGE` which now uses integer offsets in a vector of dates.

Remember to use the exact class name when adding the packages: OxAppDialogs, BprobitEx and OxAppModel respectively.

Additional examples are (building on code in `ox/src/modelbase.ox`):

- `packages/arfima/arfima.ox`
  The Arfima class shown in the screen captures above.
- `packages/DPD/dpd.ox`
  The DPD class.

D2.3 OxApp replaces OxPack

OxApp provides the functionality to write an interactive application that runs in OxMetrics 10. OxApp replaces OxPack that was used in OxMetrics 9 and 8.

With OxMetrics 9, the OxPack program was launched that positioned itself in between the OxMetrics server and the Ox client. Th OxPack program sent messages between the two, and replaced the Ox output channel, as well as the graphics functions.

The structure in OxMetrics 10 is much simpler: OxMetrics launches the app directly, with text captured and graphs send as a whole gwg file (svg for OxEdit). The new ability of Ox to suspend and resume a function call allows the app to wait for an OxMetrics dialog to finish.

- OxApp functionality is delivered by four functions. These are also defined in Ox itself, but do nothing and return zero when used outside OxApp.
  So OxApp function names do not need to be written as a string, unlike the previous OxPack ones.
- OxMetrics no longer maintains an active database, so the app needs to track this.
- Previously OxPack handled the first model formulation stage. Now this is the responsibility of the app. The Modelbase class has been updated to reflect this, and provides the new implementation.
  OxPack made callbacks to `ReceiveData` and `ReceiveModel` to send the data and model formulation to the OxPack program. These are no longer called. Instead model and data are returned by the `OP_FORMULATE` dialog and set in `Modelbase::FormulateDlg` by calling `GetModelSettings`. `SetModelSettings` is used to obtain the current model settings.
- Because of the new suspend/resume implementation, dialog results cannot return in arguments, but are a direct return value.
- Dialog returns are structured as keyed arrays, which makes processing of the results much easier, and less error prone (in combination with the nullity operator `??`).
- The OxApp interface runs within OxMetrics, rather than a separate program the pops up (and tries to stay on top).
- The `CTL_VARIABLE` dialog control type and `OP_VARIABLE` special dialog are no longer supported.

D2.3.1 OxPack compatibility layer: `oxpack_compat.ox`

This is provided to keep using the old OxPack functionality. Minimal changes to the code are required:

1. Add to your Modelbase derived class:
   ```
   #import <oxpack_compat>
   ```
2. In the constructor add:
   ```
   OxPackRegister(this);
   ```
   This is required for most OxPack functions because they need to interact with the Modelbase derived object (but is not needed for the dialog functions).
3. Provided the code calls `Modelbase::DoFormulateDlg`, it will receive the formulated model and data. Modelbase achieves this by calling `GetModelSettings` and `SetModelSettings`.

4. Provided the code calls `Modelbase::OutputHeader`, the model history will be available in the Formulate dialog, as well as Ox code.
5. In `SendVarStatus`, remove `&Y:` (etc.) from label
6. Implementation of the virtual `GetModelSettings` and `SetModelSettings` functions will need to be changed from (e.g.)

```
Switching::GetModelSettings()
{
 decl aval = Modelbase::GetModelSettings();

 return aval ~
 {
  { m_cS,    "m_cS"    },
  { m_cAR,    "m_cAR"    }
 };
}
Switching::SetModelSettings(const aValues)
{
 if (!isarray(aValues))
  return;

 Modelbase::SetModelSettings(aValues);

 for (decl i = 0; i < sizeof(aValues); ++i)
 {
  if (aValues[i][1] == "m_cS")
   m_cS = aValues[i][0];
  else if (aValues[i][1] == "m_cAR")
   m_cAR = aValues[i][0];
 }
}
```

to:

```
Switching::GetModelSettings()
{
 decl aval = Modelbase::GetModelSettings();
 if (sizeof(aval) == 0)
     return {};

    aval.settings.switching =
 {
  { m_cS,    "m_cS"    },
  { m_cAR,    "m_cAR"    }
 };
 return aval;
}
Switching::SetModelSettings(const aSettings)
{
```

```
    if (!isarray(aSettings))
     return;

    Modelbase::SetModelSettings(aSettings);

       decl aValues = aSettings.settings.switching ?? {};

    for (decl i = 0; i < sizeof(aValues); ++i)
    {
     if (aValues[i][1] == "m_cS")
      m_cS = aValues[i][0];
     else if (aValues[i][1] == "m_cAR")
      m_cAR = aValues[i][0];
    }
   }
```

D2.4 OxApp functions that can be called from Ox

Note that this functionality is only available when running via OxApp. Otherwise all functions do nothing, and return integer zero: `OxAppGet` can be used to check if the program is running as an app.

OxAppDialog

```
#include <oxapp.oxh>
OxAppDialog(const sTitle, const aDialog);
```

sTitle	in:	string, dialog title, prefix with <> for Back, Cancel, Continue, or > to deactivate Back, or <Label> for Label, Cancel, Continue
aDialog	in:	array, dialog definition

Return value

1 if Next is pressed, 0 for Cancel or Close, -1 for Back (only if that button is requested). Array with labels and values in alternating order, so each value can be accessed as *return[label]*.

Description

The aDialog argument is an array of arrays, with each entry consisting of just a text label, or of four or more fields defining the edit control:

1. text label
2. control type
3. control value
4. control arguments
5. control label

An example is:

```
Arfima::DoSettingsDlg()
{
 decl [p, q, fixar, fixma, disfixed, fixd,
```

```
    imean, dmean] = GetSettingValues();

 decl adlg =
 {
  { "ARMA" },
  { "AR order", CTL_INT, p, "p" },
  { "MA order", CTL_INT, q, "q" },
  { "Fix AR lags", CTL_STRMAT, fixar, "fixp" },
  { "Fix MA lags", CTL_STRMAT, fixma, "fixq" },
  { "Fractional parameter d"},
  { "", CTL_RADIOBOX, disfixed, "Estimate d|Fix d", "estd"},
  { "\tat:", CTL_DOUBLE, fixd, "fixd" },
  { "Treatment of mean"},
  { "", CTL_RADIOBOX, imean,
    { "None (or using Constant as regressor)",
     "Deviation from sample mean", "Fix mean"}, "mean"},
  { "\tat:", CTL_DOUBLE, dmean, "fixm" }
 };
 decl avalues = OxAppDialog("<<Model>Model Settings", adlg);
 if (isarray(avalues))
 {
  ARMA(avalues["p"], avalues["q"]);
  if (sizerc(avalues["fixp"]) > 0)
   FixAR(avalues["fixp"]);
  if (sizerc(avalues["fixq"]) > 0)
   FixMA(avalues["fixq"]);

  if (avalues["estd"] == 0)
   FreeD();
  else
   FixD(avalues["fixd"]);

  switch_single (avalues["mean"])
  {
   case 2:  FixMean(avalues["fixm"]);
   case 1:  UseSampleMean();
   default: FixMean(0);
  }
  return avalues["$goBack"] ? -1 : 1;
 }
 return 0;
}
```

If the user presses Next in the dialog, the results are returned in the remaining argument(s).
The dialog returns a keyed array, e.g.:

```
    {"p",4,"q",4,"fixp",<1;2;3>,"fixq",<1;2;3>,"estd",0,"fixd",0,
        "mean",0,"fixm",0}
```

making it possible to get at the required value:

1. using string indexation of an array, e.g. `avalues["p"]`
2. or using keyed indexation:`avalues.p`

The `<Model>` prefix in the title changes the buttons from Cancel, OK to <Model, Continue. If the extra return value `"$goBack"` captures the pressed button: -1 to go back, 1 to go forward; Cancel will result in the dialog return valuue to be zero.

Text label

The text label can have leading tabs (`\t`) to indent the label.

Control type

Control type is

1. a predefined constant defined in `modelbase.oxh`, e.g. `CTL_SELECT`, or
2. the constant as a string, e.g. `"CTL_SELECT"`, or
3. the string in lowercase without `ctl_`, e.g. `"select"`.

Possible values for the control type are:

- Labels, groups, and comment:

`CTL_LABEL`	text label
`CTL_GROUP`	start of a group
`CTL_SUBGROUP`	start of a sub group
`CTL_COMMENT`	comment
`CTL_SEPARATOR`	separator

- Boolean and integer values:

`CTL_CHECK`	check box (0 or 1)
`CTL_INT`	integer
`CTL_INTRANGE`	integer within a specified range
`CTL_RADIOBOX`	group of radio buttons
`CTL_SELECT`	drop-down list of single-select item

- Double and date values:

`CTL_DOUBLE`	double
`CTL_DATE`	date/time value, returned as a double

- String values:

`CTL_EDITOR`	pop-up text editor
`CTL_FILE`	prompt for existing file name
`CTL_FILESAVE`	prompt for file name for saving
`CTL_FOLDER`	prompt for folder name
`CTL_STRING`	string

- Matrix values:

`CTL_MATRIX`	matrix, pop-up matrix editor
`CTL_STRMAT`	matrix, edited as a string

- Sample selection values:

`CTL_SAMPLEINDEX`	index in a fixed frequency sample
`CTL_SAMPLERANGE`	period (range) in a fixed frequency sample
`CTL_DATEINDEX`	index in a date variable
`CTL_DATERANGE`	period in a date variable

- List:

`CTL_LISTBOX`	multiple selection box

- Enabler and disabler:

`CTL_ENABLER`	enables controls based on previous control
`CTL_DISABLER`	disables controls based on previous control

- Formulate a model:

CTL_FORMULATE	keyed array

- Matrix control:

CTL_TABULAR	scalar controls for matrix

Control value

The control value gives the current value of the edit field.

Control label

The last item is a string which identifies the return value. Only entries with a control label have a return value, and the label must be unique within the dialog controls.

Control arguments

The arguments for the control type are:

- Labels and groups:

control type	value	control arguments
CTL_LABEL	—	—
CTL_GROUP	int: 1=expand,0=collapse −1=disable, (2=close)	—
CTL_SUBGROUP	int: 1=start,0=end	
CTL_COMMENT	—	—
CTL_COMMENT	string: comment block	—
CTL_COMMENT	array of strings: comment block	—

CTL_LABEL defines a text label that takes a whole line. Use " ", CTL_LABEL to insert an empty line. A text string followed by nothing else is like a CTL_LABEL, but shown in bold.
The integer argument to CTL_GROUP is 1 if the group is initially expanded, 0 if initially collapsed. A group is automatically closed if a new group is created. To close a group explicitly, set the value to 2.
CTL_SUBGROUP can enclose a CTL_GROUP section to create a sub group. The integer argument is 1 to start a subgroup and 0 to end it.
The text field of CTL_COMMENT can be an empty string. The optional argument is an indented comment block in smaller font, with embdedded "nindicating a new line.
An example is:

```
{   { "PcNaive" },
    { "Monte Carlo Settings", CTL_GROUP, 1 },
    { "Replications", CTL_INT,  m_cRep,     "m_cRep" },
    { "Some text", CTL_LABEL },
    { "", CTL_SUBGROUP, 1 },
    { "Advanced Monte Carlo Settings", CTL_GROUP, 0 },
    { "Discard", CTL_INT,  m_cTdiscard, "m_cTdiscard" },
    { "", CTL_SUBGROUP, 0 },
    { "Monte Carlo Output", CTL_GROUP, 0 },
    { "Asymptotics",  CTL_CHECK,m_bAsymp,   "m_bAsymp" }
```

```
}
```

- Boolean and integer values:

control type	value	control arguments
CTL_CHECK	int: 0 or 1	—
CTL_INT	int	—
CTL_INTRANGE	int	int:min, int:max [,int:step]
CTL_RADIOBOX	int	string/array [,base]
CTL_SELECT	int	string/array [, base]

CTL_INTRANGE and CTL_INTRANGE are rendered as spin buttons. CTL_INTRANGE can have an optional argument after the minimum and maximum to specify the step size.
CTL_RADIOBOX defines a group of radio buttons that are mutually exclusive. The argument is either a string, using | as a separator, or an array of strings. A value of -1 disables the CTL_RADIOBOX control.
CTL_SELECT is rendered as a drop-down box, only allowing a choice from those specified. The argument is either a string, using | as a separator, or an array of strings. A value of -1 disables the CTL_SELECT control.
Both CTL_RADIOBOX and CTL_SELECT allow an optional base argument: then the first choice is not zero but corresponds to base.
Some further examples:

```
{
    { "Rank",   CTL_INTRANGE,   cr, 1, 20, "cr" },
    { "",       CTL_RADIOBOX, type, "AR|ECM|SEM", "t1" },
    { "Type 2", CTL_SELECT,   type, "AR|ECM|SEM", "t2" },
}
```

- Double and date values:

control type	value	control arguments
CTL_DOUBLE	double	—
CTL_DATE	double	—

The double value for a CTL_DATE control is a calendar index, and displayed as a date (or time). The user can enter a date or time, which is automatically translated to a calendar index.

- String values:

control type	value	control arguments
CTL_EDITOR	string	—
CTL_FILE	string	—
CTL_FILESAVE	string	—
CTL_FOLDER	string	—
CTL_STRING	string	—

- Matrix values:

control type	value	control arguments
CTL_MATRIX	matrix	—
CTL_MATRIX	matrix	iMinR, iMaxR, iMinC, iMaxC [,sComment, asRow, asCol]
CTL_MATRIX	matrix	asRow, asCol
CTL_STRMAT	matrix	—

CTL_MATRIX without additional arguments uses the matrix editor with the dimensions fixed by that of the input matrix. More flexibility regarding dimensions can be specified by the four integers setting the minimum, maximum row count, and the minimum and maximum column count. The sComment argument specifies an optional comment (set 0 or empty string for none). The optional asRows and asCols arguments are arrays of strings specifying labels.
Some examples:

```
{
   { "Y",  CTL_MATRIX, m_mY, 1, 5, 1, 5, "m_mY" },
   { "A",  CTL_MATRIX, m_mA, m_asY, {}, "m_mA" },
   { "B",  CTL_MATRIX, m_mB, {}, m_asY~m_asZ, "m_mB" }
}
```

- Sample selection values:

control type	value	control arguments
CTL_SAMPLEINDEX	int	int: iT1, 1×5 matrix
CTL_SAMPLERANGE	int	int: iT1, iT2, 1×5 matrix
CTL_DATEINDEX	int	int: iT1, vector of dates
CTL_DATERANGE	int	int: iT1, iT2, vector of dates as number, or array of strings
CTL_DATEINDEX	int	dbl: dDay, dDayMin, dDayMax
CTL_DATERANGE	int	dbl: dDay1, dDay2, dDayMin, dDayMax, dDaysPerWeek

CTL_SAMPLEINDEX and CTL_SAMPLERANGE are for fixed-frequency samples. The sample is specified in a 1×5 matrix with start year, start period, end year, end period and frequency. iT1 is the integer offset in the sample, while iT1,iT2 is a subsample start and end specified as offsets in the sample CTL_SAMPLEINDEX returns the index in the sample (0 is the start); CTL_SAMPLERANGE returns an array[2] with selected start and finish indices.
CTL_DATEINDEX and CTL_DATERANGE are for samples defined by a vector of calendar dates. Returned is the index in this vector, or an array of two indices for CTL_DATERANGE.
The second versions of CTL_DATEINDEX and CTL_DATERANGE areonly for daily data ranging from dDayMin to dDayMax, and returning the selected date. dDaysPerWeek is currently ignored.
Some examples:

```
if (isdated)
    adlg ~=
    {   { "starts at", CTL_DATEINDEX,
                t1est, m_mData[m_iT1sel : m_iT2sel][0], "t1" },
        { "ends at", CTL_DATEINDEX,
                t2est, m_mData[m_iT1sel : m_iT2sel][0], "t2" }
    };
else
    adlg ~=
    {   { "start", CTL_SAMPLEINDEX, t1est, selsam, "t1"},
        { "end",   CTL_SAMPLEINDEX, t2est, selsam, "t2"}
    };
```

- List:

control type	value	control arguments
CTL_LISTBOX	array of s strings	int:selupto
CTL_LISTBOX	array of s strings	$1\times \leq s$ 0-1 sel.matrix [,max-values]
CTL_LISTBOX	array of s strings	array of sel. strings [,max-values]
CTL_VARIABLE	array of selected strings	array of s strings
CTL_VARIABLE	array of selected strings	database name

Adding CTL_LISTBOX to the dialog controls creates a multiple selection box for the array of strings. If the integer argument selupto is given, entries up to selupto are selected. Note that CTL_LISTBOX *is the only control where the selection comes before the content*.
If a matrix is given, the entries will be selected where the the matrix has a non-zero, and deselected for a zero. The return value consists of a $1 \times s$ matrix with 1 if the entry is selected and 0 otherwise.
If an array is given, it should hold the indices of the selection or the selected strings. Then the return value consists of an array with the selected strings.
With a matrix or array type for the value argument, and optional additional argument limits the number of options that can be selected to max-values.
The colour of the selected items in CTL_LISTBOX changes for suffix Y or I, for example "Yb\tY".
In the following example there are m_cY strings in m_asY, so they are all selected:

```
{   { "Forecast" },
    { "Equations", CTL_LISTBOX, m_asY, sizeof(m_asY), "listbox" },
    { "Select", CTL_VARIABLE, {}, GetDbName(), "variables" }
}
```

- Enabler and disabler:

control type	value	control arguments
CTL_ENABLER	int	int: control_count
CTL_ENABLER	int	int: control_count, int: starting offset
CTL_DISABLER	int	int: control_count
CTL_DISABLER	int	int: control_count, int: starting offset

CTL_ENABLER enables the next control_count control if the previous integer valued control has the specified value. CTL_DISABLER does the opposite: it disables where CTL_ENABLER enables and vice versa.
Some examples:

```
{
    { "Type", CTL_SELECT, type, "AR|ECM|SEM", "type" },
    { "", CTL_ENABLER, 1, 1 },
    { "\tRank",   CTL_INTRANGE,   cr, 1, 20, "cr" },
    { "Set count", CTL_CHECK,  bSet,  "bSet" },
    { "", CTL_ENABLER, TRUE, 1 },
    { "\tCount",   CTL_INT,    cCount, "cCount" }
}
```

- Formulate a model

control type	value	control arguments
CTL_FORMULATE	array	aFormulate

- Matrix control (tabular data)

control type	value	control arguments
CTL_TABULAR	matrix	iMinR, iMaxR, asRow, asCol, aCtlList

Because the control value is contained in the matrix argument following CTL_TABULAR, the array of controls is defined differently from that in the dialog as it has only zero or one 'arguments':

control type	control arguments
CTL_CHECK	
CTL_INT	
CTL_INTRANGE	{ int:min, int:max }
CTL_INTRANGE	{ int:min, int:max, int:step }
CTL_RADIOBOX	string/array
CTL_SELECT	string/array
CTL_DOUBLE	
CTL_DATE	
CTL_MATRIX	{ }
CTL_MATRIX	{ iMinR, iMaxR, iMinC, iMaxC }
CTL_STRING	
CTL_STRMAT	
CTL_SAMPLEINDEX	{ year1, per1, year2, per2, freq }
CTL_DATEINDEX	array of dates
CTL_DATEINDEX	{ dDayMin, dDayMax }
CTL_LISTBOX	$1\times \leq s$ 0-1 selection matrix

CTL_STRING, CTL_MATRIX, CTL_STRMAT, and CTL_LISTBOX are only allowed with CTL_ARRAY and CTL_KWARRAY, not with CTL_TABULAR or OP_MATRIX_CTL.

OxAppGet

```
#include <oxapp.oxh>
"OxAppGet"(const sType [, const sDbName, const asVars]);
```

sType	in:	string, type of data to obtain from OxMetricsk
sDbName	in:	string, database name (required if sType equals "DbInfo", "DbDates", "DbVariable" or "DbVarMatrix")
asVars	in:	array with k_d strings, variable names (optional if sType equals "DbVariable" or "DbVarMatrix", ignored otherwise)

Return value

NB.1 that OxMetrics 10 does no longer maintain an active database.

NB.2 OxAppGet *cannot be used in a* for/foreach *loop, because canonical loops cannot be suspended and resumed. Instead use a* while *loop or* do while.

The following relate to the open databases in OxMetrics.

sType	returns
"DbDates"	$T_d \times 1$ vector with dates if the named database is dated, otherwise <>
"DbInfo"	information on named database, keyed array with: name, path, variables, frequency, sample, isDated, size, varcount
"DbNames"	array with d strings, names of databases loaded in OxMetrics
"DbVariable"	a variable from the named database: array with data matrix ($T_d \times k_d$) and 5 integers: sample of variable (year1, period1, year2, period2, frequency)
"DbVarMatrix"	a data matrix from the named database: $T_d \times k_d$ matrix with entire database content or named variables

OxAppQuery

```
#include <oxapp.oxh>
"OxAppQuery"(const sDialog, const sTitle, ...);
```

sDialog	in:	string, dialog identifier
sTitle	in:	0 for default title, or string with dialog title
...	in:	arguments differ by dialog type

Return value

If OK is pressed: array with labels and values in alternating order, so each value can be accessed as *return[label]*. Returns 0 if the dialog is cancelled.

Description

The following sDialog dialogs are defined:

sDialog	description
"OP_ARRAY"	Array editor, optionally with controls for rows
"OP_KWARRAY"	Keyword-array editor, optionally with controls for rows
"OP_FILE_OPEN"	File Open dialog
"OP_FILE_SAVE"	File Save dialog
"OP_FORMULATE"	Model formulation dialog
"OP_FORMULATE_CODE"	Code formulation dialog
"OP_FORMULATE_NODB"	Model formulation dialog without database
"OP_FUNCTIONS"	Functions
"OP_MATRIX"	Matrix editor
"OP_MATRIX_CTL"	Matrix editor with controls for columns
"OP_MESSAGE"	Message box
"OP_PROGRESS"	Progress dialog (2DO)
"OP_TEXT"	Text editor
"OP_YESNO"	Yes/No Message box

Note some changes with Ox 10:

1. The return value is no longer received in the last argument. The return value is zero for cancel, otherwise dialog specific.

2. OxMetrics does not have an active database anymore. Instead, this is tracked by the application. System formulation is alo entirely devolved to the app now.
3. The following can be replaced by OxAppDialog: "OP_EQUATIONS", "OP_EQUATIONS_SEM"
4. The lag argument to "OP_FORMULATE_NODB" is no longer optional.
5. The Ox code is now responsible for tracking the data source. The relevant dialogs have a mechanism to supply this.
6. "OP_FORMULATE_CODE" has an additional (optional) options argument.
7. For "OP_SELECTVAR" use "OP_FORMULATE" possibly with option "dbfixed": TRUE.
8. "OP_FUNCTIONS" can be replaced by "OP_MATRIX_CTL" or OxAppDialog.

"OP_ARRAY", "OP_KWARRAY"

```
"OxAppQuery"("OP_ARRAY", sTitle, sMsg, arrayVal
    [, asX, aCtlList, sComment]);
"OxAppQuery"("OP_KWARRAY", sTitle, sMsg, arrayVal
    [, 0, aCtlList, sComment[]);
```

sMsg	in:	string, message (e.g. brief explanation)
arrayVal	in:	array
asX	in:	array of strings with names of variables
aCtlList	in:	{} or array$[1]$ or array$[m]$, row types (boolean, integer, double, string, listbox, or matrix, see CTL_TABULAR)
sComment	in:	(optional) string, additional comment
Returns		array or 0 if dialog was cancelled

Example:

```
ret = OxAppQuery("OP_ARRAY", "Fixed size array", "Message text",
    {0,1,range(0,20,"a"), {2.1,2.2,2.3},<0,1,2>,.NaN});
```

If aCtlList is missing, the types are derived from the values in the array. In that case, dimensions and types cannot be changed, but values can be edited.

If aCtlList is specified, they should give a control type for each element in the (flattened) array. If aCtlList is an array with a single entry, it is applied to every row.

"OP_FILE_OPEN"

```
"OxAppQuery"("OP_FILE_OPEN", sTitle, sName);
```

sName	in:	strings, default name, can have "*" for name or for extension. Multiple extensions must be separated by a semicolon (e.g. "*.ox;*.oxh").
returns		string, name of selected file

"OP_FILE_SAVE"

```
"OxAppQuery"("OP_FILE_SAVE", sTitle, sName);
```

sName	in:	strings, default name, can have "*" for name or for extension.
returns		string, name of file to save to

"OP_FORMULATE"

```
"OxAppQuery"("OP_FORMULATE", sTitle, aSystem, options);
```

`aSystem`	in:	array, initial specification
`options`	in:	keyed array with further options possible keys: `"source"`, `"history"`
returns		0 or keyed array

When accepted, OxApp will call `SetModelSettings` with the previous model's settings (if any). This allows a model to be recalled from history.

Example:

```
fok = "OxAppQuery"("OP_FORMULATE", 0, SendSpecials(),
    SendVarStatus(), 2, 1, &dlgout);
```

"OP_FORMULATE_CODE"

```
"OxAppQuery"("OP_FORMULATE_CODE", sTitle, sMsg,
    sCode, options);
```

`sMsg`	in:	string, message (e.g. brief explanation)
`sCode`	in:	string, initial code
`options`	in:	(optional) keyed array with further options possible keys: `"data"`, `"source"`, `"description"`
returns		0 or keyed array

The return value has the `code` and `source` field if the options contain the `data` key. If not, the data sources are the open databases in OxMetrics, and the return value also has `name` (database), `sample` (year1, period1, year2, period2, frequency), `isDated` (TRUE ort FALSE), `variables` (array of names) and `data` (the data matrix). The returned variables consist of all that were found in the code. The first variable is dates for a dated database.

"OP_FORMULATE_NODB"

```
"OxAppQuery"("OP_FORMULATE_NODB", sTitle, asSpecials,
    asStatus, iLagMode, iLag, asDb, iFreq, m);
"OxAppQuery"("OP_FORMULATE_NODB", sTitle, aSystem,
    options);
```

asSpecials	in:	array of strings, list of special variables, see Modelbase::SendSpecials
asStatus	in:	array$[s]$, see Modelbase::SendVarStatus
iLagMode	in:	int, -1: no lags allowed; ≥ 0: default for first lags drop down box: 0=None, 1=Lag, 2=Lags 0 to)
iLag	in:	int, ≥ 0: default for lag value box
asDb	in:	array of strings, database variables,
iFreq	in:	(optional) int, ≥ 1: database frequency
m	in:	(optional) matrix, current formulation, see under *returns*
returns		0 or $(3+s) \times k$ matrix, where k is the number of selected variables: $[0][i]$: -1 or index in database $[1][i]$: index in specials or -1 $[2][i]$: lag length $[3+j][i]$: 1 if variable i has status j
aSystem	in:	array with system formulation (or for empty)
options	in:	(optional) keyed array with further options required keys: "specials", "status", "lag", "lagmode" possible keys: "data", "source", "frequency", "variables", "history"
returns		0 or the the formulated system

Formulation dialog which does not use the OxMetrics database.

"OP_FUNCTIONS"

```
"OxAppQuery"("OP_FUNCTIONS", sTitle, asX, aasFunc,
    aReturn);
```

asX	in:	array of strings, list of k X variables (the current database selection is used if this list is empty),
aasFunc	in:	array of array of strings, see Modelbase::SendFunctions
returns	in:	address of variable
	out:	array of strings with function calls, each of format "func(name,arg1,arg2)", where func is the function, name is a string and arg1, arg2 are integers.

"OP_MATRIX"

```
"OxAppQuery"("OP_MATRIX", sTitle, sMsg, mMat, ...);
"OxAppQuery"("OP_MATRIX", sTitle, sMsg, mMat,
    iRmin, iRmax, iCmin, iCmax, iType, asRow, asCol,sComment);
```

sMsg	in:	string, message (e.g. brief explanation)
mMat	in:	$n \times m$ matrix
iRmin	in:	int, minimum row count
iRmax	in:	int, maximum row count
iCmin	in:	int, minimum column count
iCmax	in:	int, maximum column count
iType	in:	(optional) int, type, currently unused (set this to 0)
asRow	in:	(optional) array of strings with row labels, or string with row format
asCol	in:	(optional) array of strings with columns labels, or string with column format
sComment	in:	(optional) string, additional comment
Returns		$p \times q$ matrix or 0 if dialog was cancelled

"OP_MATRIX_CTL"

```
"OxAppQuery"("OP_MATRIX_CTL", sTitle, sMsg, mMat,
    iRmin, iRmax, asRow, asCol, aCtlList);
```

sMsg	in:	string, message (e.g. brief explanation)
mMat	in:	$n \times m$ matrix
iRmin	in:	int, minimum row count
iRmax	in:	int, maximum row count
asRow	in:	array of strings with row labels, or string with row format
asCol	in:	array of strings with columns labels
aCtlList	in:	array$[m]$, column types (boolean, integer or double, see CTL_TABULAR)
Returns		$p \times m$ matrix or 0 if dialog was cancelled

Example:

```
m = "OxAppQuery"("OP_MATRIX_CTL", "Test",
    "Message", m, 0, 100, "R %d",
    {"check","int","range","dbl", "choice", "sampleindex", "date"},
    {CTL_CHECK, CTL_INT, CTL_INTRANGE, {0, 10}, CTL_DOUBLE,
        CTL_SELECT, "outlier|irregular|slope",
        CTL_SAMPLEINDEX, selsam, CTL_DATE});
```

"OP_MESSAGE"

```
"OxAppQuery"("OP_MESSAGE", sTitle, sMsg, sLine);
```

sMsg	in:	string, message test
sLine	in:	(optional) string, additional short message
returns	in:	address of variable
	out:	unused

"OP_PROGRESS"

```
"OxAppQuery"("OP_PROGRESS", sTitle, sMsg);
```

sMsg	in:	string, message (e.g. brief explanation), use 0 for none
returns	in:	address of variable
	out:	unused, progress is printed when OK is clicked.

Example:

```
"OxAppQuery"("OP_PROGRESS", 0, 0, &dlgout);
```

"OP_TEXT"

```
"OxAppQuery"("OP_TEXT", sTitle, sMsg, sCode);
"OxAppQuery"("OP_TEXT", sTitle, sMsg, sCode, iShow, sDesc);
```

sMsg	in:	string, message (e.g. brief explanation)
sCode	in:	string, initial code
iShow	in:	int, 0: unused
sDesc	in:	string, description
returns	in:	address of variable
	out:	string, the new code

"OP_YESNO"

```
"OxAppQuery"("OP_YESNO", sTitle, sMsg, sLine);
```

sMsg	in:	string, message test
sLine	in:	(optional) string, additional short query
returns	in:	address of variable
	out:	unused

OxAppSet

```
#include <oxapp.oxh>
"OxAppSet"(const sType, const arg, const options);
```

No return value.

Description

Sets data in OxMetrics.

"activate" Activates the output window.

"batch" Blocks the next batch command from the app until ready.

arg	in:	string, "running" or "ready"
options	in:	int, return value when batch is "ready"

"BatchCode" Sets the generated batch code.

arg	in:	string, Batch code

"menu" Sets the menu content.

arg	in:	string, "Model", "ModelClass", "Test"
options	in:	array, return value from SendMenu

"model" The argument is usually supplied by GetModelSettings:

arg	in:	keyed array with "label": string or array of string, with name(s) for vectors "system": system formulation "source": string, database name "settings": model settings

"OxCode" Sets the generated Ox code.
`arg` in: string, Ox code
"OxRun" Runs an Ox program.
`arg` in: string, Ox program to run
"settings" Controls wait indicators in OxMetrics.
`arg` in: string:
"running": shows the app and output window as running and blocks menu calls
"busy": only shows the app as running, blocking meny calls
"ready": app becomes ready again
"store" Stores variables in an OxMetrics database.
`arg` in: $T \times k$ data matrix to store in database
`options` in: keyed array with
"name": string or array of string, with name(s) for vectors
"index": t1, index of first observation in database (0 if missing)
"source": string, database name
"query": TRUE to confirm names in OxMetrics

D2.5 Modelbase virtual functions for OxApp

The default menu structure in Modelbase is defined through SendMenu:

```
Modelbase::SendMenu(const sMenu)
{
    if (sMenu == "Model")
    {
        return
        {   { "&Formulate...\tAlt+Y",        "OP_FORMULATE"},
            { "Model &Settings...\tAlt+S",   "OP_SETTINGS"},
            { "&Estimate...\tAlt+L",         "OP_ESTIMATE"},
            0,
            { "&Progress...",                "OP_PROGRESS"},
            0,
            { "&Options...\tAlt+O",          "OP_OPTIONS"}
        };
    }
    else if (sMenu == "Test")
    {
        return
        {   { "&Exclusion Restrictions...", "OP_TEST_SUBSET"},
            { "&Linear Restrictions...",    "OP_TEST_LINRES"},
            { "&General Restrictions...",   "OP_TEST_GENRES"}
        };
    }
    return 0;
}
```

This default does not use model classes to distinguish between different types of sub-models.

Running a Modelbase package from OxApp involves the following calls:

1. OxApp starts package, calling:
   — constructor
   — SendMenu("ModelClass") Model class items on Model menu
   — SendMenu("Model") Model menu items (below model class items)
   — SendMenu("Test") Test menu items
   — BatchCommands() Query for batch commands

2. User selects a menu item:
   — ReceiveMenuChoice(id_string)

3. User selects the "OP_FORMULATE" item:
   — ReceiveMenuChoice("OP_FORMULATE") call DoFormulateDlg(2), to initialize:
   — SendSpecials()
   — SendVarStatus()
   — GetModelSettings() show formulation dialog, if accepted:
   — SetModelSettings() sets model and model data

4. User selects the "OP_SETTINGS" item:
   — ReceiveMenuChoice("OP_SETTINGS") calling DoSettingsDlg()

5. User selects the "OP_ESTIMATE" item:
   calling DoEstimateDlg()
   — Estimate() if this is successful:
   — sets estimation sample
   — GetModelSettings()
   — GetBatchCode()
   — GetOxCode()

6. Running batch code from OxMetrics, calling:
   to prepare to receive the model formulation:
   — SendSpecials()
   — SendVarStatus()
   to receive the batch command:
   — Batch()
   to receive the estimate command:
   — SetSelSample() or SetSelSampleByDates()
   — SetForecasts()
   — SetRecursive()
   — BatchMethod()

Modelbase::DoEstimateDlg

```
virtual DoEstimateDlg(const iFirstMethod, const cMethods,
    const sMethods, const bForcAllowed, const bRecAllowed,
    const bMaxDlgAllowed);
```

iFirstMethod	in:	int, index of first method (often 0),
cMethods	in:	int, number of estimation methods (use zero to skip method selection),
sMethods	in:	string, estimation methods, separated by |,
bForcAllowed	in:	int, TRUE if observations can be withheld at this stage for forecasting,
bRecAllowed	in:	int, TRUE if recursive estimation is allowed,
bMaxDlgAllowed	in:	int, TRUE if the maximization dialog (i.e. non-automatic) is allowed.

Return value

Returns 1 if dialog accepted by user, 0 for cancel or error, -1 to return to formulation

Description

Creates and shows a sample selection dialog for estimation.

Example:

```
if (Modelbase::DoEstimateDlg(0, 2, "Newton's method|BFGS method",
    FALSE, FALSE, FALSE))
{
    if (m_iMethod == 0)
        m_fnMax = MaxNewton, m_sMax = "Newton";
    else
        m_fnMax = MaxBFGS, m_sMax = "BFGS";
```

```
        return TRUE;
    }
    return FALSE;
```

Modelbase::DoFormulateDlg

```
virtual DoFormulateDlg(const iLagMode);
virtual DoFormulateDlg(const iLagMode, const iLagDefault);
```

iLagMode	in:	int, -1: no lags allowed; ≥ 0: default for first lags drop down box: 0=None, 1=Lag, 2=Lags 0 to)
iLagDefault	in:	int, ≥ 0: default for lag value box

Return value

Returns TRUE if the users accepts the dialog.

Description

Calls the OP_FORMULATE dialog using SendSpecials and SendVarStatus.

Modelbase::DoOption, Modelbase::DoOptionsDlg

```
virtual DoOption(const sOpt, const val);
virtual DoOptionsDlg(const aMoreOptions);
```

sOpt	in:	string, label of option
val	in:	value of option
aMoreOptions	in:	array, dialog options that are to be appended to the Modelbase default.

Return value

DoOptionsDlg returns TRUE if the users accepts the dialog.

Description

Creates and shows an options dialog. Options that are appended using the aMoreOptions argument are processed by calls to DoOption, which therefore must be overridden in that case.

Modelbase::DoSettingsDlg

```
virtual DoSettingsDlg();
```

Return value

Returns TRUE if the users accepts the dialog.

Description

The settings usually relate to the model settings, such as ARMA or GARCH order, etc. The Modelbase default does not display a dialog, but just returns TRUE.

Modelbase::ForceYlag

```
ForceYlag(const iYgroup);
```

iYgroup	in:	int, identifier of group to adjust, or: array[2], identifier of group to adjust, followed by group to change from

No return value.

Description

By default, lagged dependent variables in modelbase are not classified as `Y_VAR` but as the default regressor type. Use `ForceYlag(Y_VAR)` to change lags to `Y_VAR`, which is the convention used in most Modelbase derived packages to facilitate dynamic analysis.

Or use (e.g.) `ForceYlag( {Y_VAR, X_VAR} )` to only change lagged Y's that are `X_VAR` to `Y_VAR`.

Modelbase::GetModelSettings

```
virtual GetModelSettings();
```

Return value

Returns a keyed array with labels and values.

Description

Called by OxApp after successful estimation, to get model settings for the model history. This allows model parameters to be recalled together with the model specification.

Modelbase::ReceiveMenuChoice

```
virtual ReceiveMenuChoice(const sDialog);
```

`sDialog` in: string, menu command identifier

Return value

Returns 1 if successful, 0 otherwise.

Description

Called by OxApp when the user selects a menu item.

Modelbase::SendSpecials

```
virtual SendSpecials();
```

Return value

Returns 0 if there are no special variables. Returns an array of strings listing the special variables otherwise.

Description

Used by Modelbase as part of model formulation, after `SendVarStatus`, to determine the content of the special variables listbox in the model formulation dialog. The default implementation returns {`"Constant", "Trend", "Seasonal"`}.

Modelbase::SendVarStatus

```
virtual SendVarStatus();
```

Return value

Returns an array, where each item is an array defining the type of variable:

1. string: status text,
2. character: status letter,
3. integer: status flags,
4. integer: status group.

Description

Called by Modelbase as part of model formulation, to determine the variable types which are available in the model formulation dialog. For example, the Modelbase default is:

```
return
    {{ "&Y variable", 'Y', STATUS_GROUP + STATUS_ENDOGENOUS, Y_VAR},
     { "&X variable", 'X', STATUS_GROUP, X_VAR}};
```

The status text, and is used on the data selection dialog button. The status letter used to indicate the presence of the status. The status flags can be:

- STATUS_DEFAULT: is default: no status letter displayed;
- STATUS_ENDOGENOUS: apply to first (non-special) variable at lag 0;
- STATUS_GROUP : is a group (each variable is in only one group);
- STATUS_GROUP2: is a second group (each variable is only in one of each group);
- STATUS_GROUP3: is a third group (each variable is only in one of each group);
- STATUS_GROUP4: is a fourth group (each variable is only in one of each group);
- STATUS_GROUP5: is a fifth group (each variable is only in one of each group);
- STATUS_MULTIPLE: multiple instances of a variable are allowed
- STATUS_MULTIVARIATE: apply to all (non-special) variables at lag 0;
- STATUS_ONEONLY: only one variable can have this status.
- STATUS_SPECIAL: apply to all special variables;
- STATUS_TRANSFORM: is a transformation;

Some flags can be combined by adding the values together.

As a second example, consider the status definitions of the DPD class:

```
return
  {{ "&Y variable", 'Y', STATUS_GROUP + STATUS_ENDOGENOUS, Y_VAR},
   { "&X variable", 'X', STATUS_GROUP, X_VAR},
   { "&Instrument", 'I', STATUS_GROUP2, I_VAR},
   { "&Level instr",'L', STATUS_GROUP2, IL_VAR},
   { "Yea&r",       'r', STATUS_GROUP + STATUS_ONEONLY, YEAR_VAR},
   { "I&ndex",      'n', STATUS_GROUP + STATUS_ONEONLY, IDX_VAR}
  };
```

Modelbase::SetModelSettings

```
virtual SetModelSettings(const aValues)
```

aValues	in:	if array:keyed array with labels and values.

No return value.

Description

Called by OxApp to set model settings. This is called immediately after model formulation, before model settings, to inherit default settings from the previous model, or the model that was recalled from history.

D2.6 Adding support for a Batch language

Modelbase::Batch

```
virtual Batch(const sBatch, ...);
```

Table D2.1 Batch commands (partially) handled by OxApp

`estimate("`*METHOD*`"=”OLS”,`*YEAR1*`=-1,`*PER1*`=0,` *YEAR2*`=-1,`*PER2*`=0,`*FORC*`=0,`*INIT*`=0) ;`
`model { ... }`
`nonlinear { ... }`
`progress;`
`system { ... }`

`sBatch`	in:	a string with name of the batch command
...	in:	zero or more batch arguments

Return value

Should return `TRUE` if the batch command was correct, `FALSE` if there was a syntax error.

Description

All Batch commands are directly passed to the Ox class, with the exception of those listed in Table D2.1.

The arguments of the batch command are passed separately. For example, when the batch call is

```
test("ar", 1, 2);
```

this function is called as

```
Batch("test", "ar", 1, 2);
```

Note that batch commands can have a variable number of arguments, so

```
test("ar", 1);
```

is a valid call, and the Ox class should use default values for the missing arguments.

nonlinear Stored in OxApp, before passed on in the call to `Batch("nonlinear",` code`)`.

progress Activates the `"OP_PROGRESS"` item on the model menu.

system This is handled by`Modelbase::Batch`. Calls to `SendSpecials`, `SendVarStatus` to determine the special variables, status of variable status for model formulation. The return value contains the model formulation and data.

testres Calls `TestRestrictions` with an array of strings as argument.

testlinres Calls `TestRestrictions` with two arguments: the matrix R' and the column vector r.

Modelbase::BatchCommands

```
virtual BatchCommands();
```

Return value

Should return an array of strings with batch commands.

Description

If this function does not exist, only a few predefined commands will work (`system`, `estimate`, `progress`; `estimate` will also require `BatchMethod` if there is more than one method).

Here is an example from PcGive:

```
PcGive::BatchCommands()
{
    return
    {   "adftest(\"VAR\", LAG, DETERMINISTIC=1, SUMMARY=1);",
        "arorder(AR1, AR2);",
        "comfac;",
        "cointcommon { };",
        "cointknown { };",
        "constraints { };",
        "derived { };",
        "dynamics;",
        "encompassing;",
        "forecast(NFORC, HSTEP=0, SETYPE=1);",
        "option(\"OPTION\", ARGUMENT);",
        "output(\"OPTION\", VALUE=1);",
        "rank(RANK);",
        "store(\"WHAT\", \"RENAME\"=\"\");",
        "test(\"TYPE\"=\"\", LAG0=0, LAG1=0);",
        "testlinres { }",
        "testgenres { }",
        "testres { }",
        "testsummary"
    };
}
```

Modelbase::BatchMethod

```
virtual BatchMethod(const sMethod);
```

sMethod	in:	a string with the first argument of the `estimate` batch command

Return value

Should return the index of the method type.

Description

This function is called immediately after processing the `estimate` batch command. When writing batch code, OxApp uses the return value from GetMethodLabel() to determine the first argument of `estimate`. Therefore, the input argument should match the possible return values of `GetMethodLabel()`, and the return value the index.

Modelbase::GetBatchEstimate, Modelbase::GetBatchModelSettings

```
virtual GetBatchEstimate();
virtual GetBatchModelSettings();
```

Return value

The required batch code as a string.

Description

When the Batch editor is opened, it will show the batch code for the estimated model. OxApp writes the `package`, `usedata` and `system` (or `nonlinear`) parts, then calls `GetBatchModelSettings` and `GetBatchEstimate` for the rest.

Chapter D3

The ox/dev folder

The ox/dev folder contains header and library files the facilitate the link between Ox and the foreign language. Examples can be found in ox/dev/samples.[1]

The main header file to use in your C/C++ code is oxexport.h. This in turn imports the following header files:

dependencies of oxexport.h	
jdsystem.h	platform and compiler specific defines
jdtypes.h	basic types and constants
jdmatrix.h	basic matrix services
jdmath.h	mathematical and statistical functions
oxtypes.h	basic Ox constants and types

An Ox class is provided for the following languages:

Ox.cs	C#
Ox.java	Java
Ox.vb	Visual Basic 9

The remaining sections all give examples on extending Ox:

[1] The Unix convention of forward slashes is adopted.

purpose	`ox/` directory	section
calling C code from Ox	`dev/samples/threes`	D4.1
calling Ox exported functions from C	`dev/samples/threes`	D4.1
calling Fortran code from Ox	`dev/samples/fortran`	D4.2
returning values in arguments	`dev/samples/invert`	D4.3
calling Ox matrix functions from C	`dev/samples/invert`	D4.3
calling Ox code from C	`dev/samples/callback`	D4.4
using Ox arrays in C	`dev/samples/array`	D4.5
using Ox as a library from C	`dev/samples/oxlib`	D5.2.1
using Ox as a library from C#	`dev/samples/oxlib`	D5.2.3
using Ox as a library from Java	`dev/samples/oxlib`	D5.2.2
using Ox objects from C	`dev/samples/object`	D5.3
using Ox objects from C#	`dev/samples/object`	D5.3
using Ox objects from Java	`dev/samples/object`	D5.3.1
writing a Java interface	`dev/windows/ranapp`	**??**
writing a Visual C++ interface	`dev/windows/ranapp`	**??**

Chapter D4

Ox Foreign Language Interface

D4.1 Calling C code from Ox

The objective is to write a function in C, compile it into a dynamic link library (DLL), so that the function can be called as if it were part of Ox.

We shall write a function called `Threes`, which creates a matrix of threes (cf. the library function `ones`). The first argument is the number of rows, the second the number of columns. It could be used in Ox code to create a 2×3 matrix of filled with the value 3, e.g.:

```
decl x = Threes(2, 3);
```

The C source code is in `threes.c`:

ox/dev/samples/threes/threes.c

```
#include "oxexport.h"

void OXCALL FnThrees(OxVALUE *rtn, OxVALUE *pv, int cArg)
{
    int i, j, c, r;

    OxLibCheckType(OX_INT, pv, 0, 1);

    r = OxInt(pv, 0);
    c = OxInt(pv, 1);
    OxLibValMatMalloc(rtn, r, c);

    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            OxMat(rtn, 0)[i][j] = 3;
}
```

- The `oxexport.h` header file defines all types and functions required to link to Ox.
- All functions have the same format:
  - `OXCALL` defines the calling convention;
  - `rtn` is the return value of the function. It is a pointer to an `OxVALUE` which is the container for any Ox variable. On input, it is an integer (`OX_INT`) of value 0.

If the function returns a value, it should be stored in `rtn`.
  - `pv` is an array of `cArg` `OxVALUE`s, holding the actual arguments to the function.
  - `cArg` is the number of arguments used in the function call. Unless the function has a variable number of arguments, there is no need to reference this value.
- If the function is written in C++ instead of C, it must be declared as:

```
extern "C" void OXCALL FnThrees
    (OxVALUE *rtn, OxVALUE *pv, int cArg)
```

- First, we check whether the arguments are of type `OX_INT` (we know that there are two arguments, which have index 0 and 1 in `pv`). The call to `OxLibCheckType` tests `pv` (the function arguments) from index 0 to index 1 for type `OX_INT`.

  Arguments must be checked for type before being accessed. Make sure there is a call to `OxLibCheckType` *for each argument (unless you inspect the arguments 'manually'). This check also does type conversion if that is required.*

  In this case, a double would also be valid, but automatically converted to an integer by the `OxLibCheckType` function. Any other argument type would result in a run-time error (checking for the number of arguments is done at compile time).
  The most commonly used types for Ox variables are:

`OX_INT`	integer
`OX_DOUBLE`	double
`OX_MATRIX`	matrix
`OX_STRING`	string
`OX_ARRAY`	array

- For convenience, we copy the first argument to `r`, and the second to `c`. `OxInt` accesses the integer in an `OxVALUE`. The first argument is the array of `OxVALUE`s, the second argument is the index in the array. This specifies the dimension of the requested matrix.
- The return type is a matrix, and that matrix has to be allocated in the `rtn` value, using the right dimensions. This is done with the `OxLibValMatMalloc` function. A run-time error is generated if there is not enough memory to allocate the matrix.
- Finally we have to set all elements of the matrix to the value 3. `OxMat` accesses the allocated matrix. The dimensions of that matrix are accessed by `OxMatc`, `OxMatr`, but here we already know the dimensions.

Note that the function arguments, as contained in `pv`, may only be changed if they are declared as `const`. *It is best to never change the arguments in the function*, except from conversion from int to double and vice versa (automatic conversion using `OxLibCheckType` is always safe). Another exception is when the argument is a reference to a variable in which the caller expects a return value. An example will follow shortly.

D4.1.1 Calling the Dynamic Link Library

After creating the DLL, the function can be used as follows:

ox/dev/samples/threes/threes.ox

```
#include <oxstd.h>
```

```
extern "threes,FnThrees" Threes(const r, const c);

main()
{
    print(Threes(3,3));
}
```

The function is declared as `extern`, with the DLL file in `threes` (the name may contain a relative or absolute path). The name of the function in `threes.dll` is `FnThrees`, but in our Ox code we wish to call it `Threes`. After this declaration, we can use the function `Threes` as any other standard library function (normally this would be in a header file).

Note that DLLs from different platforms can coexist in the same folder, because the Ox will first try the platform-specific version:

`threes.dll`	Windows 64-bit
`threes.so`	Linux 64-bit
`threes.dylib`	macOS 64-bit (fat binary: Apple silicon and Intel)

The language reference chapter in the Ox book has more information under external declarations; path resolution is discussed under preprocessing.

If the program does not work, check the requirements to successfully link to the Ox DLL. Under Windows:

- `OXCALL` corresponds to the standard call (`__stdcall`) calling convention; this pushes parameters from right to left, and lets the function clean the stack;
- the function is exported, and its name is not decorated.
  Make sure that `FnThrees` is the exact name in the DLL file; some compilers will change the name according to the calling convention (and C++ functions are subject to name mangling, which is avoided by declaring them as extern "C").

D4.1.2 Compiling `threes.c`

The `threes.c` file should compile without problems into a DLL file. Makefiles for a range of compilers are provided:

- `threes/lnx_gcc` — 64-bit linux using gcc
  The Linux versions do not need to specify the exported functions, nor is the Ox `.so` file needed when linking: imports are resolved at run time. See §D4.1.2.2.
- `threes/mac_clang` — macOS using clang (the default compiler)
  See §D4.1.2.3.
- `threes/win_gcc` — 64-bit Windows using MinGW gcc
  The 64-bit version links to `ox/dev/libwin/liboxwin.a`. The `threes.def` file handles the name decoration. Output is in the local folder, to keep it separate from the DLL compiled with Visual C++. More information is in §D4.1.2.4.
- `threes/win_vc` — 64-bit Windows using Microsoft Visual C++
  The path to `oxexport.h` is specified in the project file. The 64-bit version links to `ox/dev/libwin/oxwin.lib`. The exports of the DLL are defined in `threes.def`, but you could use `__declspec(dllexport)` instead. The `threes.dll` file is output to the `dev/threes` folder. More information is in §D4.1.2.1.
- `threes/win_clang` — 64-bit Windows using clang via Microsoft Visual Studio

Note the calling conventions mentioned above, which matters only under Windows. A library file specifies the imported functions, while the definition file to resolve the calls to the Ox DLL (again, Windows only).

D4.1.2.1 Windows: Microsoft Visual C++

Microsoft Visual C++ is part of Visual Studio.

The Microsoft Visual Studio solution (.sln) and project (.vcxproj) files can be found in the folder entitled: `ox/dev/samples/threes/win_vc`. The project sets the additional include directories (project properties, C/C++, General) to `..\..\..\` (this will resolve to ox/dev where the oxexport.h and other header files are). The library path (Linker, General), is set to `..\..\..\lib`. Finally, `oxwin.lib` is linked in (Linker/Input), and a .def file used to specify the exports: `../threes.def` (Linker, General).

The 64-bit release version creates `threes.dll` in the `ox/dev/threes` folder.

D4.1.2.2 Linux: gcc

Compiling the threes example works under Linux as well, but this time a make file is used: (`dev/samples/threes/lnx_gcc/threes.mak`). This is compiled by executing the command[1]

```
make
```

which creates threes.so. The header file `oxexport.h` and dependencies must be in the search path. Then run from the threes folder:

```
oxl threes
```

to see if it works. The dynamic linker must be able to find threes.so, as discussed in the Unix installation notes.[2] Unix platforms do not use name decoration of C functions.

To run the program, use OxEdit, or OxMetrics, or from the command line:

```
oxl threes
```

D4.1.2.3 macOS: clang

Under the hood, macOS is a type of Unix. Therefore, developers can use terminal windows and makefiles just as in Linux. There are a few differences:

- Instead of creating separate arm64 and x64 bit binary files, it is possible to create so-called fat binaries (or universal binaries), which hold both versions. At load time, the appropiate version is used. The compiler command-line argument for this is
  ```
  -arch arm64 -arch x86_64
  ```
  Note that the 32-bit architecture is no longer supported.
- It is necessary to install the compiler. Xcode is the interactive compiler, which can be obtained from the Apple Developer (it is actually downloaded from the Mac App store). Next, the command-line tools must be installed. Now `clang` can be used in a terminal window.

[1] The `threes` folder is in `/usr/share/OxMetrics10/ox/dev/samples/` which requires root access; you may wish to copy this to your home folder.

[2] Try `oxl -v2 threes.ox` if it does not work. This will show which paths are tried.

Open a terminal window and compile threes by executing the command

```
make
```

in the appropriate threes folder.[3] This creates threes.dylib. The header file `oxexport.h` and dependencies must be in the search path. Then load `threes.ox` in OxEdit, and run it to check if it worked.

D4.1.2.4 Windows: MinGW (gcc)

The threes example for the MinGW compiler (`www.mingw.com`) is provided in `ox/dev/samples/threes/win_gcc`. This uses the presupplied liboxwin.a file to link to the Ox DLL.

- Open a console window (Command prompt or MSYS) and locate the `threes/win_gcc` folder.
- Assuming that the paths are in your environment settings, type make to compile the `Makefile` (delete the `.dll` and `.o` files first to enforce recompilation).
- `oxl threes.ox` to check the compiled DLL.
- `ox/dev/liboxwin.a` is the import library for the Ox DLL, created with dlltool using the `--export-all --kill-at` arguments, because in MinGW stdcall functions have no _ prefix but do use the `@nn` suffix. Note that `jdsystem.h` has a `__MINGW32__` section that defines `JDCALL` (and therefore `OXCALL`) to `__stdcall` and JDCALLC to `__cdecl`.

The makefile for the 64-bit Mingw compiler targeting 64-bit Windows is in `ox/dev/samples/threes/win_gcc`. This links to `ox/dev/libwin/liboxwin.a` and does not use name deoration. Note that `__MINGW32__` is also defined in Mingw64.

D4.1.2.5 Name decoration

Some Windows compilers may use different name decoration for exported functions. The `oxwin.dll` which contains the Ox runtime exports undecorated names, which works fine with Visual C++.

[3] The `threes` folder is in `/Applications/OxMetrics10/ox/dev/samples/` which requires root access; you may wish to copy this to your Users folder.

D4.2 Calling FORTRAN code from Ox

Linking Fortran code to Ox does not pose any new problems, apart from needing to know how function calls work in Fortran. Under Windows, this requires knowledge about the function call type and the name decorations. Under Linux or Unix this tends to be irrelevant. The simplest solution is to write C wrappers around the Fortran code, and use a Fortran and C compiler from the same vendor. (An alternative is to use F2C to translate the Fortran code to C.)

Arguments in Fortran functions are always by reference: change an argument in a function, and it will be changed outside the function. For this reason, well-written Fortran code copies arguments to local variables when the change need not be global.

Two examples are provided. The directory `ox/samples/fortran` contains a simple test function in Fortran, and a C wrapper which also provides a function which is called from Fortran. The example uses gcc fortran (one make file is in `fortran/win_gcc`, using MinGW's gfortran, the other is `fortran/lnx_gcc` for 64-bit Linux). but other compilers will also be feasible.

D4.3 Calling C/C++ code from Ox: returning values in arguments

Returning a value in an argument only adds a minor complication. Remember that by default all arguments in Ox and C are passed by value, and assignments to arguments will be lost after the function returns. To return values in arguments, pass a reference to a variable in the Ox call, so that the called function may change what the variable points to.

To refresh our memory, here is some simple Ox code:

```
#include <oxstd.h>

func1(a)
{   a = 1;
}
func2(const a)
{   a[0] = 1;
}
main()
{
    decl b;

    b = 0;  func1(b);   print(b);
    b = 0;  func2(&b);  print(b);
}
```

This will print `01`. In `func1` we cannot use the `const` qualifier because we are changing the argument. In `func2` the argument is not changed, only what it points to.

The first serious example is the `invert` function from the standard library, which also illustrates the use of a variable argument list. The code for this project is in `ox/dev/samples/invert`

ox/dev/samples/invert/invert.c

```
#include "oxexport.h"
#include "jdmath.h"

void OXCALL FnInvert(OxVALUE *rtn, OxVALUE *pv, int cArg)
{
    int  r, signdet = 0;  double logdet = 0;

    OxZero(rtn, 0);

    OxLibCheckSquareMatrix(pv, 0, 0);
    if (cArg == 2)           /* either 1 or 3 arguments */
        OxRunError(ER_ARGS, "invert");
    else if (cArg == 3)
        OxLibCheckType(OX_ARRAY, pv, 1, 2);

    r = OxMatr(pv, 0);
    OxLibValMatDup(rtn, OxMat(pv, 0), r, r);

    if (IInvDet(OxMat(rtn, 0), r, &logdet, &signdet) != 0)
    {
        OxRunMessage("invert(): inversion failed");
```

```
            OxFreeByValue(rtn);
            OxZero(rtn, 0);
        }
        if (cArg == 3)
        {
            OxSetDbl( OxArray(pv,1), 0, logdet);
            OxSetInt( OxArray(pv,2), 0, signdet);
        }
    }
```

- `OxLibCheckSquareMatrix(pv, 0, 0)` is the same as making a call to `OxLibCheckType(OX_MATRIX, pv, 0, 0)` followed by a check if the matrix is square.
- Using `invert` with two arguments is an error. When there are three arguments, the code checks if the second and third are of type `OX_ARRAY`.
- `OxMatr` gets the number of rows in the first argument (we already know that it is a matrix, with the same number of rows as columns).
- Next, we duplicate (allocate a matrix and copy) the matrix in the first argument to the return value using `OxLibValMatDup`. We shall overwrite this with the actual inverse.
- `IInvDet` is an internal mathematics function used by Ox to invert a non-singular matrix. This function is also exported as a C function by the Ox run-time dynamic-link library, which is why we can use it here (provided we add `#include "jdmath.h"`).
- If the matrix inversion fails, the matrix in `rtn` is freed, and `rtn` is changed back to an integer with value 0. It is important to free before setting the value to an integer: otherwise a memory leak occurs.
- Otherwise, but only if the second and third argument were provided, do we put the log-determinant (`logdet`) and sign in those argument. `OxArray(pv,1)` accesses the array at element 1 in `pv`. This is then used in the same way as `pv` was used to access the first entry in that array (index 0).

A more complex example is that for the square root free Choleski decomposition `decldl`, again from the standard library. The first argument is the symmetric matrix to decompose, the next two are arrays in which we expect the function to return the lower triangular matrix and vector with diagonal elements.

```
void OXCALL FnDecldl(OxVALUE *rtn, OxVALUE *pv, int cArg)
{
    int  i, j, r;  MATRIX md, ml;

    OxLibCheckSquareMatrix(pv, 0, 0);
    OxLibCheckType(OX_ARRAY, pv, 1, 2);
    OxLibCheckArrayMatrix(pv, 1, 2, OxMat(pv, 0));

    r = OxMatr(pv, 0);
    OxLibValMatDup(OxArray(pv, 1), OxMat(pv, 0), r, r);
    OxLibValMatMalloc(OxArray(pv, 2), 1, r);
    ml = OxMat( OxArray(pv, 1), 0);
    md = OxMat( OxArray(pv, 2), 0);
```

```
    if (!ml || !md)
        OxRunError(ER_OM, NULL);
    if (ml == md)
        OxRunError(ER_ARGSAME, NULL);

    if ( (OxInt(rtn, 0) = !ILDLdec(ml, md[0], r)) == 0)
        OxRunMessage("decldl(): decomposition failed");

                /* diagonal of ml is 1, upper is 0 */
    for (i = 0; i < r; i++)
    {   for (j = i + 1; j < r; j++)
            ml[i][j] = 0;
        ml[i][i] = 1;
    }
}
```

The new functions here are:

- `OxLibCheckArrayMatrix` which checks that the arrays do not point to the matrix to decompose, as in `decldl(msym, &msym, &md)`.
- `OxLibValMatMalloc` allocates space for a matrix.
- `OxRunError` generates a run-time error message. The statement `if (ml == md)` checks if the arrays do not point to the same variable. If so, we have allocated a matrix twice, but end up with the last matrix for both arguments. This prevents code of the form `decldl(msym, &md, &md)`.

D4.4 Calling Ox functions from C

This section deals with reverse communication: inside the C (or C++) code, we wish to call an Ox function. The example is a numerical differentiation routine written in C, used to differentiate a function defined in Ox code.

ox/dev/samples/callback/callback.c (part of)

```
#include "oxexport.h"

/* ... for FNum1Derivative() see callback.c ... */

static int myFunc(int cP, VECTOR vP, double *pdFunc,
    VECTOR vScore, MATRIX mHess, OxVALUE *pvOxFunc)
{
    OxVALUE rtn, arg;
    /* ensure that the OxVALUEs are initialized */
    OxValSetZero(&rtn);
    OxValSetZero(&arg);

    /* get copy of parameter vector, matrix[1][cP] */
    OxValSetMat(&arg, &vP, 1, cP);

    /* execute the call back */
    if (!FOxCallBack(pvOxFunc, &rtn, &arg, 1))
        return 1;
    /* check the return value of the call back */
    OxLibCheckType(OX_DOUBLE, &rtn, 0, 0);
    *pdFunc = OxDbl(&rtn, 0);

    /* parg received a copy: free to avoid memory leak */
    OxFreeByValue(&arg);

return 0;
}
void OXCALL FnNumDer(OxVALUE *rtn, OxVALUE *pv, int cArg)
{
    int c;  OxVALUE *pvfunc;

    OxLibCheckType(OX_FUNCTION, pv, 0, 0);
    pvfunc = pv;                    /* function pointer */
    OxLibCheckType(OX_MATRIX, pv, 1, 1);

    c = OxMatc(pv, 1);
    OxLibCheckMatrixSize(pv, 1, 1, 1, c);
    OxLibValMatMalloc(rtn, 1, c);

    if (!FNum1Derivative(
        myFunc, c, OxMat(pv, 1)[0], OxMat(rtn, 0)[0], pvfunc))
    {
        OxFreeByValue(rtn);
        OxZero(rtn, 0);
    }
}
```

First we discuss FnNumDer which performs the actual numerical differentiation by calling FNum1Derivative:

- Argument 0 in pv must be a function, argument 1 a matrix, from which we only use the first row (expected to hold the parameter values at which to differentiate).[4]
- OxLibCheckMatrixSize checks whether the matrix is $1 \times$c (since the c value is taken from that matrix, only the number of rows is checked).
- Finally, the C function FNum1Derivative is called to compute the numerical derivative of myFunc. When successful, it will leave the result in the first row of the matrix in rtn (for which we have already allocated the space).

The myFunc function is a wrapper which calls the Ox function:

- Space for the arguments and the return value is required. There is always only one return value: even multiple returns are returned as one array. Here we also have just one argument for the Ox function, resulting in the OxVALUE rtn and arg. We mainly work with pointers to OxVALUEs. The argument is set to a $1 \times$cP matrix. A VECTOR is defined as a double * and a MATRIX as a double **, so that the type of &vP is MATRIX, which is always the type used for a matrix in the OxVALUE.
- FOxCallBack calls the Ox function in the first argument. The next three arguments are the arguments to that Ox function: return type, function arguments, and number of arguments. FOxCallBack returns TRUE when successful, FALSE otherwise.
- After checking the returned value for type OX_DOUBLE, we can extract that double and return it in what pdFunc points to.

The following Ox code uses the pre-programmed Ox function for the numerical differentiation, and then the function just written in callback.c. The dRosenbrock function is the Ox code which is called from C. The difference between the two here is that the second expects and returns a row vector.

ox/dev/samples/callback/callback.ox

```
#include <oxstd.h>
#import <maximize>

extern "callback,FnNumDer" FnNumDer(const sFunc, vP);

fRosenbrock(const vP, const adFunc, const avScore,
    const amHessian)
{
    adFunc[0] = -100 * (vP[1] - vP[0] ^ 2) ^ 2
        - (1 - vP[0]) ^ 2;              // function value
return 1;                           // 1 indicates success
}
dRosenbrock(const vP)
{
    decl f = -100 * (vP[1] - vP[0] ^ 2) ^ 2
        - (1 - vP[0]) ^ 2;
    return f;                      // return function value
}
```

[4] The Ox 6 version of this example was storing the function argument in a static global variable s_pvOxFunc. This could then be directly used in myFunc, avoiding the final argument. The drawback is that this makes the function call non-reentrant: it is not safe to call it from multiple threads (i.e. in a parallel for loop).

```
main()
{
    decl vp = zeros(2, 1), vscore;

    //numerical differentiation using provided Ox function
    Num1Derivative(fRosenbrock, vp, &vscore);
    print(vscore);

    // now using provided C function from DLL
    vscore = FnNumDer(dRosenbrock, vp');// expects row vec
    print(vscore);
}
```

A mistake in the callback function is handled in the same way as other Ox errors. For example, when changing `vP[1]` to `vP[3]` in `dRosenbrock` at line 15:

```
Runtime error in dRosenbrock (16): '[3] in matrix[1][2]' index out of range
Call trace:
.../ox/dev/samples/callback/callback.ox (16): dRosenbrock
Runtime error in dRosenbrock (16): in callback function
```

The callback code presented here is reentrant: it can be safely called simultaneously from multiple threads. If that is not the case, the external call must be labelled as `serial` in the Ox `extern` statement:

```
extern serial "callback,FnNumDer" FnNumDer(const sFunc, vP);
```

D4.5 C macros and functions to access an OxVALUE; using arrays

An `OxVALUE` is a structure that holds all the information of a variable that is used in Ox code. The Ox run time manages the memory of each `OxVALUE`, and provides methods to create and manipulate these from your code. Internally, the `OxVALUE` is a rather complex `struct`. From C, there are two ways to manipulate an `OxVALUE`:

1. Macros to manipulate the `struct` explicitly, and
2. Functions that treat the `OxVALUE` more as an opaque memory object.

The C code presented sofar used quite a few macros, but it is perhaps preferable to use functions.[5]

Here is a comparison of the macro and function versions of some cases seen sofar:

Macro	Function
`r = OxInt(pv, 0);`	`OxValGetInt(pv, &r);`
`c = OxInt(pv, 1);`	`OxValGetInt(pv + 1, &c);`
`OxMat(rtn, 0)[i][j] = 3;`	`OxValGetMat(rtn)[i][j] = 3;`
`OxZero(rtn, 0);`	`OxValSetZero(rtn);`
`r = OxMatr(pv, 0);`	`r = OxValGetMatr(pv);`
`OxSetDbl(OxArray(pv,1),0,logdet);`	`OxValSetDouble(` `OxValGetArrayVal(pv,1), logdet);`
`*pdFunc = OxDbl(prtn, 0);`	`OxValGetDouble(prtn, pdFunc);`

Note that an allocated `OxVALUE` must be freed, unless it is passed back to Ox code. Also note that `OxValSetMat` frees the object first, which is why `OxValSetZero` is used to first initialize it to integer zero. This is a subtle difference with the macro versions that do not imply a call to `OxFreeByValue`.[6] More information is under `OxValSet`... in Chapter D6.

When using Java or C#, the macros cannot be used, and only the functions are available.

An Ox array is an array of `OxVALUE`s. `ox/dev/array` provides an example on how these can be accessed from C.

[5] Traditionally, macros are more efficient, although that need not be the case with modern compilers.

[6] An argument may hold it's own copy, or be a reference. `OxFreeByValue` can determine this, and will only free the memory if appropriate. When an `OxVALUE` is uninitialized, it holds 'random' bytes, which may erroneously indicate the need for freeing the object.

Chapter D5

Who is in charge?

D5.1 Introduction

An Ox program can have only one `main` function, which is where program execution starts. The same is true for C, C++, Java, etc. In the previous chapter, the foreign code was compiled into a DLL that was called from Ox. In that case Ox is in charge.

Alternatively, the foreign language can be the executable that is in charge. It remains possible to make calls to Ox, but specific functions are called:

1. Using low-level maths functions only
   In this case Ox is used as a mathematical and statistical library. This does not pose any new challenges, except for using the documentation in §D6.5. An example is provided in `ox/dev/oxlib`. We use this to illustrate how Ox can be used from Java and C#.
2. Calling Ox functions.
   Any Ox code can be launched and run.

D5.2 Using Ox as a mathematical and statistical library

D5.2.1 Using Ox as a library from C/C++

This simply amounts to calling any of the mathematical and statistical functions exported by the Ox DLL. An example is in `ox/dev/samples/oxlib/oxlib.c`.

D5.2.2 Using Ox as a library from Java

Two Java examples are provided. The first is a plain command-line application:

ox/dev/samples/oxlib/HelloOx.java

```
/* HelloWorld.java */

import com.sun.jna.Pointer;
import com.sun.jna.ptr.IntByReference;
import com.sun.jna.ptr.DoubleByReference;
```

```
import ox.*;

public class HelloOx {
    static public void main(String argv[]) {

        double d = Ox.c_abs(0.5, 0.5);
        System.out.println ("Hello from Ox = " + d);

        DoubleByReference zr = new DoubleByReference();
        DoubleByReference zi = new DoubleByReference();

        Ox.c_div(0.5, 0.5, 0.5, 0.5, zr, zi);
        System.out.println(
        "Hello from Ox = re=" + zr.getValue() + " im=" + zi.getValue());

        double[] vx = {1, 0, 2, 3};
        System.out.println(vx[0] + vx[1] + vx[2] + vx[3]);
        double dsum = Ox.DVecsum(vx, 4);
        System.out.println("DVecsum: " + dsum);

        DoubleByReference logdet = new DoubleByReference();
        IntByReference sign = new IntByReference();

        Pointer pmat = Ox.MatAllocBlock(2, 2);
        Ox.MatCopyVecr(pmat, vx, 2, 2);
        Ox.IInvDet(pmat, 2, logdet, sign);
        Ox.VecrCopyMat(vx, pmat, 2, 2);
        Ox.MatFreeBlock(pmat);
        System.out.println("m[0][]: " + vx[0] + " " + vx[1]);
        System.out.println("m[1][]: " + vx[2] + " " + vx[3]);
        System.out.println("logdet= " + logdet.getValue());
        System.out.println("sign= " + sign.getValue());
    }
}
```

- If not yet done so, the JDK must be installed to enable `javac`.
  Java Native Access (JNA) is used for simplified calling to native code in the Ox DLL. JNA must be downloaded (`jna.jar` and `platform.jar`).
  The `dev/Ox.java` file defines the Ox class that imports the Ox functionality.
- We've created a subfolder for the Ox interface: `oxlib/java/ox`, where a copy of `dev/Ox.java` is put. `Ox.java` defines the exported functions and constant values. This is then compiled in a Console (or Terminal) window from the `oxlib/java` folder:

  ```
  javac -classpath /usr/share/java/jna.jar ox/Ox.java
  ```

  The correct path to `jna.jar` has to be specified (under Linux I have it in /usr/share/java/). This creates `oxlib/java/ox/Ox.class`.
- To compile HelloOx:

  ```
  javac -classpath /usr/share/java/jna.jar:. HelloOx.java
  ```

  Use a semicolon instead of a colon for the path separator under Windows.
  And to run HelloOx:[1]

[1] To run, the Ox DLL must be in the search path. Under Linux, the Ox environment must also

```
java -classpath /usr/share/java/jna.jar:. HelloOx
```

- `DoubleByReference` is used to pass a pointer to a double to the `c_div` call.
- A `double[]` is used when the Ox function expects a `VECTOR`.
- The matrix is created inside the Ox run time (and freed there), and stored in a `Pointer` object. The matrix cannot be accessed directly. Instead, it is vectorized into a `double[]` that can be referenced.

The following table lists the types that may be encountered in the Ox foreign language interface:

C/Ox type	Java equivalent
`int`	`int`
`int *`	`IntByReference`
`BOOL`	`int`
`double`	`double`
`double *`	`DoubleByReference`
`char *`	`String`
`VECTOR`	`double[]`
`MATRIX`	`Pointer` (see example)
`OxVALUE *`	`Pointer`

be found:
OX10PATH="/usr/share/OxMetrics10/ox/include:/usr/share/OxMetrics10/apps:/usr/share/OxMetrics10/ox";
export OX10PATH

D5.2.3 Using Ox as a library from C#

ox/dev/samples/oxlib/winform_cs/Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace winform_cs
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double d1 = 0.5;
            // Need to call RanSetRan at least once to initialize rng environment
            Ox.RanSetRan("Default");
            d1 = Ox.DRanU();
            textBox1.Text = Convert.ToString(d1);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            double d1 = Convert.ToDouble(textBox2.Text);
            try
            {
                d1 = Ox.DLogGamma(d1);
                textBox2.Text = Convert.ToString(d1);
            }
            catch (BadImageFormatException exception)
            {
                textBox2.Text = "failed";
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            double[] mat = new double[4];
            IntPtr pmat = default(IntPtr);
            int result = 0;
            Int32 sign = 0;
            double logdet = 0.0;

            mat[0] = Convert.ToDouble(textBox3.Text);
            mat[1] = Convert.ToDouble(textBox4.Text);
            mat[2] = Convert.ToDouble(textBox5.Text);
```

```
            mat[3] = Convert.ToDouble(textBox6.Text);

            pmat = Ox.MatAllocBlock(2, 2);
            Ox.MatCopyVecr(pmat, mat, 2, 2);
            Ox.SetNaN(ref logdet);
            result = Ox.IInvDet(pmat, 2, ref logdet, ref sign);
            Ox.VecrCopyMat(mat, pmat, 2, 2);
            Ox.MatFreeBlock(pmat);

            textBox3.Text = Convert.ToString(mat[0]);
            textBox4.Text = Convert.ToString(mat[1]);
            textBox5.Text = Convert.ToString(mat[2]);
            textBox6.Text = Convert.ToString(mat[3]);

            textBox7.Text = "logdet=" + Convert.ToString(logdet);
            textBox8.Text = "sign det=" + Convert.ToString(sign);
        }

        private void button4_Click(object sender, EventArgs e)
        {
            double[] Vec = new double[6];
            double[] VecAcf = new double[6];
            int result = 0;

            Vec[0] = 1;
            Vec[1] = 2;
            Vec[2] = 3;
            Vec[3] = 0;
            Vec[4] = 1;
            Vec[5] = 4;

            result = Ox.IGetAcf(Vec, 6, 2, VecAcf, 0);

            textBox9.Text = Convert.ToString(VecAcf[0]);
            textBox10.Text = Convert.ToString(VecAcf[1]);
        }
    }
}
```

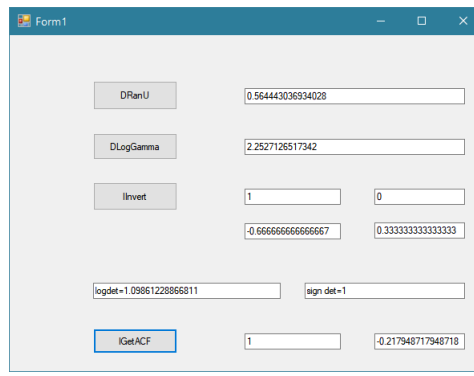
- C# has Platform Invocation Services (PInvoke) to allow managed code to call unmanaged functions that are implemented in the Ox DLL. The `dev/Ox.cs` file defines the Ox class that imports the Ox functionality.
- Load the `winform_cs.sln` solution file in Visual Studio (or convert to a newer version), to learn more about the program.
- The program needs to find `oxwin.dll`. Either this should be in the default search path, or, as done in this program, can be set explicitly at program start up:

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto,
    SetLastError = true)]
static extern bool SetDllDirectory(string lpPathName);

[STAThread]
static void Main()
```

```
{
    SetDllDirectory("C:\\Program Files\\OxMetrics10\\ox");
    //...
```

- The Ox functionality is accessed from a dialog:



Pressing a button will result in a call to the Ox DLL.

- `ref logdet` is used to pass a pointer to a double (`logdet` in this case) to the `Ox.IInvDet` call.
- A `double[]` is used when the Ox function expects a `VECTOR`.
- The matrix is created inside the Ox run time (and freed there), and stored in a `IntPtr` object. The matrix cannot be accessed directly. Instead, it is vectorized into a `double[]` that can be referenced in C#.

The following table lists the types that may be encountered in the Ox foreign language interface:

C/Ox type	C# equivalent	
`int`	`int`	
`int *`	`int`	call with `ref`
`BOOL`	`int`	
`double`	`double`	
`double *`	`double`	call with `ref`
`char *`	`string`	
`VECTOR`	`double[]`	
`MATRIX`	`IntPtr` (see example)	
`OxVALUE *`	`IntPtr`	

D5.3 Creating and using Ox objects

It is more common that the foreign language in charge wishes to call Ox code, rather than the underlying matrix and statistical library. That means that the foreign language needs to

1. Start the Ox run-time engine, load and compile Ox code
   Use `OxMain` or `OxMainCmd`. The former expects an array of strings, in the same was as the C main function. The latter takes a string as an argument, ans is more

convenient from Java or C#. The `-r` switch is used to compile but not (yet) run the code.

2. Make function calls, or create an object of a class and call its methods
   An object is created with `FOxCreateObject`. Function members of that object are called with `FOxCallBackMember`.
   Alternatively, a function can be called with `FOxCallBack`, after creating an OxVALUE with the name of the function as a string.
3. Shut down the Ox run-time engine when done.
   `OxRunExit` followed by `OxMainExit`.

Three examples of this process are provided: C, Java and C#. We focus on the Java version below. All versions use the following Ox code:

ox/dev/samples/object/class_test.ox

```
#include <oxstd.h>

class Test
{
    Test();
    ReturnDbl();
    ReturnMat();
    Func1(const a);
    Print();
    ~Test();

    decl m_mX;
};

Test::Test()
{
    println("Test object constructed");
    m_mX = 0;
}
Test::~Test()
{
    println("Test object destructed");
}
Test::ReturnDbl()
{
    return 5.3;
}
Test::ReturnMat()
{
    return <1.5, 3.5, 4.5; 7, 8, 9>;
}
Test::Func1(const a)
{
    println("Argument 1=", a);
    return "done";
}
Test::Print()
{
    println("m_mX=", m_mX);
}
```

D5.3.1 Creating and using Ox objects from Java

ox/dev/samples/object/javaCallObject.java

```
import com.sun.jna.Pointer;
import com.sun.jna.ptr.IntByReference;
import com.sun.jna.ptr.DoubleByReference;

import ox.*;

public class CallObject {

    static public void main(String argv[]) {

        if (Ox.OxMainCmd("-r- ../class_test") <= 1)
        {
            System.out.println("Java: Failed to start Ox program");
            return;
        }
        else
            System.out.println("Java: Ox program successfully started");

        Pointer oxval = Ox.OxStoreCreate(1);
        Pointer rtnval = Ox.OxStoreCreate(1);
        Pointer clval = Ox.OxStoreCreate(1);

        /* create an Ox object */
        if (Ox.FOxCreateObject("Test", clval, Pointer.NULL, 0) != 1)
        {
            System.out.println("Java:Failed to create object\n");
            return;
        }
        System.out.println("Java: Created an object of class Test\n");

        if (Ox.FOxCallBackMember(clval, "Print", rtnval, Pointer.NULL, 0) != 1)
        {
            System.out.println("Java: Failed to call Test::Print");
            return;
        }

        int i, j, k, r, c;

        Ox.FOxCallBackMember(clval, "ReturnMat", rtnval, Pointer.NULL, 0);
        if (Ox.OxValHasType(rtnval, Ox.OxTypes.OX_MATRIX.getValue()) != 0)
        {
            r = Ox.OxValGetMatr(rtnval);
            c = Ox.OxValGetMatc(rtnval);
            System.out.println(
                "Java: Return value is a " + r + " x " + c + " matrix:");

            double[] vx = new double[r * c];
            Ox.OxValGetVecr(rtnval, vx);
            for (i = k = 0; i < r; ++i)
            {
                System.out.print("Java: ");
                for (j = 0; j < c; ++j, ++k)
                    System.out.print(" " + vx[k]);
```

```
                System.out.println("");
            }
        }
        else
            System.out.println("Java: Return value is not a matrix");

        /* change the value of the m_mX member variable to a matrix */
        Ox.OxValSetMatZero(oxval, 3, 3);
        Ox.FOxSetDataMember(clval, "m_mX", oxval);
        Ox.FOxCallBackMember(clval, "Print", rtnval, Pointer.NULL, 0);

        Ox.OxStoreDelete(oxval, 1);
        Ox.OxStoreDelete(rtnval, 1);
        Ox.OxStoreDelete(clval, 1);

        Ox.OxRunExit();
        Ox.OxMainExit();
    }
}
```

- `Ox.OxMainCmd` specifies that Ox file that should be loaded and compiled (this could also be a `.oxo` file[2]).
- `OxVALUE` objects are created and managed in the Ox run-time by using `OxStoreCreate`. These are initialized to integers with value zero.
- Next, an object of class `Test` is created and stored in `clval`. `FOxCreateObject` also calls the constructor, which takes no arguments. This is followed by a call to `Print`, which again takes no argument.
- Then we call `Test::ReturnMat` from the object, which takes no arguments, but returns a matrix in `rtnval`. The returned matrix is converted to a vector `vx`, which can be handled inside the Java code.
- Next, a matrix of zeros is created in the `oxval` variable using `OxValSetMatZero`. The `m_mX` data member of the object is set to this matrix using `FOxSetDataMember`.
- Finally, the created `OxVALUE`s are removed with `OxStoreDelete` (executing the call to the destructor as well), and the Ox run time is closed down.

[2] Ox Professional has a command -line switch to link in all dependencies: `-cl`

This is the output under Windows:

```
Java: Ox program successfully started
Test object constructed
Java: Created an object of class Test

m_mX=0
Java: Return value is a 2 x 3 matrix:
Java:  1.5 3.5 4.5
Java:  7.0 8.0 9.0
m_mX=
      0.00000      0.00000      0.00000
      0.00000      0.00000      0.00000
      0.00000      0.00000      0.00000
Test object destructed
```

D5.4 Adding a user-friendly interface

Ox is limited in terms of user interaction, only providing console style input using the `scan` function. It is possible, however, to use much more powerful external tools to add dialogs and menus. In that way, a much better interface could be written than ever possible directly in Ox. Indeed, there are no plans to make generic interface components an intrinsic part of Ox: this would always lag behind the latest developments.

Various approaches could be considered to add a user interface:

(1) Write a separate interactive program which creates an input file.
This input file is read by an Ox program that is run separately.

(2) Write a separate interactive program which generates an Ox source file.
This is similar to (1), but now a whole Ox program is written, rather than the input configuration.
This approach is taken by PcNaive: it collects user input on Monte Carlo design, generates an Ox program from this, and calls `OxRun` to run the generated code. It can also retrieve settings from previously generated source code, to produce a sophisticated interactive package.

(3) Use OxApp to add an interactive front-end, using OxMetrics for output. The program can then be started from OxMetrics after adding it to the Ox Apps.
This is what PcNaive uses, and discussed in Chapter D2.

(4) Write a DLL that exports dialogs to be used in Ox source code.
This approach is used by TSM (see `www.timeseriesmodelling.com`, created by James Davidson), using the OxJapi package (a wrapper around the AWT GUI toolkit of Java, available from `personal.vu.nl/c.s.bos/software.html`).

(5) Call Ox source code from an interactive Java, C# or C++ program.
In this case an Ox object can be created in the foreign language, and function members called, see §D5.3.

Chapter D6

Ox Exported Functions

D6.1 Introduction

This chapter documents the *Ox* related functions which are exported from the Ox DLL. The low level mathematical and statistical functions are listed in §D6.5. The OxMetrics specific functions are documented in §D6.4.

§D6.1 lists the *Ox* related functions. All these functions section require `oxexport.h`.

Functions which interface with Ox use the `OXCALL` specifier. This, in turn, is just a relabelling of `JDCALL`, defined in `ox/dev/jdsystem.h`. Currently, this declares the calling convention under Windows (depending on the compiler). On other platforms, it defaults to nothing. So, to add support for a new compiler, you could:

1. leave `jdsystem.h` unchanged, and set the right compiler options when compiling (this is the preferred approach);
2. add support for the new compiler in `jdsystem.h`.

Ox extension function syntax

```
void OXCALL FnFunction(OxVALUE *rtn, OxVALUE *pv, int cArg);
```

`rtn`	in:	pointer to an `OxVALUE` of type `OX_INT` and value 0
	out:	receives the return value of pvFunc
`pv`	in:	the arguments of the function call; *they must be checked for type before being accessed*.
	out:	unchanged, apart from possible conversion from `OX_INT` to `OX_DOUBLE` or vice versa
`cArg`	in:	number of elements in `pv`; unless the function has a variable number of arguments, there is no need to reference this value.

No return value.

Description

This is the syntax required to make a function callable from Ox. `FnFunction` should be replaced by an appropriate name, but is not the name under which the function is known inside an Ox program.

The standard types for Ox variables that are visible from extension code are:

Ox types	description
OX_INT	integer
OX_DOUBLE	double
OX_MATRIX	matrix
OX_STRING	string
OX_ARRAY	array
OX_FUNCTION	function
OX_CLASS	class object
OX_INTFUNC	internal function
OX_FILE	open file
OX_LAMBDA	lambda function
OX_BLOB	blob: opaque memory block
OX_EMPTY	empty: .Null

D6.2 Ox function summary

FOxCallBack, FOxCallBackMember

```
BOOL FOxCallBack(OxVALUE *pvFunc, OxVALUE *rtn, OxVALUE *pv,
    int cArg);
BOOL FOxCallBackMember(OxVALUE *pvClass, const char *sMember,
    OxVALUE *rtn, OxVALUE *pv, int cArg);
```

pvFunc	in:	the function to call, must be of type OX_FUNCTION, OX_INTFUNC, OX_LAMBDA, or OX_STRING
pvClass	in:	the object from which to call a member, must be of type OX_CLASS
sMember	in:	name of the member function
rtn	out:	receives the return value of the function call
pv	in:	the arguments of pvFunc
cArg	in:	number of elements in pv

Return value

TRUE if the function is called successfully, FALSE otherwise.

Description

Calls an Ox function from C.

If the returned value rtn is not passed back to Ox, call OxFreeByValue(rtn) to free it.

FOxCreateObject

```
BOOL FOxCreateObject(const char *sClass, OxVALUE *rtn,
    OxVALUE *pv, int cArg);
```

sClass	in:	name of class
rtn	in:	pointer to Ox_VALUE
	out:	receives the created object
pv	in:	the arguments for the constructor
cArg	in:	number of elements in pv
pvClass	in:	the object from which to delete, previously created with FOxCreateObject

Return value

Returns TRUE if the function is called successfully, FALSE otherwise.

Description

FOxCreateObject creates an object of the named class which can be used from C; the constructor will be called by this function. Use OxDeleteObject to delete the object.

FOxGetDataMember

```
BOOL FOxGetDataMember(OxVALUE *pvClass, const char *sMember,
    OxVALUE *rtn);
```

pvClass	in:	the object from which to get a data member, must be of type OX_CLASS
sMember	in:	name of the data member
rtn	out:	receives the return value of the function call

Return value

TRUE if the function is called successfully, FALSE otherwise.

Description

Gets a data member from an object. The returned value is for reference only, and should not be changed, and should only be used for temporary reference.

FOxLibAddFunction

```
BOOL FOxLibAddFunction(char *sFunc, OxFUNCP pFunc, BOOL fVarArg);
```

sFunc	in:	string describing function
pFunc	in:	pointer to C function to install
fVarArg	in:	TRUE: has variable argument list

Return value

TRUE if function installed successfully, FALSE otherwise.

Description

OxFUNCP is a pointer to a function declared as:

```
void OXCALL Func(OxVALUE *rtn, OxVALUE *pv, int cArg);
```

The syntax of sFunc is:

arg_types$*function_name*\0

arg_types is a c (indicating a const argument) or a space, with one entry for each declared argument.

This function links in C library functions statically, e.g. for part of the drawing library:

```
FOxLibAddFunction("cccc$Draw",          fnDraw,       0);
FOxLibAddFunction("ccccc$DrawT",        fnDrawT,      0);
FOxLibAddFunction("ccc$DrawX",          fnDrawX,      0);
```

```
FOxLibAddFunction("ccccc$DrawMatrix",   fnDrawMatrix,  1);
FOxLibAddFunction("cccccc$DrawTMatrix", fnDrawTMatrix, 1);
FOxLibAddFunction("ccccc$DrawXMatrix",  fnDrawXMatrix, 1);
```

This function is not required when using the `extern` specifier for external linking, as used in most examples in this chapter.

FOxLibAddFunctionEx

```
BOOL  FOxLibAddFunctionEx(char *sFunc, OxFUNCP pFunc, int cArgs,
    int flFlags);
```

`sFunc`	in:	name of the function
`pFunc`	in:	pointer to C function to install
`cArg`	in:	number of required arguments
`flFlags`	in:	0, or a combination of: `OX_VARARGS` (variable no of arguments) `OX_SERIAL` (cannot be called in parallel)

Return value

`TRUE` if function installed successfully, `FALSE` otherwise.

Description

`OxFUNCP` is a pointer to a function declared as:

```
void OXCALL Func(OxVALUE *rtn, OxVALUE *pv, int cArg);
```

This function links in C library functions statically, e.g.:

```
FOxLibAddFunctionEx("fprint", fnFprint, 2, OX_VARARGS | OX_SERIAL);
```

This function is not required when using the `extern` specifier for external linking, as used in most examples in this chapter.

FOxRun

```
BOOL FOxRun(int iMainIP, char *sFunc);
```

`iMainIP`	in:	return value from `OxMain`
`sFunc`	in:	name in Ox code of function to call

Return value

`TRUE` if the function is run successfully, `FALSE` otherwise.

Description

Calls a function by name, bypassing `main()`.

FOxSetDataMember

```
BOOL FOxSetDataMember(OxVALUE *pvClass, const char *sMember,
    OxVALUE *pv);
```

`pvClass`	in:	the object in which to set a data member, must be of type `OX_CLASS`
`sMember`	in:	name of the data member
`pv`	in:	new value of the data member

Return value

`TRUE` if the function is called successfully, `FALSE` otherwise.

Description

Sets a data member from an object. The assigned value is taken over (if it is by value, it is transferred, and `pv` will have lost its by value property (`OX_VALUE`).

IOxRunInit

```
int  IOxRunInit(void);
```

Return value

Zero for success, or the number of link errors.

Description

Links the compiled code and initializes to prepare for running the code.

IOxVersion IOxVersionIsProfessional IOxVersionOxo

```
int  IOxVersion(void);
int  IOxVersionOxo(void);
int  IOxVersionIsProfessional(void);
```

Return value

`IOxVersion` returns 100 times the Ox version number, so `100` for version 1.00.
`IOxVersionOxo` returns 100 times the version number of compiled Ox code (.oxo files), so `100` for version 1.00. Note that this changes less often than the version of Ox.
`IOxVersionIsProfessional` ruturns 1 for Ox Professional, 0 for Ox Console.

OxCloneObject

```
void OxCloneObject(OxVALUE *rtn, OxVALUE *pvObject, BOOL bDeep);
```

`rtn`	out:	cloned object
`pvObject`	out:	object to duplicate
`bDeep`	out:	TRUE (deep copy) or FALSE (shallow copy)

No return value.

Description

`OxCloneObject` clones the object by taking a duplicate. Use `OxDeleteObject` to delete a cloned object.

OxDeleteObject

```
void OxDeleteObject(OxVALUE *pvClass);
```

`sClass`	in:	name of class

No return value.

Description

`OxDeleteObject` deletes the object; this calls the destructor, and deallocates all memory owned by the object. Use `FOxCreateObject` to create an object.

OxFnDouble, OxFnDouble2, OxFnDouble3, OxFnDouble4, OxFnDoubleInt

```
void OxFnDouble(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn1)(double) );
void OxFnDouble2(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn2)(double,double) );
void OxFnDouble3(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn3)(double,double,double) );
void OxFnDouble4(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn4)(double,double,double,double) );
void OXCALL OxFnDoubleInt(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fndi)(double,int) )
void OXCALL OxFnDoubleVec(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn6)(double), int iVecAt);
void OXCALL OxFnDouble2Vec(OxVALUE *rtn, OxVALUE *pv,
```

```
    double (OXCALL * fn7)(double,double), int iVecAt);
void OXCALL OxFnDouble3Vec(OxVALUE *rtn, OxVALUE *pv,
    double (OXCALL * fn8)(double,double,double), int iVecAt);
```

rtn	out:	return value of function
pv	in:	arguments for function fn
fn1	in:	function of one double, returning a double
fn2	in:	function of two doubles, returning a double
fn3	in:	function of three doubles, returning a double
fn4	in:	function of four doubles, returning a double
fndi	in:	function of a double and an int, returning a double

No return value.

Description

These functions are to simplify calling C functions, as for example in:

```
static void OXCALL
    fnProbgamma(OxVALUE *rtn, OxVALUE *pv, int cArg)
    {   OxFnDouble3(rtn, pv, DProbGamma);
    }
static void OXCALL
    fnProbchi(OxVALUE *rtn, OxVALUE *pv, int cArg)
    {   OxFnDouble2(rtn, pv, DProbChi);
    }
static void OXCALL
    fnProbnormal(OxVALUE *rtn, OxVALUE *pv, int cArg)
    {   OxFnDouble(rtn, pv, DProbNormal);
    }
```

OxFreeByValue

```
void OxFreeByValue(OxVALUE *pv);
```

pv	in:	pointer to value to free
	out:	freed value

No return value.

Description

Frees the matrix/string/array (i.e. pv is OX_MATRIX, OX_ARRAY, or OX_STRING) if it has property OX_VALUE.

OxGetMainArgs, OxGetOxArgs

```
void OxGetMainArgs(int *pcArgc, char ***pasArgv);
void OxGetOxArgs(int *pcArgc, char ***pasArgv);
```

pcArgc	in:	pointer to integer
	out:	destination holds number of arguments returned
pasArgv	in:	pointer to array (pointer) of strings (char pointer)
	out:	destination points to the array of arguments

No return value.

Description

Queries Ox for the current executable arguments (OxGetMainArgs), and the arguments specified to the running Ox program (available to the Ox code), OxGetOxArgs.

Note that just a pointer to the array of characters is passed, and the contents may not be modified (see the `OxSet` variants for changing the arguments).
By convention, the first argument for the executable is the name of the executable, while for the Ox program it is the name of the Ox file .

OxGetPrintlevel

```
int  OxGetPrintlevel(void);
```

Return value

returns the current print level (see `OxSetPrintlevel`).

OxGetUserExitCode

```
int  OxGetUserExitCode(void);
```

Return value

Returns the current exit code, as set by calling the Ox function `exit()` (the default is 0).

OxLibArgError

```
void OxLibArgError(int iArg);
```

`iArg`	in:	argument index

No return value.

Description

Reports an error in argument `iArg`, and generates a run-time error.

OxLibArgTypeError

```
void OxLibArgTypeError(int iArg, int iExpect, int iFound);
```

`iArg`	in:	argument index
`iExpect`	in:	expected type, one of `OX_INT`, `OX_DOUBLE`, `OX_MATRIX`, etc.
`iFound`	in:	found type

No return value.

Description

Reports a type error in argument `iArg`, and generates a run-time error.

OxLibCheckArrayMatrix

```
void OxLibCheckArrayMatrix(OxVALUE *pv, int iFirst, int iLast,
    MATRIX m);
```

`pv`	in:	array of values of type `OX_ARRAY`
`iFirst`	in:	first in array to check
`iLast`	in:	last in array to check
`m`	in:	matrix

No return value.

Description

Checks if any of the values in `pv[iFirst]...pv[iLast]` (these must be of type `OX_ARRAY`) coincide with the matrix m.

OxLibCheckMatrixSize

```
void OxLibCheckMatrixSize(OxVALUE *pv, int iFirst, int iLast,
    int r, int c);
```

pv	in:	array of values of any type
iFirst	in:	first in array to check
iLast	in:	last in array to check
r	in:	required row dimension
c	in:	required column dimension

No return value.

Description

Checks whether all the values in `pv[iFirst]...pv[iLast]` are of type `OX_MATRIX`, and whether they have the required dimension and are non-empty.

OxLibCheckSquareMatrix

```
void OxLibCheckSquareMatrix(OxVALUE *pv, int iFirst, int iLast);
```

pv	in:	array of values of any type
iFirst	in:	first in array to check
iLast	in:	last in array to check

No return value.

Description

Checks whether all the values in `pv[iFirst]...pv[iLast]` are of type `OX_MATRIX`, and whether the matrices are square and non-empty.

OxLibCheckType

```
void OxLibCheckType(int iType, OxVALUE *pv, int iFirst,int iLast);
```

iType	in:	required type, one of `OX_INT`, `OX_DOUBLE`, `OX_MATRIX`, etc.
pv	in:	array of values of any type
	out:	`OX_INT` changed to `OX_DOUBLE` or vice versa
iFirst	in:	first in array to check
iLast	in:	last in array to check

No return value.

Description

Checks whether all the values in `pv[iFirst]...pv[iLast]` are of type `iType`. If the argument(s) cannot be converted (typecast) to the requested type, a run-time error is generated. More specifically:

1. if the argument does not have a value, it is an error;
2. if the argument is of the requested type, is is a success;
3. else it is an error if the argument is an empty matrix or cannot be converted (typecast) to the specified type. The following conversions will succeed
   int from double, matrix[1][1], string (first character);
   double from int, matrix[1][1], string (up to 8 bytes mapped);
   matrix from int, double, string (first character);
   string from int, double, function, class.

OxMain,OxMain_T,OxMainCmd

```
int  OxMain(int argc, char *argv[]);
int  OxMainCmd(char *sCommand);
```

`argc`	in:	number of command line arguments
`argv`	in:	command line argument list (first is program name)
`sCommand`	in:	command line as one string

Return value

The entry point for `main()` if successful, or a value ≤ 1 if there was a compilation or link error.

Description

Processes the Ox command line, including compilation, linking and running. The arguments to `OxMain` are provided as an array of pointers to strings, with the first entry being ignored.

The arguments to `OxMainCmd` are provided as one command line string, with arguments separated by a space. A part in double quotes is considered one argument, so `"-r- ranapp.ox"` and `"-r- "ranapp.ox""` are the same (the latter is written as `"-r- \"ranapp.ox\""` in C).

OxMainExit

```
void OxMainExit(void);
```

No return value.

Description

Deallocates run-time buffers.

OxMainInit

```
void OxMainInit(void);
```

No return value.

Description

Sets output destination to `stdout`, and links the standard run-time and drawing library.

OxMakeByValue

```
void OxMakeByValue(OxVALUE *pv);
```

`pv`	in:	pointer to value to make by value
	out:	copied value (if not already by value)

No return value.

Description

Makes the matrix/string/array (i.e. `pv` is `OX_MATRIX`, `OX_ARRAY`, or `OX_STRING`) by value. That is, if it doesn't already have the `OX_VALUE` property, the contents are copied, and the `OX_VALUE` flag is set. Note that a newly allocated value automatically has the `OX_VALUE` flag.

OxMessage

```
void OxMessage(char *s);
```

`s`	in:	text to print

No return value.

Description

Prints a message.

OxPuts

```
void OxPuts(char *s);
```

s	in:	text to print

No return value.

Description

Prints text (equivalent to using `print(s)` inside the Ox code).

OxRunAbort

```
void OxRunAbort(int i);
```

i	in:	currently not used

No return value.

Description

Exits the run-time interpreter at the next end-of-line. The code should have end-of-line coding on (so not using `-on`). This exits cleanly, so that, when an external program is running Ox functions (e.g. using `FOxRun`), the next call will work as expected.

OxRunError

```
void OxRunError(int iErno, char *sToken);
```

iErno	in:	error number as defined in `oxexport.h`, or: −1: skips text of error type, but still reports *sToken*
sToken	in:	NULL or offending token

No return value.

Description

Throws a run-time error exception.

OxRunErrorMessage

```
void OxRunErrorMessage(char *s);
```

s	in:	message text

No return value.

Description

Reports the specified run-time error message using `OxRunMessage`, the call trace, and then *exits the program*.

OxRunExit

```
void OxRunExit(void);
```

No return value.

Description

Cleans up after running a program.

OxRunMainExitCall

```
void OxRunMainExitCall(void (OXCALL * fn)(void));
```

fn	in:	function to be called when Ox `main` finishes

No return value.

Description

Schedules a function to be called at the end of `main`. This can be used if a library needs a termination call (or, e.g. for a final barrier synchronization in parallel code). Currently, only 10 such functions can be added.

OxRunMessage

```
void OxRunMessage(char *s);
```

`s` in: message text

No return value.

Description

Reports a run-time message.

OxRunErrorMessage

```
void OxRunWarningMessage(char *sFunc, char *sMsg);
```

`sFunc` in: name of function reporting warning

`sMsg` in: text of user-defined warning message

No return value.

Description

Reports user-determined warning message (of type `WFL_USER`, which can be switched off in Ox code with `oxwarning`).

OxSetMainArgs, OxSetOxArgs

```
void  OxSetMainArgs(int cArgc, char *asArgv[]);
void  OxSetOxArgs(int cArgc, char **asArgv);
```

`cArgc` in: number of arguments

`asArgv` in: array (pointer) of argument strings (char pointer)

No return value.

Description

Specifies the current executable arguments (`OxSetMainArgs`), and the arguments specified to the running Ox program (available to the Ox code), `OxSetOxArgs`.

By convention, the first argument for the executable is the name of the executable, while for the Ox program it is the name of the Ox file.

The OxMPI package provides an example of the use of `OxGetMainArgs` and `OxSetMainArgs`, because MPI needs to rewrite arguments to communicate information.

OxSetPrintlevel

```
void OxSetPrintlevel(int iSet);
```

`iSet` in: print level

No return value.

Description

-1	no output,
0	output from print and println is suppressed, but messages and warnings are printed,
1	normal output.

OxSetUserExitCode

```
void OxSetUserExitCode(int iSet);
```

iSet	in:	exit code

OxThreadCount, OxThreadNo

```
int     OXCALL  OxThreadCount(void);
int     OXCALL  OxThreadNo(void);
```

Return value

OxThreadCount returns the number of threads, OxThreadNo returns the current thread number (0 is the main thread).

OxStoreCreate, OxStoreDelete

```
OxVALUE *OXCALL OxStoreCreate(int c);
void    OXCALL  OxStoreDelete(OxVALUE *pv, int c);
```

pv	in:	pointer to OxVALUE
c	in:	number of OxVALUE in pv

Return value

OxStoreCreate returns a pointer to the first element in the created array of OxVALUEs. They are initialized to an integer with value 0.

Description

These functions can be useful to work with OxVALUEs, but leaving ownership of the memory within the Ox DLL (e.g. using languages other than C/C++). Every call to OxStoreCreate must be matched by OxStoreDelete.

If an element is made an object by using FOxCreateObject, it will be automatically be deleted (and the destructor called) by OxStoreDelete.

To get access to an element beyond the first use OxValGetVal.

OxValColumns

```
int  OxValColumns(OxVALUE *pv);
```

pv	in:	OxVALUE to get size of

No return value.

Description

OxValColumns	as Ox function columns

OxValDuplicate

```
OxVALUE OxValDuplicate(OxVALUE *pv)
void    OXCALL  OxValDuplicate2(OxVALUE *pvDest, OxVALUE *pvSrc);
```

pv	in:	Ox variable to duplicate
pvDest	out:	set to to duplicate (it is not freed first)
pvSrc	in:	Ox variable to duplicate

Return value

OxValDuplicate returns an Ox variable that is the duplicate of the argument. For objects, just the referencce is returned (use OxCloneObject to take a copy.

OxValGet...

```
OxVALUE *OxValGetArray(OxVALUE *pv);
int      OxValGetArrayLen(OxVALUE *pv);
OxVALUE *OxValGetArrayVal(OxVALUE *pv, int i);
void    *OxValGetBlob(OxVALUE *pv, int *pI1, *int pI2);
const char * OxValGetClassName(OxVALUE *pv);
BOOL     OxValGetDouble(OxVALUE *pv, double *pdVal);
BOOL     OxValGetInt(OxVALUE *pv, int *piVal);
MATRIX   OxValGetMat(OxVALUE *pv);
int      OxValGetMatc(OxVALUE *pv);
int      OxValGetMatr(OxVALUE *pv);
int      OxValGetMatrc(OxVALUE *pv);
OxVALUE *OxValGetStaticObject(OxVALUE *pv);
char    *OxValGetString(OxVALUE *pv);
BOOL     OxValGetStringCopy(OxVALUE *pv, char *s, int mxLen);
int      OxValGetStringLen(OxVALUE *pv);
OxVALUE *OxValGetVal(OxVALUE *pv, int i);
BOOL     OxValGetVecc(OxVALUE *pv, VECTOR vX);
BOOL     OxValGetVecr(OxVALUE *pv, VECTOR vX);
```

`pv`	in:	`OxVALUE` to get information from
	out:	could have changed to reflect requested type
`i`	in:	index in array
`pdVal`	out:	double value (if successful)

Return value

OxValGetArray	array of OxVALUEs or NULL if not OX_ARRAY
OxValGetArrayLen	array length or 0 if not OX_ARRAY
OxValGetArrayVal	ith OxVALUE or NULL if not OX_ARRAY or index is beyond array bounds
OxValGetBlob	returns the contents of the OX_BLOB
OxValGetClassName	returns the name of the class for this object
OxValGetDouble	TRUE if value in pv can be interpreted as a double
OxValGetInt	TRUE if value in pv can be interpreted as an integer
OxValGetMat	MATRIX if value in pv can be interpreted as a matrix or NULL if failed
OxValGetMatc	number of columns if successful or 0 if failed
OxValGetMatr	number of rows if successful or 0 if failed
OxValGetMatrc	number of elements if successful or 0 if failed
OxValGetStaticObject	returns the global OxVALUE for this object, which holds the static members
OxValGetString	pointer to string (may be NULL for empty string) or NULL if not OX_STRING
OxValGetStringCopy	copies the string (up to mxLen - 1 characters), sets ending \0; returns TRUE if type is OX_STRING, FALSE otherwise
OxValGetStringLen	string length (≥ 0) or 0 if not OX_STRING
OxValGetVal	returns the ith OxVALUE in pv (without checking the pv array bounds)

Description

Gets information from an OxVALUE. A type conversion is applied to pv if the OxVALUE is not of the requested type (which is unlike the macro versions of §D6.3). The conversion is similar to making a call to OxLibCheckType first, and then using the macro version. If conversion to the requested type cannot be made, this is reflected in the return value.

OxValHasType, OxValHasFlag

```
BOOL OxValHasType(OxVALUE *pv, int iType);
BOOL OxValHasFlag(OxVALUE *pv, int iFlag);
```

pv	in:	OxVALUE to get information from
iType	in:	type to test for
iFlag	in:	flag (property) to test for

Return value

TRUE if pv has the specified type/property.

OxValInit...

```
void OxValInit(OxVALUE *pv);
void OxValInitNull(OxVALUE *pv);
```

pv	in:	OxVALUE to set
	out:	changed value

No return value.

Description

`OxValInit` sets `pv` to an integer with value 0

`OxValInitNull` sets `pv` to `.Null`

These functions *do not* call `OxFreeByValue` before changing the value (unlike the `OxValSet` functions, but like the macro versions). If the argument is not received from Ox (e.g. a locally declared `OxVALUE` variable), you should initialize it with one of these functions to avoid a spurious call to free memory.

These functions were called `OxValSetNull` and `OxValSetZero` in Ox 8.

`OxValInitNull` sets `pv` to an integer of value zero with type `OX_EMPTY`. Using such a value in an expression in Ox leads to a run-time error (variable has no value).

A compound type (matrix, string, array) can be freed using `OxFreeByValue`, but normally that would be left to the Ox run-time system. If `pv` is not received from Ox, you should initialize before calling this function, for example:

```
OxVALUE tmp;
OxValInit(&tmp);
OxValSetMatZero(&tmp, 2, 2);
```

Failure to do so could bring down the Ox system.

OxValRows

```
int  OxValRows(OxVALUE *pv);
```

`pv` in: `OxVALUE` to get size of

No return value.

Description

`OxValRows` as Ox function `rows`

OxValSet...

```
void OxValSetArray(OxVALUE *pv, int c);
void OxValSetBlob(OxVALUE *pv, int i1, int i2, void *p);
void OxValSetDouble(OxVALUE *pv, double dVal);
void OxValSetInt(OxVALUE *pv, int iVal);
void OxValSetNaN(OxVALUE *pv);
void OxValSetMat(OxVALUE *pv, MATRIX mVal, int r, int c);
void OxValSetMatDouble(OxVALUE *pv, int r, int c, double dVal);
void OxValSetMatNaN(OxVALUE *pv, int r, int c);
void OxValSetMatSize(OxVALUE *pv, int r, int c);
void OxValSetMatZero(OxVALUE *pv, int r, int c);
void OxValSetString(OxVALUE *pv, const char *sVal);
void OxValSetString_T(OxVALUE *pv, const TCHAR *sVal);
void OxValSetStringChar(OxVALUE *pv, int len, int chVal);
void OxValSetVecc(OxVALUE *pv, VECTOR vX, int r, int c);
void OxValSetVecr(OxVALUE *pv, VECTOR vX, int r, int c);
```

pv	in:	OxVALUE to set
	out:	changed value
dVal	in:	double value
iVal	in:	integer value
sVal	in:	string value
chVal	in:	character value
mVal[r][c]	in:	matrix value
vX[r×c]	in:	vectorized matrix value

No return value.

Description

OxValSetArray	sets pv to an array of c elements (initialized to null)
OxValSetBlob	sets pv to an opaque type storing two integeres and a pointer
OxValSetDouble	sets pv to a double
OxValSetInt	sets pv to an integer
OxValSetMat	sets pv to a copy of the specified matrix
OxValSetMatDouble	sets pv to a matrix filled with the double value
OxValSetMatNaN	sets pv to a matrix filled with NaN
OxValSetMatSize	sets pv to a matrix with unititialized values
OxValSetMatZero	sets pv to a matrix filled with zeros
OxValSetNaN	sets pv to a double type with value NaN
OxValSetString	sets pv to a string (the string is duplicated)
OxValSetString_T	sets pv to a string (the string is duplicated)
OxValSetStringChar	sets pv to a string of length len filled with chVal
OxValSetVecc	sets pv to a copy of the specified vec of the matrix
OxValSetVecr	sets pv to a copy of the specified vecr of the matrix

All these functions call OxFreeByValue before changing the value. Note that in a function call from Ox, the arguments are set to the value of the call, and the return value is initialized to an integer zero.
A compound type (matrix, string, array) can be freed using OxFreeByValue, but normally that would be left to the Ox run-time system. If pv is not received from Ox, you should set it to an integer before calling these functions, for example:

```
OxVALUE tmp;
OxValInit(&tmp);
OxValSetMatZero(&tmp, 2, 2);
```

Failure to do so could bring down the Ox system.

The len argument of OxValSetStringChar does not include the terminating zero.

The mVal argument of OxValSetMat can be NULL, leaving the matrix unitialized.

OxValSizec, OxValSizer, OxValSizerc

```
int  OxValSizec(OxVALUE *pv);
int  OxValSizer(OxVALUE *pv);
int  OxValSizerc(OxVALUE *pv);
```

pv	in:	OxVALUE to get size of

No return value.

Description

`OxValSizec`	as Ox function `sizec`
`OxValSizer`	as Ox function `sizer`
`OxValSizerc`	as Ox function `sizerc`

OxValTransfer

```
OxVALUE OxValTransfer(OxVALUE *pv)
```

`pv`	out:	Ox variable to transfer

Return value

Transfers an Ox variable. The returned object is by value: if the original was by value it is now not any more (i.e. the contents are stolen, but pv still refers to it); otherwise a duplicate is made.

OxValType

```
int  OxValType(OxVALUE *pv);
```

`pv`	in:	`OxVALUE` to get information from

Return value

returns the type of `pv`.

SetOxExit

```
void SetOxExit(void (OXCALL * pfnNewOxExit)(int) );
```

pfnNewOxExit	in:	new exit handler function

No return value.

Description

Installs a exit handler function for `OxExit` which is called when a run-time error or a fatal error occurs. The default `OxExit` function does nothing.

A run-time error is handled by `OxRunErrorMessage` as follows:

1. Report the text of the error message.
2. If `OxRunError` is called with `iErno` > 1, then call `OxExit(iErno)`.
3. If control is passed on, call `OxExit(0)`.
4. If control is passed on, and Ox is in run-time mode: the run-time engine unwinds and exits after cleaning up (or when interpreting: is ready to accept the next command). If Ox is not in run-time mode: treat as fatal error.

A fatal error is handled as follows:

1. Call `OxExit(1)`.
2. If control is passed on, call `exit(1)`.

Fatal errors can occur during compilation when Ox runs out of memory, or any of the symbol/literal/code tables are full.

SetOxGets

```
void SetOxGets(
    char * (OXCALL * pfnNewOxGets)(char *s, int n) );
```

pfnNewOxGets	in:	new `OxGets` function
s	out:	read input
n	in:	allocated size of s

No return value.

Description

Replaces the `OxGets` function by `pfnNewOxGets`. Is used together with `SetOxPipe` to redirect the output from `scan`.

`pfnNewOxGets` should return to `s` if successful, and `NULL` if it failed.

SetOxMessage

```
void SetOxMessage(
    void (OXCALL * pfnNewOxMessage)(char *) );
```

pfnNewOxMessage	in:	new message handler function

No return value.

Description

Installs a message handler function which is used by `OxMessage`.

SetOxPipe

```
void SetOxPipe(int cPipe);
```

cPipe	in:	> 0: sets pipe buffer size, 0 uses default buffer size, < 0 frees pipe

No return value.

Description

Activates piping of output to another destination than `stdout`. The output from the `print` function will from now on be handled by the `OxPuts` function, and input by `OxGets`. A subsequent attempt for output or input will fail if no new handler for `OxPuts` or `OxGets` has been installed.

SetOxPuts

```
void SetOxPuts(void (OXCALL * pfnNewOxPuts)(char *s) );
```

`pfnNewOxPuts`	in:	new `OxPuts` function
`s`	in:	null-terminated string to output

No return value.

Description

Replaces the `OxPuts` function by `pfnNewOxPuts`. Is used together with `SetOxPipe` to redirect the output from `print`.

SetOxRunMessage

```
void SetOxRunMessage(void (OXCALL * pfnNewOxRunMessage)(char *) );
```

`pfnNewOxRunMessage`	in:	new message handler function

No return value.

Description

Installs a message handler function which is used by `OxRunMessage` and `OxRunErrorMessage`.

SOxGetTypeName

```
char * SOxGetTypeName(int iType);
```

`iType`	in:	type, one of `OX_INT`, `OX_DOUBLE`, `OX_MATRIX`, etc.

Return value

A pointer to the text of the type name.

SOxIntFunc

```
char * SOxIntFunc(void);
```

Return value

A pointer to the name of the currently active internal function.

D6.3 Macros to access OxVALUEs

The OxVALUE is the container for all Ox types. It contains the type identifier, a range of property flags, and the actual data. The type, flags and data can be accessed through functions listed above, or through macros when using C or C++. All constants, types and macros are defined in oxtypes.h. For example, macros are defined to access the type of an OxVALUE:

ISINT(pv)	TRUE if integer type
ISDOUBLE(pv)	TRUE if double type
ISMATRIX(pv)	TRUE if MATRIX type
ISSTRING(pv)	TRUE if string type (array of characters)
ISARRAY(pv)	TRUE if array of OxVALUEs
ISFUNCTION(pv)	TRUE if function type (written in Ox code)
ISCLASS(pv)	TRUE if class object type
ISINTFUNC(pv)	TRUE if internal (library) function
ISFILE(pv)	TRUE if file type
ISLAMBDA(pv)	TRUE if lambda
ISEMPTY(pv)	TRUE if file type
GETPVTYPE(pv)	gets the type of the argument (OxVALUE pointer)
ISARITHMETIC(pv)	TRUE if has OX_INT, OX_DOUBLE, OX_MATRIX
ISCALLABLE(pv)	TRUE if has OX_FUNCTION, OX_INTFUNC, OX_LAMBDA
ISADDRESS(pv)	TRUE if has OX_ADDRESS property

An OxVALUE is a structure which contains a union of other structures. For example when using OxVALUE *pv:

GETPVTYPE(pv)	content	description
OX_INT	pv->type	type and property flags
	pv->t.ival	integer value
OX_DOUBLE	pv->type	type and property flags
	pv->t.dval	double value
OX_MATRIX	pv->type	type and property flags
	pv->t.mval.data	MATRIX value
	pv->t.mval.c	number of columns
	pv->t.mval.r	number of rows
OX_STRING	pv->type	type and property flags
	pv->t.sval.size	string length
	pv->t.sval.data	actual string (null terminated)
OX_ARRAY	pv->type	type and property flags
	pv->t.aval.size	array length
	pv->t.aval.data	pointer to array of OxVALUEs

The macros below provide easy access to these values. They all access an element in an array of OxVALUEs. None of these check the input type, and it is assumed that the correct type is already known.

macro	purpose	input type
OxArray(pv, i)	accesses the array value in pv[i]	OX_ARRAY
OxArrayData(pv)	accesses the array data in pv	OX_ARRAY
OxArrayLen(pv, i)	accesses the array length in pv[i]	OX_ARRAY
OxArraySize(pv)	accesses the array length in pv	OX_ARRAY
OxDbl(pv, i)	accesses the double value in pv[i]	OX_DOUBLE
OxInt(pv, i)	accesses the integer value in pv[i]	OX_INT
OxMat(pv, i)	accesses the MATRIX value in pv[i]	OX_MATRIX
OxMatc(pv, i)	accesses the no of columns in pv[i]	OX_MATRIX
OxMatr(pv, i)	accesses the no of rows in pv[i]	OX_MATRIX
OxMatrc(pv, i)	gets the no of elements in pv[i]	OX_MATRIX
OxSetDbl(pv, i, d)	sets pv[i] to OX_DOUBLE of value d	—
OxSetInt(pv, i, j)	sets pv[i] to OX_INT of value j	—
OxSetMatPtr(pv, i, m, cr, cc)	sets pv[i] to OX_MATRIX pointing to the cr × cc matrix m	—
OxStr(pv, i)	accesses the string value in pv[i]	OX_STRING
OxStringData(pv)	accesses the string data in pv	OX_STRING
OxStrLen(pv, i)	accesses the string length in pv[i]	OX_STRING
OxStrSize(pv)	accesses the string length in pv	OX_STRING
OxZero(pv, i)	sets pv[i] to OX_INT of value 0	—

D6.4 Ox-OxMetrics function summary

This section documents the *Ox* related functions that are specific for use with OxMetrics. These functions are exported from `oxmetrics7.dll`. All functions in this section require `ox_oxmetrics.h`.

FOxOxMetricsStart,FOxOxMetricsStartBatch

```
BOOL FOxOxMetricsStart(LPCTSTR OxModuleName,
    LPCTSTR OxWindowName, BOOL bUseStdHandles);
BOOL FOxOxMetricsStartBatch(LPCTSTR OxModuleName,
    LPCTSTR OxWindowName, BOOL bUseStdHandles, int iBatch);
```

`OxModuleName`	in:	name to be used for module
`OxWindowName`	out:	name of output window in OxMetrics
`bUseStdHandles`	in:	`TRUE`: use standard input/output, else use OxMetrics pipe
`iBatch`	in:	index of batch automation function, use -1 if no batch

Return value

`TRUE` if successful.

Description

These functions establish a link to OxMetrics, and can only be used with OxMetrics under Windows. The required header file is `ox_oxmetrics.h`.
The DLL which is linked to is `oxmetrics7.dll`. It exports the same functionality as OxMetrics, see the OxMetrics Developer's Kit.

OxOxMetricsFinish

```
void OxOxMetricsFinish(BOOL bFocusText);
```

`bFocusText`	in:	`TRUE`: switch to OxMetrics and set focus to the output window

No return value.

Description

Closes the link to OxMetrics, and can only be used with OxMetrics under Windows. The required header file is `ox_oxmetrics.h`. `OxOxMetricsFinish` matches `FOxOxMetricsStart`.

OxMetricsGetOxHandlerA

```
void * OXCALL OxMetricsGetOxHandlerA(const char *sName);
```

`sName`	in:	name of the handler to get

Return value

A pointer to the requested handler.

Description

Use the `sName` argument to get the required OxMetrics handler (function) for Ox:

"OxDraw"
"OxDrawWindow"
"OxMessage"
"OxPuts"
"OxRunMessage"
"OxTextWindow"

OxOxMetricsRestart

```
void OxOxMetricsRestart();
```

No return value.

Description

Restarts the link to OxMetrics after `OxMainInit` is called (e.g. to compile a new Ox file (`OxMainInit` should be called again after `OxMainExit`). `OxMainInit` sets Ox to using Console output (stdout).

OxOxMetricsUsingStdHandles

```
BOOL OxOxMetricsUsingStdHandles();
```

Return value

`TRUE` if `FOxOxMetricsStart` was called using `bUseStdHandles` set to `TRUE`.

D6.5 Ox exported mathematics functions

D6.5.1 `MATRIX` and `VECTOR` types

This section documents the C functions exported from the OxWin DLL to perform mathematical tasks. With the DLL installed, any C or C++ function could call these functions to perform a mathematical task. The primary purpose is, if you, for example, wish to use some random numbers in your C extension to Ox. It is also possible to just use these functions without using Ox at all.

To use any of the functions in this section, you need to include both `jdtypes.h` and `jdmath.h` (in this order), e.g.

```
#include "/ox/dev/jdtypes.h"
#include "/ox/dev/jdmath.h"
```

Or, if you have set up the information for your compiler such that `/ox/dev` is in the include search path:

```
#include "jdtypes.h"
#include "jdmath.h"
```

Several types are defined in `ox/dev/jdtypes.h`, of which the most important are `MATRIX`, `VECTOR` and `BOOL`.

The `MATRIX` type used in this library is a pointer to a column of pointers, each pointing to a row of doubles. A `VECTOR` is just a pointer to an array of doubles. In a `MATRIX`, consecutive rows (the `VECTOR`s) do occupy contiguous memory space (although that would not be strictly necessary in this pointer to array of pointers model). Suppose `m` is a 3 by 3 matrix, then the memory layout can be visualized as:

`m`	$\longrightarrow$	`m[0]`	
`m[0]`	$\longrightarrow$	`m[0][0]`, `m[0][1]`, `m[0][2]`	first row
`m[1]`	$\longrightarrow$	`m[1][0]`, `m[1][1]`, `m[1][2]`	second row
`m[2]`	$\longrightarrow$	`m[2][0]`, `m[2][1]`, `m[2][2]`	third row

Matrices can be manipulated as follows, using the 3×3 matrix `m`:

- `m[0]` is a `VECTOR`, the first row of `m`;
- `&m[1]` is a `MATRIX`, the last two rows of `m`;
- `&m[1][1]` is a `VECTOR`, the last two elements of the second row.
- `&(&m[1])[1]` is a `MATRIX`, the last two elements of the second row (this is only a 1 row matrix, since there is no pointer to the third row).

A `MATRIX` is allocated by a call to `MatAlloc` and deallocated with `MatFree`. For a `VECTOR` the functions are `VecAlloc` and `free`, e.g.:

```
MATRIX m;  VECTOR v;  int i, j;

m = MatAlloc(3, 3);
v = VecAlloc(3);


if (!m || !v)          /* yes: error exit */
    printf("error: allocation failed!");

MatZero(m, 3, 3);           /* set m to 0 */
MatZero(&v, 1, 3);          /* set v to 0 */
```

```
for (i = 0; i < 3; ++i)  /* set both to 1 */
{   for (j = 0; j < 3; ++j)
        m[i][j] = 1;
    v[i] = 1;
}
/* ... do more work                        */

MatFree(m2, 3, 3);   /* done: free memory */
free(v);
```

Note that the memory of a matrix is owned by the original matrix. It is **NOT** safe to exchange rows by swapping pointers. Rows also cannot be exchanged between different matrices; instead the elements must be copied from one row to the other. Columns have to be done element by element as well.

As a final example, we show how to define a matrix which points to part of another matrix. For example, to set up a matrix which points to the 2 by 2 lower right block in `m`, allocate the pointers to rows:

```
MATRIX m2 = MatAlloc(2, 0);
m2[0] = &m[1][1];
m2[1] = &m[2][1];
// do work with m and m2, then free m2:

MatFree(m2, 2, 0);
```

Again note that the memory of the elements is still owned by `m`; deallocating `m` deletes what m2 tries to point to.

When a language supports C-style DLLs, but not the pointer-to-pointer model used in the `MATRIX` type, the following functions may be used to provide the necessary mapping:

`MatAllocBlock`	function version of `MatAlloc`
`MatCopyVecc`	store column-vectorized matrix in a `MATRIX`
`MatCopyVecr`	store row-vectorized matrix in a `MATRIX`
`MatFreeBlock`	function version of `MatFree`
`MatGetAt`	get an element in a `MATRIX`
`MatSetAt`	set an element in a `MATRIX`
`VeccCopyMat`	store a `MATRIX` as a column vector
`VecrCopyMat`	store a `MATRIX` as a row vector

D6.5.2 Exported matrix functions

The following list gives the exported C functions, with their Ox equivalent.

`c_abs`	`cabs`
`c_div`	`cdiv`
`c_erf`	`cerf`
`c_exp`	`cexp`
`c_log`	`clog`
`c_mul`	`cmul`
`c_sqrt`	`csqrt`

DBessel01	bessel
DBesselnu	bessel
DBetaFunc	betafunc
DDawson	dawson
DDensBeta	densbeta
DDensChi	denschi
DDensF	densf
DDensGamma	densgamma
DDensGH	densgh
DDensGIG	densgig
DDensMises	densmises
DDensNormal	densn
DDensPoisson	denspoisson
DDensT	denst
DDiagXSXt	outer
DDiagXtSXtt	outer
DErf	erf
DExpInt	expint
DExpInt1	expint
DExpInte	expint
DGammaFunc	gammafunc
DGamma	gammafact
DGetInvertEps	inverteps
DGetInvertEpsNorm	
DLogGamma	loggamma
DPolyGamma	polygamma
DProbBeta	probbeta
DProbBVN	probbvn
DProbChi	probchi
DProbChiNc	probchi
DProbF	probf
DProbFNc	probf
DProbGamma	probgamma
DProbMises	probmises
DProbMVN	probmvn
DProbNormal	probn
DProbPoisson	probpoisson
DProbT	probt
DProbTNc	probt
DQuanBeta	quanbeta
DQuanChi	quanchi
DQuanF	quanf
DQuanGamma	quangamma
DQuanMises	quanmises
DQuanNormal	quann
DQuanT	quant

DRanBeta	ranbeta
DRanChi	ranchi
DRanExp	ranexp
DRanF	ranf
DRanGamma	rangamma
DRanGIG	rangig
DRanInvGaussian	raninvgaussian
DRanLogNormal	ranlogn
DRanLogistic	ranlogistic
DRanMises	ranmises
DRanNormalPM	rann
DRanStable	ranstable
DRanT	rant
DRanU	ranu
DRanU	ranu
DTailProbChi	tailchi
DTailProbF	tailf
DTailProbNormal	tailn
DTailProbT	tailt
DTraceAB	trace(AB)
DTrace	trace
DVecsum	sumr(A)
DecQRtMul	decqrmul
EigVecDiv	
FCubicSpline	spline
FFT1d	fft1d
FftComplex	fft
FftDiscrete	dfft
FftReal	fft
FIsInf	isinf
FIsNaN	isnan
FPPtDec	choleski
FPeriodogram	periodogram
FPeriodogramAcf	
IDecQRt	decqr
IDecQRtEx	decqr
IDecQRtRank	decqr
IDecSVD	decsvd
IEigValPoly	polyroots
IEigen	eigen
IEigenSym	eigensym
IGenEigVecSym	eigensymgen
IGetAcf	acf
IInvDet	invert
IInvert	invert
ILDLbandDec	decldlband

ILDLdec	decldl
ILUPdec	declu, determinant
ILUPlogdet	declu, determinant
IMatRank	rank
INullSpace	nullspace
IOlsNorm	ols2c, ols2r
IOlsQR	ols2, ols2
IRanBinomial	ranbinomial
IRanLogarithmic	ranlogarithmic
IRanNegBin	rannegbin
IRanPoisson	ranpoisson
ISymInv	invertsym
ISymInvDet	invertsym
IntMatAlloc	
IntMatFree	
IntVecAlloc	
LDLbandSolve	solveldlband
LDLsolve	solveldl
LDLsolveInv	solveldl
LUPsolve	solvelu
LUPsolveInv	solvelu
MatABt	A*B’
MatAB	A*B
MatAcf	acf
MatAdd	A+c*B
MatAllocBlock	
MatAlloc	
MatAtB	A’B
MatBBt	BB’
MatBSBt	BSB’
MatBtBVec	A=B-y; A’A
MatBtB	B’B
MatBtSB	B’SB
MatCopyTranspose	
MatCopyVecc	
MatCopyVecr	
MatCopy	
MatDup	A = B
MatFreeBlock	
MatFree	
MatGenInvert	1 / A, decsvd
MatGenInvertSym	1 / A, decsvd
MatGetAt	
MatI	unit
MatNaN	
MatRan	ranu

MatRann	rann
MatReflect	reflect
MatSetAt	
MatStandardize	standardize
MatTranspose	transpose operator: ’
MatVariance	variance
MatZero	zeros
MatZero	zeros
OlsQRacc	ols
RanDirichlet	randirichlet
RanGetSeed	ranseed
RanSetRan	ranseed
RanSetSeed	ranseed
RanSubSample	ransubsample
RanUorder	ranuorder
RanWishart	ranwishart
SetFastMath	use command line switch to turn off
SetInf	= M_INF
SetInvertEps	inverteps
SetNaN	= M_NAN
SortVec	sortr
SortMatCol	sortc
SortmXByCol	sortbyc
SortmXtByVec	sortbyr
ToeplitzSolve	solvetoeplitz
VecAllocBlock	
VecCopy	
VecDiscretize	discretize
VecDupBlock	
VecFreeBlock	
VecTranspose	
VeccCopyMat	
VecrCopyMat	

D6.5.3 Matrix function reference

c_abs, c_div, c_erf, c_exp, c_log, c_mul, c_sqrt

```
double c_abs(double xr, double xi);
BOOL c_div(double xr, double xi, double yr, double yi,
    double *zr, double *zi);
void c_erf(double x, double y, double *erfx, double *erfy);
void c_exp(double xr, double xi, double *yr, double *yi);
void c_log(double xr, double xi, double *yr, double *yi);
void c_mul(double xr, double xi, double yr, double yi,
    double *zr, double *zi);
```

```
void c_sqrt(double xr, double xi, double *yr,double *yi);
```

Return value

`c_abs` returns the result. `c_div` returns FALSE in an attempt to divide by 0, TRUE otherwise. The other functions have no return value.

DBessel01, DBesselNu

```
double DBessel01(double x, int type, int n);
double DBesselNu(double x, int type, double nu);
```

`x`	in:	x, point at which to evaluate
`type`	in:	character, type of Bessel function: `'J'`, `'Y'`, `'I'`, `'K'`
`n`	in:	integer, 0 or 1: order of Bessel function
`nu`	in:	double, fractional order of Bessel function

Return value

Returns the Bessel function.

DBetaFunc

```
double DBetaFunc(double dX, double dA, double dB);
```

Return value

Returns the incomplete beta function $B_x(a, b)$.

DDawson

```
double DDawson(double x);
```

Return value

Returns the Dawson integral.

DDens...

```
double DDensBeta(double x, double a, double b);
double DDensChi(double x, double dDf);
double DDensF(double x, double dDf1, double dDf2);
double DDensGamma(double g, double r, double a);
double DDensGH(double dX, double dNu, double dDelta,
    double dGamma, double dBeta);
double DDensGIG(double dX, double dNu, double dDelta,
    double dGamma);
double DDensMises(double x, double dMu, double dKappa);
double DDensNormal(double x);
double DDensPoisson(double dMu, int k);
double DDensT(double x, double dDf);
```

Return value

Value of density at `x`.

DecQRtMul

```
void DecQRtMul(MATRIX mQt, int cX, int cT, MATRIX mY, int cY,
    int cR);
void DecQRtMult(MATRIX mQt, int cX, int cT, MATRIX mYt, int cY,
    int cR);
```

`mQt[cX][cT]`	in:	householder vectors of QR decomposition of X
`mYt[cY][cT]`	in:	matrix Y
	out:	$Q'Y$
`mY[cT][cY]`	in:	matrix Y
	out:	$Q'Y$
`cR`	in:	row rank of X'

Return value

Computes $Q'Y$.

Description

Performs multiplication by Q' after a QR decomposition.

DDiagXSXt, DDiagXtSXtt

```
double DDiagXSXt(int iT, MATRIX mX, MATRIX mS, int cS);
double DDiagXtSXtt(int cX, MATRIX mXt, MATRIX mS, int cS);
```

`mXt[cX][cS]`	in:	matrix X
`mX[cS][cX]`	in:	matrix X'
`mS[cS][cS]`	in:	symmetric matrix S

Return value

DDiagXtSXtt returns $Xt[][iT]'SXt[][iT]$; DDiagXSXt returns $X[iT][]SX[iT][]'$.

Description

Performs multiplication by Q' after a QR decomposition.

DErf, DExpInt, DExpInte, DExpInt1

```
double DErf(double x);
double DExpInt(double x);
double DExpInte(double x);
double DExpInt1(double x);
```

Return value

`DErf` returns the error function erf(x).

`DExpInt` returns the exponential integral Ei(x).

`DExpInte` returns the exponential integral $\exp(-x)$Ei(x).

`DExpInt1` returns the exponential integral E1(x).

DGamma, DGammaFunc

```
double DGamma(double z);
double DGammaFunc(double dX, double dR);
```

Return value

`DGamma` returns the complete gamma function $\Gamma(z)$.

`DGammaFunc` returns the incomplete gamma function $G_x(r)$.

DGetInvertEps

```
double DGetInvertEps(void);
double DGetInvertEpsNorm(MATRIX mA, int cA);
```

Return value

`DGetInvertEps` returns inversion epsilon, ϵ_{inv}, see `SetInvertEps`.

`DGetInvertEpsNorm` returns $\epsilon_{inv}||A||_\infty$.

DLogGamma

```
double DLogGamma(double dA);
```

Return value

Returns the logarithm of the gamma function.

DPolyGamma

```
double DPolyGamma(double dA, int n);
```

Return value

Returns the derivatives of the loggamma function; $n = 0$ is first derivative: digamma function, and so on.

DProb...

```
double DProbBeta(double x, double a, double b);
double DProbBVN(double dLo1, double dLo2, double dRho);
double DProbChi(double x, double dDf);
double DProbChiNc(double x, double df, double dNc);
double DProbF(double x, double dDf1, double dDf2);
double DProbFNc(double x, double dDf1, double dDf2, double dNc);
double DProbGamma(double x, double dR, double dA);
double DProbMises(double x, double dMu, double dKappa);
double DProbMVN(int n, VECTOR vX, MATRIX mSigma);
double DProbNormal(double x);
double DProbPoisson(double dMu, int k);
double DProbT(double x, int iDf);
double DProbTNc(double x, double dDf, double dNc);
```

Return value

Probabilities of value less than or equal to `x`.

DQuan...

```
double DQuanBeta(double x, double a, double b);
double DQuanChi(double p, double dDf);
double DQuanF(double p, double dDf1, double dDf2);
double DQuanGamma(double p, double r, double a);
double DQuanMises(double p, double dMu, double dKappa);
double DQuanNormal(double p);
double DQuanT(double p, int iDf);
double DQuanTD(double p, double dDf)
```

Return value

Quantiles at `p`.

DRan...

```
double DRanBeta(double a, double b);
double DRanChi(double dDf);
double DRanExp(double dLambda);
double DRanF(double dDf1, double dDf2);
double DRanGamma(double dR, double dA);
```

```
double DRanGIG(double dNu, double dDelta, double dGamma);
double DRanInvGaussian(double dMu, double dLambda);
double DRanLogNormal(void);
double DRanLogistic(void);
double DRanMises(double dKappa);
double DRanNormalPM(void);
double DRanStable(double dA, double dB);
double DRanStudentT(double dDf)
double DRanT(int iDf);
double DRanU();
```

Return value

Returns random numbers from various distributions.

`DRanU` generates uniform $(0, 1)$ pseudo random numbers according to the active generation method (see `RanSetRan`).

`DRanNormalPM` standard normals (PM = polar-Marsaglia).

Note that, if the Ox run-time is bypassed, and this functions is called directly, the application will crash unless `RanSetRan` is called to initialize the random number environment

DTail...

```
double DTailProbChi(double x, double dDf);
double DTailProbF(double x, double dDf1, double dDf2);
double DTailProbNormal(double x);
double DTailProbT(double x, int iDf);
```

Return value

Probabilities of values greater than `x`.

DTrace, DTraceAB

```
double DTrace(MATRIX mat, int cA);
double DTraceAB(MATRIX mA, MATRIX mB, int cM, int cN);
     mA[cM][cN]          in:  matrix
     mB[cN][cM]          in:  matrix
```

Return value

DTrace returns the trace of A.

DTraceAB returns the trace of AB.

DVecsum

```
double DVecsum(VECTOR vA, int cA);
     vA[cA]              in:  vector
```

Return value

DVecsum returns the sum of the elements in the vector.

EigVecDiv

```
void EigVecDiv(MATRIX mE, VECTOR vEr, VECTOR vEi, int cA)
     vEr[cA]             out: real part of eigenvalues
     vEi[cA]             out: imaginary part of eigenvalues
     mE[cA][cA]          in:  matrix with eigenvectors in rows
                         out: rescaled eigenvectors
```

Return value
Scales each eigenvector (in rows) with the largest row element.

FCubicSpline

```
BOOL FCubicSpline(VECTOR vY, VECTOR vT, int cT, double *pdAlpha,
    VECTOR vG, VECTOR vX, double *pdCV, double *pdPar, BOOL fAuto,
    int iDesiredPar);
BOOL FCubicSplineTime(VECTOR vY, int cT, double dAlpha, VECTOR vG,
    BOOL fHP)
```

`vY[cT]`	in:	variable of which to compute spline
`vT[cT]`	in:	x-variable or NULL (then against time)
`cT`	in:	number of observations, T
`dAlpha`	in:	bandwidth parameter (if $\leq$ 1e-20: 1600 is used)
`vG[cT]`	out:	natural cubic spline, according to `vY` (unsorted), unless `vX` is given, in which case it is according to `vX` (the sorted `vT`)
`pdCV`	in:	NULL or pointer
	out:	cross-validation value
`vX[cT]`	in:	NULL or vector
	out:	xaxis (sorted `vT`) for drawing, only if `vT != NULL`
`pdPar`	in:	NULL or pointer
	out:	equivalent number of parameters
`iDesiredPar`	in:	desired equivalent no of parameters or 0
`fHP`	in:	FALSE: use spline, TRUE: Hodrick-Prescott

Return value
Returns TRUE if successful, FALSE if out of memory.
`FCubicSpline` fits a natural cubic spline to a scatter, skips missing values. `FCubicSplineTime` fits a natural cubic spline to evenly spaced data, *not* skipping over missing values.

FFT1d, FftComplex, FftReal, FftDiscrete

```
int  FFT1d(MATRIX mDest, MATRIX mSrc, int cM, int iForward,
    int isComplex)
void FftComplex(VECTOR vXr, VECTOR vXi, int iPower, int iForward);
void FftReal(VECTOR vXr, VECTOR vXi, int iPower, int iForward);
BOOL FftDiscrete(VECTOR vXr, VECTOR vXi, int cN, int iForward);
```

mDest[c_1][cN]	in:	matrix; $c_1 = 2$ unless isComplex=FALSE and iForward= 0
	out:	FFT (or inverse FFT) first vector is real part, second imaginary
mSrc[c_2][cN]	in:	data matrix, first vector is real part, second imaginary; $c_2 = 2$ unless isComplex=FALSE and iForward= 1
	out:	unchanged
vXr[n]	in:	vector with real part, $n = 2^{\texttt{iPower}}$ (discrete FFT: $n =$ cN)
	out:	FFT (or inverse FFT) real part
vXi[n]	in:	vector with imaginary part, $n = 2^{\texttt{iPower}}$ (discrete FFT: $n =$ cN)
	out:	FFT (or inverse FFT) imaginary part
iPower	in:	the vector sizes is $2^{\texttt{iPower}}$
iForward	in:	indicates whether an FFT (iForward $= 1$) or an inverse FFT must be performed (iForward $= 0$)

Return value

FFT1d and FftDiscrete return FALSE if there is not enough memory, TRUE otherwise. Also see under fft and dfft.

FIsInf, FIsNaN

```
BOOL FIsNaN(double d);
BOOL FIsInf(double d);
```

d	in:	value to check

Description

Returns TRUE if the argument is infinity (.Inf) or not-a-number (.NaN) respectively.

FPeriodogram, FPeriodogramAcf

```
BOOL FPeriodogram(VECTOR vX, int cT, int iTrunc, int cS,
    VECTOR vS, int iMode);
BOOL FPeriodogramAcf(VECTOR vAcf, int cT, int iTrunc, int cS1,
    VECTOR vS, int iMode, int cTwin)
```

vX[cT]	in:	variable of which to compute correlogram
cT	in:	number of observations, T
iTrunc	in:	truncation parameter m
cS	in:	no of points at which to evaluate spectrum
vS[cS]	out:	periodogram
iMode	in:	0: (truncated) periodogram, 1: smoothed periodogram using Parzen window, 2: estimated spectral density using Parzen window (as option 1, but divided by c(0)).
vAcf[cT]	in:	ACF
	out:	overwritten by weighted ACF
cS1	in:	> 0: no of points at which to evaluate spectrum ≤ 0: using all points with window $2\pi/$cTwin

Return value
Returns TRUE if successful, FALSE if out of memory.

FPPtDec

```
BOOL FPPtDec(MATRIX mA, int cA)
```

`mA[cA][cA]`	in:	symmetric p.d. matrix to be decomposed
	out:	contains P

Return value
`TRUE`: no error;
`FALSE`: Choleski decomposition failed.

Description
Computes the Choleski decomposition of a symmetric pd matrix A: $A = PP'$. P has zeros above the diagonal.

IDecQRt...

```
int IDecQRt(MATRIX mXt, int cX, int cT, int *piPiv, int *pcR);
int IDecQRtEx(MATRIX mXt, int cX, int cT, int *piPiv, VECTOR vTau);
int IDecQRtRank(MATRIX mQt, int cX, int cT, int *pcR);
```

`mXt[cX][cT]`	in:	X' data matrix
	out:	householder vectors of QR decomposition of X, holds H in lower diagonal, and R in upper diagonal
`piPiv[cX]`	in:	allocated vector or NULL
	out:	pivots (if argument is NULL on input, there will be no pivoting)
`pcR`	in:	pointer to integer
	out:	row rank of X'
`vTau[cX]`	in:	allocated vector
	out:	$-2/h'h$ for each vector h of H
`mQt[cX][cT]`	in:	output from `IDecQRtEx`

Return value
`IDecQRtEx` returns 1 if successful, 0 if out of memory. `IDecQRt` and `IDecQRtRank` return:

0:	out of memory,
1:	success,
2:	ratio of diagonal elements of $(X'X)$ is large, rescaling is adviced,
-1:	$(X'X)$ is (numerically) singular,
-2:	combines 2 and -1.

Description
Performs QR decomposition. `IDecQRt` amounts to a call to `IDecQRtEx` followed by `IDecQRtRank` to determine the rank and return value.

IDecSVD

```
int IDecSVD(MATRIX mA, int cM, int cN, VECTOR vW, int fDoU,
    MATRIX mU, int fDoV, MATRIX mV, int fSort);
```

`mA[cM][cN]`	in:	matrix to decompose, `cM` $\geq$ `cN`
	out:	unchanged
`vW[cN]`	in:	vector
	out:	the n (non-negative) singular values of A
`fDoU`	in:	TRUE: U matrix of decomposition required
`mU[cM][cN]`	in:	matrix
	out:	the matrix U (orth column vectors) of the decomposition if `fDoU == TRUE`. Otherwise used as workspace. `mU` may coincide with `mA`.
`fDoV`	in:	`TRUE`: V matrix required
`mV[cM][cN]`	in:	matrix
`mV[cN][cN]`	out:	the matrix V of the decomposition if `fDoV == TRUE`. Otherwise not referenced. `mV` may coincide with `mU` if `mU` is not needed.
`fSort`	in:	if `TRUE` the singular values are sorted in decreasing order with U, V accordingly.

Return value

0: success

k: if the k-th singular value (with index k - 1) has not been determined after 50 iterations. The singular values and corresponding U, V should be correct for indices $\geq$ k.

Description

Computes the singular value decomposition.

IEigValPoly,IEigen

```
int IEigValPoly(VECTOR vPoly, VECTOR vEr, VECTOR vEi, int cA);
int IEigen(MATRIX mA, int cA, VECTOR vEr, VECTOR vEi, MATRIX mE);
```

`vPoly[cA]`	in:	coefficients of polynomial $a_1 \dots a_m$ ($a_0 = 1$).
	out:	unchanged.
`mA[cA][cA]`	in:	unsymmetric matrix.
	out:	used as working space. `IEigVecReal`: holds eigenvecs in rows (eigenvalue i is complex: row i is real, row $i + 1$ is imaginary part).
`vEr[cA]`	out:	real part of eigenvalues
`vEi[cA]`	out:	imaginary part of eigenvalues
`mE[cA][cA]`	in:	NULL or matrix.
	out:	if !NULL: holds eigenvecs in rows (eigenvalue i is complex: row i is real, row $i + 1$ is imaginary part).

Return value

0 success

1 maximum no of iterations (50) reached

2 NULL pointer arguments or memory allocation not succeeded.

Description

`IEigValPoly` computes the roots of a polynomial, see `polyroots()`.

`IEigen` computes the eigenvalues and optionally the eigenvectors of a double unsymmetric matrix. On output, the eigenvectors are *not* standardized by the largest

element. EigVecDiv can be used for standardization: it takes the eigenvectors and values from IEigen as input, and gives the standardized eigenvectors on output.

IEigenSym

```
int  IEigenSym(MATRIX mA, int cA, VECTOR vEval, int fDoVectors);
```

mA[cA][cA]	in:	symmetric matrix.
	out:	work space.
if fDoVectors $\neq$ 0: the rows contain the normalized eigenvectors (ordered).		
vEv[cA]	out:	ordered eigenvalues (smallest first)
fDoVectors	in:	eigenvectors are to be computed

Return value

See IEigen.

Description

IEigenSym computes the eigenvalues of a symmetric matrix, and optionally the (normalized) eigenvectors.

IGenEigVecSym

```
int  IGenEigVecSym(MATRIX mA, MATRIX mB, VECTOR vEval,
    VECTOR vSubd, int cA);
```

mA[cA][cA]	in:	symmetric matrix.
	out:	the rows contain the normalized eigenvectors (sorted according to eigenvals, largest first)
mB[cA][cA]	in:	symmetric pd. matrix.
	out:	work
vEval[cA]	out:	ordered eigenvalues (smallest first)
vSubd[cA]	out:	index of ordered eigenvalues
cA	in:	dimension of matrix;

Return value

0,1,2: see IEigen; -1: Choleski decomposition failed.

Description

Solves the general eigenproblem $Ax = \lambda Bx$, where A and B are symmetric, B also positive definite.

IGetAcf

```
int IGetAcf(VECTOR vX, int cT, int cLag, VECTOR vAcf, BOOL bCov);
```

vX[cT]	in:	variable of which to compute correlogram
cT	in:	number of observations
cLag	in:	required no of correlation coeffs
vAcf[cLag]	out:	correlation coeffs 1... cLag (0. if failed); unlike acf(), the autocorrelation at lag 0 (which is 1) is not included.
bCov	in:	FALSE: autocorrelation, else autocovariances

Return value

`IGetAcf` uses the full sample means (the standard textbook correlogram). `IGetAcf` skips over missing values, in contrast to `MatAcf`. Also see under `acf` and `DrawCorrelogram`.

IInvert, IInvDet

```
int  IInvert(MATRIX mA, int cA);
int  IInvDet(MATRIX mA, int cA, double *pdLogDet, int *piSignDet);
```

`mA[cA][cA]`	in:	ptr to matrix to be inverted
	out:	contains the inverse, if successful
`pdLogDet`	out:	the *logarithm* of the absolute value of the determinant of A
`piSignDet`	out:	the sign of the determinant of A; 0: singular; $-1, -2$: negative determinant; $+1, +2$: positive determinant; $-2, +2$: result is unreliable

Return value

0: success; 1,2,3: see `ILDLdec`.

Description

Computes inverse of a matrix using LU decomposition.

ILDLbandDec

```
int ILDLbandDec(MATRIX mA, VECTOR vD, int cB, int cA);
```

`mA[cB][cA]`	in:	ptr to sym. pd. band matrix to be decomposed
	out:	contains the L matrix (except for the 1's on the diagonal)
`vD[cA]`	out:	the reciprocal of D (not the square root!)
`cB`	in:	1+bandwidth

Return value

See `ILDLdec`.

Description

Computes the Choleski decomposition of a symmetric positive band matrix. The matrix is stored as in `decldlband`.

ILDLdec

```
int  ILDLdec(MATRIX mA, VECTOR vD, int cA);
```

`mA[cA][cA]`	in:	ptr to sym. pd. matrix to be decomposed only the lower diagonal is referenced;
	out:	the strict lower diagonal of A contains the L matrix (except for the 1's on the diagonal)
`vD[cA]`	out:	the reciprocal of D (not the square root!)

Return value

0 no error;
1 the matrix is negative definite;
2 the matrix is (numerically) singular;
3 NULL pointer argument

Description

Computes the Choleski decomposition of a symmetric positive definite matrix.

ILUPdec

```
int ILUPdec(MATRIX mA, int cA, int *piPiv, double *pdLogDet,
    int *piSignDet, MATRIX mUt);
```

`mA[cA][cA]`	in:	ptr to matrix to be decomposed
	out:	the strict lower diagonal of A contains the L matrix (except for the 1's on the diagonal) the upper diagonal contains U.
`piPiv[cA]`	out:	the pivot information
`pdLogDet`	out:	the *logarithm* of the absolute value of the determinant of A
`piSignDet`	out:	the sign of the determinant of A; 0: singular; $-1, -2$: negative determinant; $+1, +2$: positive determinant; $-2, +2$: result is unreliable
`mUt[cA][cA]`	in:	NULL or matrix
	out:	used as workspace

Return value

0	no error;
-1	out of memory;
≥ 1	the matrix is (numerically) singular; the return value is one plus the singular pivot.

Description

Computes the LU decomposition of a matrix A as: PA = LU.

ILUPlogdet

```
int ILUPlogdet(MATRIX mU, int cA, int *piPiv, double dNormEps,
    double *pdLogDet);
```

`mU[cA][cA]`	in:	LU matrix, only diagonal elements are used
`piPiv[cA]`	in:	the pivot information (NULL: no pivoting)
`dNormEps`	in:	norm(A)*eps, use result from DGetInvertEpsNorm on original matrix A
`pdLogDet`	out:	the *logarithm* of the absolute value of the determinant of A

Return value

Returns the sign of the determinant of $A = LUP$; 0: singular; $-1, -2$: negative determinant; $+1, +2$: positive determinant; $-2, +2$: result is unreliable.

Description

Computes the log-determinant from the LU decomposition of a matrix A.

IMatRank

```
int IMatRank(MATRIX mA, int cM, int cN, double dEps,
    BOOL bAbsolute);
```

`mA[cM][cN]`	in:	`cM` by `cN` matrix of rank `cN`
	out:	unchanged
`dEps`	in:	tolerance to use
`bAbsolute`	in:	`TRUE`: use `dEps`, `FALSE`: `dEps` $\times$ norm

Return value

-1: failure: out of memory; -2: failure: couldn't find all singular values;

≥ 0: rank of matrix.

Description

Uses `IDecSVD` to find the rank of an $m \times n$ matrix A.

IntMatAlloc, IntMatFree, IntVecAlloc

```
INTMAT IntMatAlloc(int cM, int cN);
void IntMatFree(INTMAT im, int cM, int cN);
INTVEC IntVecAlloc(int cM);
```

`cM`, `cN`	in:	required matrix dimensions

Return value

`IntMatAlloc` returns a pointer to the newly allocated `cM` $\times$ `cN` matrix of integers (`INTMAT` corresponds to `int **`), or `NULL` if the allocation failed, or if `cM` was 0. Use `IntMatFree` to free such a matrix.

`IntVecAlloc` returns a pointer to the newly allocated `cM` vector of integers (`INTVEC` corresponds to `int *`), or `NULL` if the allocation failed, or if `cM` was 0. Use the standard C function `free` to free such a matrix.

The allocated types are a matrix or vector of *integers*; there is no corresponding type in Ox, and the allocated matrix cannot be passed directly to Ox code. Also note that these are implemented as macros.

INullSpace

```
int INullSpace(MATRIX mA, int cM, int cN, BOOL fAppend);
```

`mA[cM][cM]`	in:	`cM` by `cN` matrix of rank `cN`, `cM` $>$ `cN` (allocated size must be `cM` by `cM`)
	out:	null space of A is appended (fAppend==TRUE) or mA is overwritten by null space.

Return value

-1: failure: couldn't find all singular values, or out of memory;
≥ 0: rank of null space.

Description

Uses `IDecSVD` to find the orthogonal complement A^*, $m \times m - n$, of an $m \times n$ matrix A of rank n, $n < m$, such that $A^{*\prime}A^* = I, A^{*\prime}A = 0$.

Note that the append option requires that A has full column rank (if not the last $m - n$ columns of U are appended).

IOlsNorm, IOlsQR, OlsQRacc

```
int IOlsNorm(MATRIX mXt, int cX, int cT, MATRIX mYt, int cY,
    MATRIX mB, MATRIX mXtXinv, MATRIX mXtX, BOOL fInRows);
```

`mXt[cX][cT]`	in:	X data matrix
	out:	unchanged
`mYt[cY][cT]`	in:	Y data matrix
	out:	unchanged
`mB[cY][cX]`	in:	allocated matrix
	out:	coefficients
`mXtXinv[cX][cX]`	in:	allocated matrix or NULL
	out:	$(X'X)^{-1}$ if !NULL
`mXtX[cX][cX]`	in:	allocated matrix or NULL
	out:	$X'X$ if !NULL
`fInRows`	in:	if FALSE, input is `mXt[cT][cX]`, `mYt[cT][cY]`

```
int IOlsQR(MATRIX mXt, int cX, int cT, MATRIX mYt, int cY,
    MATRIX mB, MATRIX mXtXinv, MATRIX mXtX, VECTOR vW);
```

`mXt[cX][cT]`	in:	X data matrix
	out:	QR decomposition of X, but only if all three return arguments `mB`, `mXtXinv`, `mXtX` are NULL
`mYt[cY][cT]`	in:	Y data matrix
	out:	$Q'Y$
`mB[cY][cX]`	in:	allocated matrix or NULL
	out:	coefficients if !NULL
`mXtXinv[cX][cX]`	in:	allocated matrix or NULL
	out:	$(X'X)^{-1}$ if !NULL
`mXtX[cX][cX]`	in:	allocated matrix or NULL
	out:	$X'X$ if !NULL
`vW[cT]`	in:	vector
	out:	workspace

Return value

0: out of memory,
1: success,
2: ratio of diagonal elements of $(X'X)$ is large, rescaling is adviced,
-1: $(X'X)$ is (numerically) singular,
-2: combines 2 and -1.

```
void OlsQRacc(MATRIX mXt, int cX, int cT, int *piPiv, int cR,
    VECTOR vTau, MATRIX mYt, int cY, MATRIX mB, MATRIX mXtXinv,
    MATRIX mXtX)
```

`mXt[cX][cT]`	in:	result from `IDecQRt`
	out:	may have been overwritten
`piPiv[cX]`	in:	pivots (output from `IDecQRt`)
`pcR`	in:	row rank of X' (output from `IDecQRt`)
`vTau[cX]`	in:	scale factors (output from `IDecQRt`)
...		other arguments are as for `IOlsQR`

Description

performs ordinary least squares (OLS).

IRanBinomial, IRanLogarithmic, IRanNegBin, IRanPoisson

```
int IRanBinomial(int n, double p);
int IRanLogarithmic(double dA);
int IRanNegBin(int iN, double dP);
int IRanPoisson(double dMu);
```

Return value

Returns random numbers from Binomial/Logarithmic/Negative binomial/Poisson distributions.

ISymInv,ISymInvDet

```
int ISymInv(MATRIX mA, int cA);
int ISymInvDet(MATRIX mA, int cA, double *pdLogDet);
```

`mA[cA][cA]`	in:	ptr to sym. pd. matrix to be inverted
	out:	contains the inverse, if successful
`pdLogDet`	in:	address of double or NULL
	out:	contains the log determinant (if not NULL on input)

Return value

0: success; 1,2,3: see `ILDLdec`.

LDLbandSolve

```
void LDLbandSolve(MATRIX mL, VECTOR vD, VECTOR vX, VECTOR vB,
    int cB, int cA);
```

`mL[cB][cA]`	in:	L from calling `ILDLbandDec`
`vD[cA]`	in:	the reciprocal of D
`vX[cA]`	out:	the solution vX (if (`vX == vB`) then vB is overwritten by the solution)
`vB[cA]`	in:	pointer containing the r.h.s. of $Lx = b$
`cB`	in:	1+bandwidth

No return value.

Description

Solves $Ax = b$, with $A = LDL'$ a symmetric positive definite band matrix.

LDLsolve

```
void LDLsolve(MATRIX mL, VECTOR vD, VECTOR vX, VECTOR vB, int cA);
```

`mL[cA][cA]`	in:	ptr to a matrix of which the strict lower diagonal must contain L from the Choleski decomposition computed using `ILDLdec`. (the upper diagonal is not referenced);
`vD[cA]`	in:	contains the reciprocal of D
`vX[cA]`	in:	pointer containing the r.h.s. of $Lx = b$;
`vB[cA]`	out:	contains the solution x (if (`vX == vB`) then vB is overwritten by the solution)

No return value.

Description

Solves $Ax = b$, with $A = LDL'$ a symmetric positive definite matrix.

LDLsolveInv

```
void LDLsolveInv(MATRIX mLDLt, MATRIX mAinv, int cA);
```

`mLDLt[cA][cA]`	in:	ptr to a matrix holding $L : L'$ with $1/D$ on the diagonal
`mAinv[cA][cA]`	in:	ptr to a matrix.
	out:	contains the inverse

No return value.

Description

Computes the inverse of a symmetric matrix A, L, D must be the Choleski decomposition.

LUPsolve, LUPsolveInv

```
void LUPsolve(MATRIX mL, MATRIX mU, int *piPiv, VECTOR vB,int cA);
void LUPsolveInv(MATRIX mL, MATRIX mU, int *piPiv, MATRIX mAinv,
    int cA);
```

`mL[cA][cA]`	in:	the strict lower diagonal contains the L matrix (except for the 1's on diag, so that `mL` and `mU` may coincide)
`mU[cA][cA]`	in:	the upper diagonal contains U: $PA = LU$ output from `ILUPdec`.
`piPiv[cA]`	in:	the pivot information (P)
`vB[cA]`	in:	rhs vector of system to be solved: $Ax = b$.
	out:	contains x.
`mAinv[cA][cA]`	in:	ptr to a matrix.
	out:	contains the inverse of A

No return value.

Description

Solves $AX = B$, with $A = LU$ a square matrix. Normally, this will be preceded by a call to `ILUPdec`. That function returns LU stored in one matrix, which can then be used for both `mL` and `mU`.

MatAcf

```
MATRIX MatAcf(MATRIX mAcf, MATRIX mX, int cT, int cX, int mxLag);
```

`mAcf[mxLag+1] [cX]`	out:	correlation coefficients (0. if failed)
`mX[cT][cX]`	in:	variable of which to compute correlogram
`cT`	in:	number of observations
`mxLag`	in:	required no of correlation coeffs

Return value

Returns `mAcf` if successful, NULL if not enough observations.

MatAdd

```
MATRIX MatAdd(MATRIX mA, int cM, int cN, MATRIX mB, double dFac,
    MATRIX mAplusB);
```

`mA[cM][cN]`	in:	matrix A
`mB[cM][cN]`	in:	matrix B
`dFac`	in:	scalar c
`mAplusB[cM][cN]`	out:	$A + cB$

Return value
returns `mAplusB` $=A+cB$.

MatAB, MatABt, MatAtB, MatBSBt, MatBtSB, MatBBt, MatBtB, MatBtBVec

```
MATRIX MatAB(MATRIX mA, int cA, int cC, MATRIX mB,int cB,mat mAB);
```
`mA[cA][cC]` in: matrix A
`mB[cC][cB]` in: matrix B
`mAB[cA][cB]` out: AB

```
MATRIX MatABt(MATRIX mA, int cA, int cC, MATRIX mB,
    int cB, mat mABt);
```
`mA[cA][cC]` in: matrix A
`mB[cB][cC]` in: matrix B
`mABt[cA][cB]` out: AB'

```
MATRIX MatAtB(MATRIX mA, int cA, int cC, MATRIX mB,
    int cB, mat mAtB);
```
`mA[cA][cC]` in: matrix A
`mB[cA][cB]` in: matrix B
`mAtB[cC][cB]` out: $A'B$

```
MATRIX MatBBt(MATRIX mB, int cB, int cS, MATRIX mBBt);
```
`mB[cB][cS]` in: matrix B
`mBBt[cB][cB]` out: matrix containing BB'

```
MATRIX MatBSBt(MATRIX mB, int cB, MATRIX mS,
    int cS, MATRIX mBSBt);
```
`mB[cB][cS]` in: matrix B
`mS[cS][cS]` in: symmetric matrix S or NULL (equivalent to $S = I$)
`mBSBt[cB][cB]` out: matrix containing BSB'

```
MATRIX MatBtSB(MATRIX mB, int cB, MATRIX mS,
    int cS, MATRIX mBtSB);
```
`mB[cB][cS]` in: matrix B
`mS[cB][cB]` in: symmetric matrix S or NULL (equivalent to $S = I$)
`mBtSB[cS][cS]` out: matrix containing $B'SB$

```
MATRIX MatBtB(MATRIX mB, int cB, int cS, MATRIX mBtB);
```
`mB[cB][cS]` in: matrix B
`mBtB[cS][cS]` out: matrix containing $B'B$

```
MATRIX MatBtBVec(MATRIX mB, int cB, int cS, VECTOR vY, MATRIX mBtB);
```
`mB[cB][cS]` in: matrix B
`vY[cS]` in: vector y
`mBtB[cS][cS]` out: matrix containing $(B-y)'(B-y)$

Return value
`MatAB` returns `mAB` $=AB$.
`MatABt` returns `mABt` $=AB'$.
`MatAtB` returns `mAtB` $=A'B$.

MatBBt returns mBBt $=BB'$.
MatBSBt returns mBSBt $=BSB'$.
MatBtSB returns mBtSB $=B'SB$.
MatBtB returns mBtB $=B'B$.
MatBtBVec returns mBtB $=(B-y)'(B-y)$.

MatAlloc, MatAllocBlock

```
MATRIX MatAlloc(int cM, int cN);
MATRIX MatAllocBlock(int cR, int cC);
```

cM, cN	in:	required matrix dimensions

Return value

Returns a pointer to the newly allocated cM $\times$ cN matrix, or NULL if the allocation failed, or if cM was 0. Use MatFree to free the matrix.

Description

MatAlloc(a,b) is the macro version which maps to MatAllocBlock(a,b).

MatCopy...

```
MATRIX MatCopy(MATRIX mDest, MATRIX mSrc, int cM,int cN);
MATRIX MatCopyTranspose(MATRIX mDestT, MATRIX mSrc,
    int cM, int cN);
void   MatCopyVecr(MATRIX mDest, VECTOR vSrc_r, int cM, int cN);
void   MatCopyVecc(MATRIX mDest, VECTOR vSrc_c, int cM, int cN);
```

mSrc[cM][cN]	in:	$m \times n$ matrix A to copy
vSrc_r[cM*cN]	in:	vectorized $m \times n$ matrix (stored by row)
vSrc_c[cM*cN]	in:	vectorized $m \times n$ matrix (stored by column)
mDest[cM][cN]	in:	allocated matrix
	out:	copy of source matrix
mDestT[cN][cM]	in:	allocated matrix
	out:	copy of transpose of mSrc

Return value

MatCopy and MatCopyTranspose return a pointer to the destination matrix which holds a copy of the source matrix.

MatDup

```
MATRIX MatDup(MATRIX mSrc, int cM, int cN);
```

mSrc[cM][cN]	in:	$m \times n$ matrix A to duplicate

Return value

Returns a pointer to a newly allocated matrix, which must be deallocated with MatFree. A return value of NULL indicates allocation failure.

MatFree, MatFreeBlock

```
void MatFree(MATRIX mA, int cM, int cN);
void MatFreeBlock(MATRIX m);
```

mA[cM][cN]	in:	matrix to free, previously allocated using MatAlloc or MatDup

No return value.

Description

`MatFree(m,a,b)` is the macro version which maps to `MatFreeBlock(m)`.

MatGenInvert, MatGenInvert

```
MATRIX MatGenInvert(MATRIX mA, int cM, int cN, MATRIX mRes,
    VECTOR vSval);
MATRIX MatGenInvertSym(MATRIX mAs, int cM, MATRIX mRes, VECTOR vSval);
```

`mA[cM][cN]`	in:	$m \times n$ matrix A to invert
`mAs[cM][cM]`	in:	$m \times m$ symmetric matrix A to invert
`mRes[cN][cM]`	in:	allocated matrix (may be equal to `mA`)
	out:	generalized inverse of A using SVD
`vSval[ min(cM,cN)]`	in:	NULL or allocated vector
	out:	sing.vals of A (if $m \geq n$) or A' (if $m < n$);

Return value

`!NULL`: pointer to mRes indicating success;
`NULL`: failure: not enough memory or couldn't find all singular values.

Description

Uses `IDecSVD` to find the generalized inverse.

MatGetAt

```
double MatGetAt(MATRIX mSrc, int i, int j);
```

`mSrc`	in:	matrix
`i`	in:	row index
`j`	in:	column index

Return value

Returns `mSrc[i][j]`.

MatI

```
MATRIX MatI(MATRIX mDest, int cM);
```

`mDest[cM][cM]`	in:	allocated matrix
	out:	identity matrix

Return value

Returns a pointer to `mDest`.

MatNaN

```
MATRIX MatNaN(MATRIX mDest, int cM, int cN);
```

`mDest[cM][cN]`	in:	allocated matrix
	out:	matrix filled with the NaN value (Not a Number)

Return value

Returns a pointer to `mDest`.

MatPartAcf

```
MATRIX MatPartAcf(MATRIX mPartAcf, MATRIX mAcf, int cAcf, MATRIX mY,
    int cY, double *pdLogDet, BOOL bFilter)
```

`mPartAcf`	in:	matrix: `mPartAcf[cAcf][1+cY]`
	out:	partial autocorrelation function in first column (`mY` is NULL; first value will be 1), or residuals from filter applied to `mY` (then last column holds variances)
`mAcf[cAcf][1]`	in:	autocovariance function, only first column is used
`mY[cAcf][cY]`	in:	NULL, or data columns to apply filter or smoother to
`pdLogDet`	in:	NULL, or pointer to double
	out:	determinant of filter
`bFilter`	in:	`TRUE`: apply filter to, else smoother

Return value

Returns `mPartAcf` if successful, NULL if not enough observations.

MatRan, MatRann

```
MATRIX MatRan(MATRIX mA, int cR, int cC);
MATRIX MatRann(MATRIX mA, int cR, int cC);
```

`mA[cR][cC]`	in:	allocated matrix
	out:	filled with random numbers

Return value

Both functions return `mA`

`MatRan` generates uniform random numbers, `MatRann` standard normals.

MatReflect, MatTranspose

```
MATRIX MatReflect(MATRIX mA, int cA);
MATRIX MatTranspose(MATRIX mA, int cA);
```

`mA[cA][cA]`	in:	matrix
	out:	transposed matrix.

Return value

Both return a pointer to `mA`.

Description

MatTranspose transposes a square matrix. MatReflect reflects a square matrix around its secondary diagonal.

MatSetAt

```
void  MatSetAt(MATRIX mDest, double d, int i, int j);
```

`mDest`	in:	matrix to change
	out:	changed: `mDest[i][j] = d`
`d`	in:	value
`i`	in:	row index
`j`	in:	column index

No return value.

MatStandardize

```
MATRIX MatStandardize(MATRIX mXdest, MATRIX mX, int cT, int cX);
```

`mXdest[cT][cX]`	out:	standardized mX matrix
`mX[cT][cX]`	in:	data which to standardize
`cT`	in:	number of observations

Return value

Returns mXdest if successful, NULL if not enough observations.

MatVariance

```
MATRIX MatVariance(MATRIX mXtX, MATRIX mX, int cT, int cX,
    BOOL fCorr);
```

`mXtX[cX][cX]`	out:	variance matrix (fCorr is FALSE) or correlation matrix (fCorr is TRUE)
`mX[cT][cX]`	in:	variable of which to compute correlogram
`cT`	in:	number of observations

Return value

Returns mXtX if successful, NULL if not enough observations.

MatZero

```
MATRIX MatZero(MATRIX mDest, int cM, int cN);
```

`MatZero[cM][cN]`	in:	allocated matrix
	out:	matrix of zeros

Return value

Returns a pointer to `mDest`.

RanDirichlet

```
void   RanDirichlet(VECTOR vX, VECTOR vAlpha, int cAlpha);
```

`vX[cAlpha - 1]`	out:	random values
`vAlpha[cAlpha]`	in:	shape parameters

RanGetSeed

```
int    RanGetSeed(int *piSeed, int cSeed);
```

`piSeed`	in:	`NULL` (only returns the seed count), or array with `cSeed` integer elements
`piSeed`	out:	current seeds

Return value

Returns the number of seeds used in the current generator, or -1 if the default random number generator has not been initialized.

RanInit

```
void  RanInit(struct ranState *pRan)
```

No return value.

`RanInit(NULL)` must be called to initialize the default random number generator (it does nothing if the default has already been initialized).

RanNewRan, RanSetRan

```
void   RanNewRan(DRANFUN fnDRanu,
    RANSETSEEDFUN fnRanSetSeed, RANGETSEEDFUN fnRanGetSeed);
void   RanSetRan(const char *sRan);
```

sRan	in:	string, as in Ox function ranseed
fnDRanu	in:	pointer to new random number generator (same syntax as DRanU)
fnRanSetSeed	in:	pointer to new set seed function (same syntax as RanSetSeed)
fnRanGetSeed	in:	pointer to new get seed function (same syntax as RanSetGeed)

Description

RanSetRan chooses one of the built-in generators. RanNewRan installs a new generator.

RanSetSeed

```
void   RanSetSeed(int *piSeed, int cSeed);
```

piSeed	in:	NULL (means a reset to initial seed), or array with cSeed new seeds (which may not be 0)

Description

Sets the seeds for the current random number generator.

RanUorder, RanSubSample, RanWishart

```
void  RanUorder(VECTOR vU, int cU);
void  RanSubSample(VECTOR vU, int cU, int cN);
void  RanWishart(MATRIX mX, int cX, int cT);
```

vU[cU]	out: random values
mX[cX][cX]	out: random values, Wishart(cT, $\mathbf{I}_{\texttt{cX}}$)

SetFastMath

```
void   SetFastMath(BOOL fYes);
```

fYes	in:	TRUE: switches *Fastmath* mode on, else switches it off

Description

When *FastMath* is active, memory is used to optimize some matrix operations. *FastMath* mode uses memory to achieve the speed improvements. The following function are *FastMath* enhanced: MatBtB, MatBtBVec

SetInvertEps

```
void   SetInvertEps(double dEps);
```

dEps	in:	sets inversion epsilon ϵ_{inv} to dEps if dEps ≥ 0, else to the default.

Description

The following functions return singular status if the pivoting element is less than or equal to ϵ_{inv}: ILDLdec, ILUPdec, ILDLbandDec. Less than $10\epsilon_{inv}$ is used by IOlsQR.

A singular value is considered zero when less than $||A||_\infty 10\epsilon_{inv}$ in MatGenInvert. The default value for ϵ_{inv} is $1000 \times$ DBL_EPSILON.

SetInf, SetNaN

```
void SetInf(double *pd);
void SetNaN(double *pd);
```

`*pd`	out:	set value

Description

Sets the argument to infinity (`.Inf`) or not-a-number (`.NaN`).

SortVec, SortMatCol, SortmXtByVec, SortmXByCol

```
int SortVec(VECTOR vX, int cT);
int SortMatCol(MATRIX mX, int iCol, int cT);
int SortmXtByVec(int cT, VECTOR vBy, MATRIX mXt, int cX);
int SortmXByCol(int iCol, MATRIX mX, int cT, int cX);
```

`vX[cT]`	in:	vector
	out:	sorted vector
`mX[cT][.]`	in:	matrix
	out:	matrix with column `iCol` sorted (`SortMatCol`)
`mX[cT][cX]`	in:	matrix
	out:	matrix with columns sorted according to column `iCol` (`SortmXByCol`)
`mXt[cX][cT]`	in:	matrix
	out:	matrix with rows sorted according to vector `vBy[cT]` (`SortmXtByVec`)

Description

Sorting functions (.NaNs are pushed to the beginning).

ToeplitzSolve

```
void ToeplitzSolve(VECTOR vR, int cR, int cM, MATRIX mB,
    int cB, VECTOR v_1);
```

`vR[cR]`	in:	vector specifying Toeplitz matrix
`cM`	in:	dimension of Toeplitz matrix, `cM` $\geq$ `cR`, remainder of `vR` is assumed zero.
`mB[cM][cB]`	in:	`cM` $\times$ `cB` rhs of system to be solved
	out:	contains X, the solution to $AX = B$
`v_1[cM]`	in:	work vector
	out:	changed, `v_1[0]` is the logarithm of the determinant

Return value

0: success; 1: singular matrix or `v_1` is NULL.

Description

Solves $AX = B$ when A is symmetric Toeplitz.

VecAlloc, VecDup, VecFree

```
VECTOR VecAlloc(int cM);
VECTOR VecDup(VECTOR vSrc, int cM);
void VecFree(VECTOR vX);
```

`cM`	in:	required size of vector
`vSrc[cM]`	in:	m vector to duplicate
`vX`	in:	m vector to free

Return value

`VecAlloc` gives a pointer to the newly allocated vector, or `NULL` if the allocation failed, or if `cM` was 0.

`VecDup` gives a pointer to the newly allocated destination vector, which holds a copy of the source vector. A return value of `NULL` indicates allocation failure.

Description

These are implemented as macros, and memory is allocated and freed in the caller's process.

A vector allocated with `VecAlloc` may be freed by using the standard C function `free` or using the macro `VecFree`.

VecAllocBlock, VecDupBlock, VecFreeBlock

```
VECTOR VecAllocBlock(int cM);
VECTOR VecDupBlock(VECTOR vSrc, int cM);
void VecFreeBlock(VECTOR vX);
```

`cM`	in:	required size of vector
`vSrc[cM]`	in:	m vector to duplicate
`vX`	in:	m vector to free

Return value

`VecAllocBlock` returns a pointer to the newly allocated vector, or `NULL` if the allocation failed, or if `cM` was 0.

`VecDupBlock` returns a pointer to the newly allocated destination vector, which holds a copy of the source vector. A return value of `NULL` indicates allocation failure.

Description

These are implemented as functions, and memory is allocated and freed in the Ox DLL.

A vector allocated with `VecAllocBlock` must be freed by using `VecFreeBlock`.

VecCopy

```
VECTOR VecCopy(VECTOR vDest, VECTOR vSrc, int cX)
```

`vDest[cM]`	in:	m vector: destination for copy
`vSrc[cM]`	in:	m vector to copy

Return value

Return a pointer to `vDest`.

VecrCopyMat, VeccCopyMat

```
void VecrCopyMat(VECTOR vDest_r, MATRIX mSrc, int cM, int cN);
void VeccCopyMat(VECTOR vDest_c, MATRIX mSrc, int cM, int cN);
```

`vDest_r[cM*cN]`	in:	allocated vector
	out:	vectorized $m \times n$ matrix (stored by row)
`vDest_c[cM*cN]`	in:	allocated vector
	out:	vectorized $m \times n$ matrix (stored by column)
`mSrc[cM][cN]`	in:	$m \times n$ source matrix

No return value.

VecDiscretize

```
VECTOR VecDiscretize(VECTOR vY, int cY, double dMin, double dMax,
    VECTOR vDisc, int cM, VECTOR vT, int iOption);
```

`vY[cY]`	in:	T vector to discretize
`dMin`	in:	first point
`dMax`	in:	last point, if `dMin == dMax`, the data minimum and maximum will be used
`vDisc[cM]`	in:	m vector
	out:	discretized data
`vT[cY]`	in:	`NULL` or T vector
	out:	if `!NULL`: points (x-axis)

Return value

Return a pointer to `vDisc`, which holds the discretized data.

VecTranspose

```
VECTOR VecTranspose(VECTOR vA, int cM, int cN);
```

`vA[cM * cN]`	in:	$M \times N$ matrix stored as vector
	out:	$N \times M$ transposed matrix.

Return value

Returns a pointer to `vA`.

Description

VecTranspose transposes a matrix which is stored as a column.

Subject Index