FOSSEE Optimization Toolbox

Developer's Manual

toolbox@scilab.in

August 7, 2019

Contents

1

Introduction

FOSSEE Optimization toolbox is a toolbox in Scilab maintained and developed by FOSSEE[1](Free and Open Source Software in Education), IIT Bombay[2]. It can solve the following optimization problems :

1. Linear programming(LP)
2. Quadratic programming(QP)
3. Nonlinear programming(NLP)
4. Integer programming(IP)
5. Second order Conic Programming(SOCP) problems

It also solves specific optimization problems like least squares, minimax and goal attainment problem.
Scilab is a open-source numerical computational package licensed under GPLv2 license which has broad applications in educational and engineering domains. It is a competitive alternative to Matlab and Octave. Scilab was initially maintained and developed by INRIA[3](French Institute for Research in Computer Science and Automation). Scilab consortium was formed in June 2010 which now handles Scilab development. FOSSEE Optimization toolbox uses a dozen of open-source optimization solvers to solve the optimization problems. Many of these solvers are part of the COIN-OR[4] initiative which promotes the development and use of open-source softwares for operations research community. Scilab provides API functions to call libraries from C, C++ and FORTRAN. Most of the solvers used by the toolbox is programmed in C++.

1.1 Scilab

Scilab, as mentioned above is a numerical computational software. It can be downloaded from the Scilab website[5] or from the respective repositories in

[1]https://fossee.in/
[2]http://iitb.ac.in/
[3]https://www.inria.fr/en/
[4]https://www.coin-or.org/
[5]http://www.scilab.org/download/latest

case of Linux or macOS. The standard IDE of Scilab is given in figure **??**.It contains a file browser,variable browser, command history and the console. Scilab also provides a command line interface which can be run by `scilab-cli` from the terminal.
Native scilab provides only limited functions. For advanced and specific functions, the user has to download toolboxes which are scilab packages. The scilab package manager is called ATOMS(AuTomatic mOdules Management for Scilab).The available scilab toolboxes on ATOMS can either accessed through the ATOMS website[6] or by clicking Applications»Module manager in the Scilab menu bar.

NOTE
It is good practise to run `atomsSystemUpdate()` in scilab console to update ATOMS upon fresh install. In case, you are using Scilab 5.5, execute `atomsRepositoryAdd('http://atoms.scilab.org')` instead of the previous command

1.2 Optimization Libraries

FOSSEE Optimization toolbox mainly uses six mathematical optimization libraries, namely :

1. CLP[7] (Coin-or Linear Programming)
2. Ipopt[8] (Interior Point OPTimizer)
3. Symphony[9]
4. Bonmin[10] (Basic Open-source Nonlinear Mixed INteger programming)
5. CBC[11](Coin-or branch and cut)
6. ECOS[12]

These libraries, in turn are dependent other libraries such as:

1. CGL[13] (Cut Generation Library)
2. LAPACK[14](Linear Algebra PACKage)
3. BLAS[15] (Basic Linear Algebra Subprograms)

[6]https://atoms.scilab.org/
[7]https://projects.coin-or.org/Clp
[8]https://projects.coin-or.org/Ipopt
[9]https://projects.coin-or.org/SYMPHONY
[10]https://www.coin-or.org/Bonmin/
[11]https://projects.coin-or.org/Cbc
[12]https://www.embotech.com/ECOS/
[13]https://projects.coin-or.org/Cgl
[14]https:/www.netlib.org/lapack/
[15]https:/www.netlib.org/blas/

4. MUMPS[16] (MUltifrontal Massively Parallel Sparse direct Solver)

5. OSI[17] (Open Solver Interface)

NOTE
The source codes for all the COIN-OR libraries can be found at coin-or download page[a] or on github[b]

[a]https://www.coin-or.org/download/source/
[b]https://github.com/coin-or

1.3 Downloading the toolbox

FOT can be downloaded onto your system in the following ways:

1. From ATOMS : by executing `atomsInstall('FOT')` on Scilab prompt followed by restarting Scilab.

2. From github : clone from the github repository, `git clone https://github.com/FOSSEE/FOSSEE-Optimization-toolbox.git`

We prefer the latter method and the following instructions are with respect to this mode. You can load the toolbox in Scilab by going to the FOT root directory and running `exec builder.sce` followed by `exec loader.sce` on the Scilab console. These commands compiles and loads the files onto Scilab memory. In case, the toolbox is already compiled, only the latter command has to be run `exec cleaner.sce` will delete all the compiled binary files.

1.4 Prerequisites

We expect you to have a novice knowledge of both Scilab and C++. You should be familiar with Object Oriented Programming concepts and comfortable with pointers in C++. In addition to this, you should have good command over Linux and familiar with its workflows.

1.5 Purpose of document

This manual is made to help developers and enthusiasts get beyond the initial barrier and contribute productively to the development and maintenance of FOSSEE Optimization toolbox. This point is highly relevant for this toolbox because of its multidisciplinary domain.

[16]https:/mumps.enseeiht.fr/

[17]https://projects.coin-or.org/Osi

This Developer's Manual consists of 8 chapters. chapter 2 deals with structure of the toolbox and describes them briefly.It is followed by chapter 3 which explains in detail how the toolbox is built and loaded onto Scilab. chapter 4 describes the start and quit files which are run at the initialization and finalization of the toolbox. chapter 5 is about Scilab function files and explains it by taking one of function files. After that, chapter 6 is on the C++ files that is used to call the solver libraries and is followed by chapter 7 which consists of detailed description of downloading and building each of the libraries needed for the Toolbox. chapter 8 guides on the help documentation and its related aspects. The Appendix of this manual is divided into three different chapters. Appendix A displays some codes which are referred to in the some of the chapters. Appendix B gives some tutorials for practise and is followed by Appendix C which are assignments which gives you first hand experience in toolbox development.

2

Toolbox Structure

Scilab toolboxes are important for it's development. Scilab, being a numerical computational package caters to a vast domain. This being said, a person working with Image processing may not need the functions related fuzzy logic. Hence, releasing these specialized functions as modules or toolboxes not only serves as the aforementioned purpose, but also helps in reducing the size of the basic Scilab installation.

Toolboxes are developed when there are a number of functions to be provided in the particular domain. If the number of functions are very few, the developer can use `ilib_build` function to build the required files rather than create a toolbox. For more details about `ilib_build`, check the help page.

To be versatile in usage, Scilab has a particular folder structure for its toolbox. A detailed article on the Scilab toolboxes written by the Scilab team can be found here[1]. It is recommended for a new developer to go through that article. Advanced toolbox development can be found in this document written by Michaël Baudin [2].

FOSSEE Optimization toolbox also follows a similar folder structure. We will be discussing about these folders in the coming chapters. A tree of the current version of FOT is given below.

Some of the important directories are :

1. macros : Contains the scilab function files containing the functions that are to be called form Scilab and their respective builder files.
2. sci_gateway : Contains gateway files which are to be called from the thirdparty directory
3. thirdparty : Contains the header files and dynamic linking libraries(shared libraries) of the toolbox classified by OS and architecture
4. etc : Contains the files needed at the time of building the toolbox

[1] https://wiki.scilab.org/howto/Create%20a%20toolbox

[2] https://forge.scilab.org/index.php/p/docsciextensions/downloads/547/

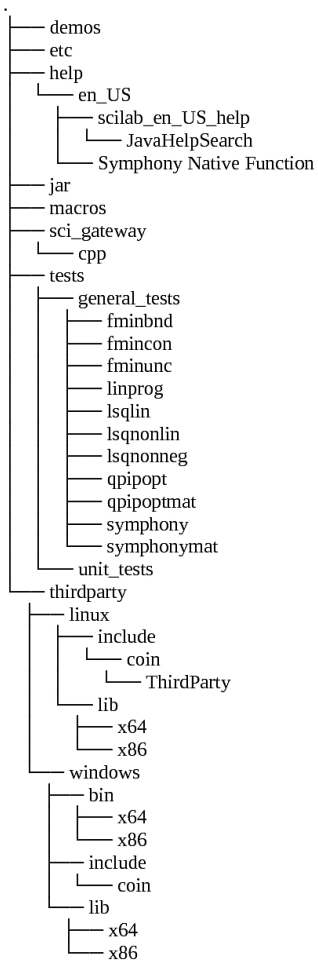

Figure 2.1: FOT folder structure tree

5. help : Contains the required help files for the toolbox
6. tests : Contains the test files for the toolbox
7. demo : Contains demo files for the toolbox

3

Builder files

3.1 Introduction

A builder file is a file used to build the toolbox. It creates binaries of the toolbox that is loaded into Scilab memory. Most of the directories mentioned in the previous chapter has their own builder files. The general file name of builder files are builder*.sce where * is replaced by a string.

3.2 builder.sce

The following is the main FOT builder.sce file which is located at the root directory:

```
lines(0);
try
 getversion('scilab');
catch
 error(gettext('Scilab 5.0 or more is required.'));
end;
//
    ============================================================================

if ~with_module("development_tools") then
  error(msprintf(gettext("%s module not installed."),"
      development_tools"));
end
//
    ============================================================================

TOOLBOX_NAME = "FOSSEE_Optimization_Toolbox";
TOOLBOX_TITLE = "FOSSEE Optimization Toolbox";
//
    ============================================================================

toolbox_dir = get_absolute_file_path("builder.sce");
tbx_builder_macros(toolbox_dir);
```

```
tbx_builder_gateway(toolbox_dir);
tbx_builder_help(toolbox_dir);
tbx_build_loader(TOOLBOX_NAME, toolbox_dir);
tbx_build_cleaner(TOOLBOX_NAME, toolbox_dir);
clear toolbox_dir TOOLBOX_NAME TOOLBOX_TITLE;
```

`lines(0)` disables vertical paging of the display. The try-catch condition exits the builder if the required Scilab version is not available. The if loop following that checks if the development_tool module is present. Generally, it should be present with your default Scilab installation. The following lines assign name and title, gets the path of the builder file and executes the other builder files which compiles the required directories with respective commands. It also builds a loader.sce which is used to load the binary files into Scilab memory and a cleaner.sce file to remove binary files.

3.3 buildmacros.sce

The buildmacros.sce file is located in macros directory. It compiles the scilab macro files which are scilab function files. The following is the content of the buildmacros.sce file:

```
tbx_build_macros("FOSSEE_Optimization_Toolbox",
                     get_absolute_file_path("buildmacros.sce"))
                         ;
clear tbx_build_macros;
```

`tbx_build_macros` is the scilab function used to compile the macros file. The first argument is the toolbox name which is set in the main builder file and the second input is the path of the macros builder file. `clear tbx_build_macros` clears the tbx_build_macros function from the scilab memory.

3.4 builder_gateway.sce

builder_gateway.sce file is used to run the compiles the builder files which are located in sub-directories.

```
sci_gateway_dir = get_absolute_file_path('builder_gateway.sce')
    ;

tbx_builder_gateway_lang('cpp', sci_gateway_dir);
tbx_build_gateway_loader(['cpp'], sci_gateway_dir);

clear tbx_builder_gateway_lang tbx_build_gateway_loader;
clear sci_gateway_dir;
```

`sci_gateway_dir` stores the path to the this file. `tbx_builder_gateway_lang` runs the builder file in the respective sub-directories which in turn compiles the scripts. The first argument for this function is an array of languages which are to be compiled and the second argument is the path the

builder_gateway.sce file. The sub-directories have to be named in the name of the languages that the toolbox uses in these files. Scilab accepts C, C++ and FORTRAN as API languages.
`tbx_build_gateway_loader(['cpp'], sci_gateway_dir);` generates the loader_gateway script. The inputs are the same as `tbx_builder_gateway_lang`.

3.4.1 builder_gateway_cpp.sce

`builder_gateway_cpp.sce` file in `root_dir/sci_gateway/cpp/` is the builder file for C++ files. The explanation for this file can be found in Section 4.6.4 of Michaël Baudin's article mentioned in chapter 2.

3.5 builder_help.sce

builder_help.sce file is used to compile the help files which are located in directories, the names of which are the locales. locales are parameters which refers to the user's languages. The FOT uses en_US locale and hence only has one sub-directory in the help directory.

```
mode(-1)
lines(0)

toolbox_title = "FOSSEE_Optimization_Toolbox"

help_dir = get_absolute_file_path('builder_help.sce');

tbx_builder_help_lang("en_US", help_dir);

clear toolbox_title;
```

`mode(-1)` executes the builder_help.sce file silently. `tbx_builder_help_lang` is used to build the help files which are located in the en_US directory.

4

etc directory

4.1 Introduction

etc directory contains the initialization and finalization script of the toolbox which are run at the beginning and termination of the toolbox. They are executed while executing the loader and unloader files.

4.2 FOSSEE_Optimization_Toolbox.start

The name of the initialization script for a toolbox is the name of the toolbox followed by ".start". In our case, the name of the file is "FOSSEE_Optimization_Toolbox.start". The code for the same is given in Appendix section A.1.

The FOSSEE_Optimization_Toolbox.start file is executed when we run the loader. It's purpose includes :

1. Load function libraries from macros directory.
2. Load gateway and shared libraries form sci_gateway and thirdparty directory.
3. Load help from help directory.
4. Load demos from demos directory.

4.3 FOSSEE_Optimization_Toolbox.quit

The name of the finalization script for a toolbox is the name of the toolbox followed by ".quit". In our case, the name of the file is "FOSSEE_Optimization_Toolbox.quit". The code for the same is given in section A.2.

The FOSSEE_Optimization_Toolbox.quit file is executed when we run the loader. It's purpose includes :

1. Unlink the toolbox libraries.

2. Remove any preferences that was set by the toolbox.

5

macros directory

5.1 Introduction

Macros folder contains scilab function files(*.sci). Files with extensions other than sci will not be compiled when the builder is run. Scilab macros can be:

1. A Scilab function file which returns the result after computation.
2. A Scilab function which calls a C, C++ or FORTRAN code.
3. A Scilab function which calls a binary library.

In FOT, the macros files are of second kind which calls a C++ file. They should have the same file name as that of function inside. Except the licence information and some information's in comments ,the whole code and the help texts in comments are inside the function.

5.2 Outline of a macros file

The general outline of most of macros files in FOT is as follows:

1. Help page comments
2. Input retrieval
3. Error checks
4. Input modifications
5. Call to the C++ library
6. Output retrieval,checks and modifications

Lets take each of these steps and go through `ROOT_DIR/macros/fmincon.sci` file to analyze it. The function is declared with `function [xopt,fopt,exitflag, output,lambda,gradient,hessian] = fmincon (varargin)` where `varargin` helps facilitates the input argument which can be a variable. It is a list with the input variables.

5.2.1 Commented Help page

The function declaration is immediately followed by the help documentation comments which is explained in detail in chapter 8.

5.2.2 Input retrieval

The inputs from the Scilab execution is parsed here using `varargin` function.

```
[lhs , rhs] = argn();

//To check the number of arguments given by the user
if ( rhs<4 | rhs>10 ) then
   errmsg = msprintf(gettext("%s: Unexpected number of input
       arguments : %d provided while it should be 4,6,8,9,10"),
        "fmincon", rhs);
   error(errmsg)
end

if (rhs==5 | rhs==7) then
errmsg = msprintf(gettext("%s: Unexpected number of input
    arguments : %d provided while it should be 4,6,8,9,10s"), "
    fmincon", rhs);
error(errmsg)
end

//Storing the Input Parameters
fun        = varargin(1);
x0     = varargin(2);
A      = varargin(3);
b      = varargin(4);
Aeq    = [];
beq    = [];
lb       = [];
ub       = [];
nlc      = [];

if (rhs>4) then
   Aeq    = varargin(5);
   beq    = varargin(6);
end

if (rhs>6) then
   lb       = varargin(7);
   ub       = varargin(8);
end
```

```
if (rhs>8) then
   nlc       = varargin(9);
end
```

`argn` function returns the number of lhs and rhs arguments in the call. The subsequent `if` conditions eliminate the possibility of unplanned inputs. `varargin` being a list is parsed to extract the respective inputs.

5.2.3 Error checks

The code for Error checks is not shown here. It starts from the end of Input retrieval and goes on till the declaration of `fGrad1` function. These error checks can be :

1. Checking the type of the variable passed. The `Checktype.sci` file in the macros directory assists in this process.
2. Dimension checks in case of a real number,lists or Matrix.
3. Dependency checks. For example, if input A is empty, then input b should also be empty and vice versa.
4. Initialization checks in case of a function to check if the current value is feasible for the function.
5. Double checks, to affirm the credibility of user provided values.
6. Content check, to check if the content of a variable is valid. For example, lower bound,lb cannot take infinity.
7. Input modification checks, if the given input has to be modified to get the actual input. For example, if an input taking row vector is given a column vector.

5.2.4 Input modifications

The input modification for `fmincon.sci` includes the functions `fGrad1`,`lHess1` and `addcGrad1`. Some of the inputs are also modified in the above error checking process. In these modifications, the inputs are modified to suit to the specifications of the solver libraries.

5.2.5 Call to the C++ library

Following is the code to call the C++ library.

```
//Creating a Dummy Variable for IPopt use
empty=[0];

//Calling the Ipopt function for solving the above problem
```

```
[xopt,fopt,status,iter,cpu,obj_eval,dual,lambda1,zl,zu,gradient
    ,hessian1] = solveminconp(f,A,b,Aeq,beq,lb,ub,no_nlc,
    no_nlic,addnlc1,fGrad1,lHess1,addcGrad1,x0,options,empty)
```

The empty dummy variable is provided to aid in retrieving functions in using `scilab_call` API function of Scilab. We will discuss about it in the coming sections.

5.2.6 Output retrieval,checks and modifications

The subsequent lines after calling the solver library are to manage output and pass it back to Scilab which are self-explanatory.

6

sci_gateway files

6.1 Introduction

The gateway files as the name suggests, acts as a gateway between Scilab and C++. The inputs from Scilab are not compatible with other languages and hence Scilab provides an array of API functions to accomplish this. sci_gateway files generally are used to get input form Scilab ,pass it to the respective library, retrieve the results and pass the results back to Scilab.

6.2 Basic Scilab API Functions

The Scilab 6 API to pass and return values from C and C++ has been a major improvement over the previous versions of Scilab. In the context of the FOSSEE Optimization Toolbox, it reduces the dependence on the sci_iofunc file.

The main purpose of sci_ iofunc file was to management input and output for FOT. It made the main code less cumbersome. However, since the advent of the new API, a lot of the advantages offered by sci_iofunc are rendered moot.

A list of API functions provided by Scilab can be found at this link[1].These API functions are used in the interfaces directly, instead of relying on `root_dir/sci_gateway/cpp/sci_iofunc.cpp` as we did in previous versions.

6.3 Outline of a sci_gateway file

The general outline of a sci_gateway file are as follows :

1. Variable initialization
2. Input retrieval

[1] https://help.scilab.org/docs/6.0.2/en_US/api_scilab.html

3. Input modifications, if any
4. Calling the library
5. Output management
6. Returning output to Scilab

It is categorized so to help developers get an overall view of the functions. Many of the files don't have clear cut difference between two of the above said outline. We will go through this section with the help of `root_dir/sci_gateway/cpp/cpp_intfmincon.cpp` file.

6.3.1 Variable initialization

Variable declarations are not just restricted to variables, but namespace declarations as well.

```
using namespace Ipopt;

if (nin !=13)  //Checking the input arguments
{
   Scierror(999, "%s: Wrong number of input arguments: %d
       expected.\n", fname, 13);
          return STATUS_ERROR;
}

if (nout !=3) //Checking the output arguments

{
   Scierror(999, "%s: Wrong number of output argument(s): %d
       expected.\n", fname, 3);
      return 1;
}



//Function pointers, input matrix(Starting point) pointer, flag
    variable
double *x0ptr=NULL, *lbptr=NULL, *ubptr=NULL,*Aptr=NULL, *bptr=
   NULL, *Aeqptr=NULL, *beqptr=NULL;
double nonlinCon=0,nonlinIneqCon=0;


   // Input arguments
double *cpu_time=NULL,*max_iter=NULL;
static unsigned int nVars = 0,nCons = 0;
unsigned int temp1 = 0,temp2 = 0, iret = 0;
int x0_rows=0, x0_cols=0, lb_rows=0, lb_cols=0, ub_rows=0,
   ub_cols=0, A_rows=0, A_cols=0, b_rows=0, b_cols=0, Aeq_rows
   =0, Aeq_cols=0, beq_rows=0, beq_cols=0;

// Output arguments
```

```
double  ObjVal=0,iteration=0,cpuTime=0,fobj_eval=0;
double dual_inf, constr_viol, complementarity, kkt_error;
const double *fX = NULL, *fGrad =  NULL;
const double *fHess =  NULL;
const double *fLambda = NULL;
const double *fZl=NULL;
const double *fZu=NULL;
int rstatus = 0;
int int_fobj_eval, int_constr_eval, int_fobj_grad_eval,
    int_constr_jac_eval, int_hess_eval;
```

`nin` and `nout` are Scilab API variables which check the inputs and outputs of the Scilab call.

6.3.2 Input retrieval

The following lines code below of `sci_ipoptfmincon.cpp` retrieves the input from Scilab into C++.

```
if (scilab_isDouble(env, in[5]) == 0 || scilab_isMatrix2d(env,
    in[5]) == 0)
   {
      Scierror(999, "%s: Wrong type for input argument #%d: A
          double matrix expected.\n", fname, 6);
      return 1;
   }

   scilab_getDoubleArray(env, in[5], &x0);
   int size1 = scilab_getDim2d(env, in[5], &x0_rows, &x0_cols);


   if (scilab_isDouble(env, in[6]) == 0 || scilab_isMatrix2d(
       env, in[6]) == 0)
   {
      Scierror(999, "%s: Wrong type for input argument #%d: A
          double matrix expected.\n", fname, 7);
      return 1;
   }

   scilab_getDoubleArray(env, in[6], &lb);



   if (scilab_isDouble(env, in[7]) == 0 || scilab_isMatrix2d(
       env, in[7]) == 0)
   {
      Scierror(999, "%s: Wrong type for input argument #%d: A
          double matrix expected.\n", fname, 8);
      return 1;
   }

   scilab_getDoubleArray(env, in[7], &ub);
```

```
if (scilab_isDouble(env, in[8]) == 0 || scilab_isMatrix2d(
    env, in[8]) == 0)
{
   Scierror(999, "%s: Wrong type for input argument #%d: A
       double matrix expected.\n", fname, 9);
   return 1;
}

scilab_getDoubleArray(env, in[8], &conLb);
size1 = scilab_getDim2d(env, in[8], &nCons, &nCons2);



if (scilab_isDouble(env, in[9]) == 0 || scilab_isMatrix2d(
    env, in[9]) == 0)
{
   Scierror(999, "%s: Wrong type for input argument #%d: A
       double matrix expected.\n", fname, 10);
   return 1;
}

scilab_getDoubleArray(env, in[9], &conUb);


// Getting intcon
if (scilab_isDouble(env, in[10]) == 0 || scilab_isMatrix2d(
    env, in[10]) == 0)
{
   Scierror(999, "%s: Wrong type for input argument #%d: A
       double matrix expected.\n", fname, 11);
   return 1;
}

scilab_getDoubleArray(env, in[10], &intcon);
size1 = scilab_getDim2d(env, in[10], &intconSize, &
    intconSize2);



if (scilab_isDouble(env, in[12]) == 0 || scilab_isMatrix2d(
    env, in[12]) == 0)
{
   Scierror(999, "%s: Wrong type for input argument #%d: A
       double matrix expected.\n", fname, 13);
   return 1;
}

scilab_getDoubleArray(env, in[12], &LC);



 //Initialization of parameters
```

```
//Getting parameters
if (scilab_isList(env, in[11]) == 0)
 {
     Scierror(999, "%s: Wrong type for input argument #%d: A
         list expected.\n", fname, 12);
     return 1;
 }

scilabVar temp1 = scilab_getListItem( env, in[11], 1);
scilabVar temp2 = scilab_getListItem( env, in[11], 3);
scilabVar temp3 = scilab_getListItem( env, in[11], 5);
scilabVar temp4 = scilab_getListItem( env, in[11], 7);
scilabVar temp5 = scilab_getListItem( env, in[11], 9);

double integertolerance=0, allowable_gap=0, maxnodes =0,
    cpuTime=0, maxiter=0;

scilab_getDouble(env, temp1, &integertolerance);
scilab_getDouble(env, temp2, &maxnodes);
scilab_getDouble(env, temp3, &cpuTime);
scilab_getDouble(env, temp4, &allowable_gap);
scilab_getDouble(env, temp5, &maxiter);


int max_nodes = (int)maxnodes;
int cpu_time = (int)cpuTime;
int iterLim = (int)maxiter;
```

Each of `if` conditions processes one of inputs from Scilab and saves the address to a pointer which was declared earlier. `getFixedSizeDoubleMatrixInList` is used to retrieve the options which are passed as a list.

scilab_call function

`scilab_call` is a C++ API function for Scilab which helps in evaluating a Scilab function at the given input. It is of particular interest because of the way it interacts with the Scilab call. We will see about this function with an example from `root_dir/sci_gateway/cpp/sci_minconNLP.cpp`. Before we move ahead, give a thought to the inputs of the API call :

```
[xopt,fopt,status,iter,cpu,obj_eval,dual,lambda1,zl,zu,gradient
    ,hessian1] = solveminconp(f,A,b,Aeq,beq,lb,ub,no_nlc,
    no_nlic,addnlc1,fGrad1,lHess1,addcGrad1,x0,options,empty)
```

Notice that the first input `f` is a function, and so are the 10^{th},11^{th},12^{th} and 13^{th} input which are `addnlc1`, `fGrad1`, `lHess1` and `addcGrad1` respectively. There last three real inputs in the end, i.e. `x0`,`options` and `empty`,the dummy variable.

Now we return to the `minconNLP::eval_h` function in `root_dir/sci_gateway/cpp/sci_minconNLP.cpp`. If the values are null, the hessian, which is represented as a sparse matrix is given random values. Else, the hessian matrix from is taken in from Scilab using the following code :

```
double check;


const Number *xNew=x;

const Number *lambdaNew=lambda;
double objfac=obj_factor;

scilabVar* funcIn = (scilabVar*)malloc(sizeof(scilabVar) * (
    numVars_) * 1);
funcIn[0] = scilab_createDoubleMatrix2d(env_, 1, numVars_, 0);
scilab_setDoubleArray(env_, funcIn[0], x);
double t= 2;
funcIn[1] = scilab_createDouble(env_, objfac);
funcIn[2] = scilab_createDoubleMatrix2d(env_, 1, numConstr_, 0)
    ;

scilab_setDoubleArray(env_, funcIn[2], lambdaNew);

scilab_call(env_, L"lHess1", 3, funcIn, 2, out);


double* resCh;

if (scilab_isDouble(env_, out[1]) == 0 || scilab_isScalar(env_,
     out[1]) == 0)
{
   Scierror(999, "Wrong type for input argument #%d: An int
       expected.\n", 2);
          return 1;
}


scilab_getDouble(env_, out[1], &check);
if (check==1)
{
   return true;
}
else
{
   if (scilab_isDouble(env_, out[0]) == 0 || scilab_isMatrix2d(
       env_, out[0]) == 0)
   {
      Scierror(999, "Wrong type for input argument #%d: An int
          expected.\n", 2);
             return 1;
   }

   scilab_getDoubleArray(env_, out[0], &resCh);

   Index index=0;
   for (Index row=0;row < numVars_ ;++row)
   {
      for (Index col=0; col < numVars_; ++col)
```

```
        {
            values[index++]=resCh[numVars_*row+col];
        }
    }
}

Index index=0;
for (Index row=0;row < numVars_ ;++row)
{
    for (Index col=0; col <= row; ++col)
    {
        finalHessian_[n*row+col]=values[index++];
    }
}

index=0;
for (Index col=0;col < numVars_ ;++col)
{
    for (Index row=0; row <= col; ++row)
    {
        finalHessian_[n*row+col]=values[index++];
    }
}

}
```

`scilab_call` function sends back the evaluated value of the Scilab function at input. In order to use `scilab_call`, we have to obtain the the inputs using Scilab API writing function like `scilab_getDouble`. Then `scilab_call` function is called and the outputs from Scilab functions are overwritten over the first position of input that you gave earlier.

In this case, the lHess1 function in `fmincon.sci` has three inputs and two outputs. For the three inputs, the `get*` functions are used, with their respective arguments. The dummy variable was exclusively declared for this function as the other functions only have two inputs. `scilab_call` has as the first input the name of the scilabEnv variable and the second is the function name. The third and fith inputs of the function are the numbers of inputs and outputs respectively. The fourth and sixth inputs are the pointers to the input and output variables. Once the `scilab_call` function is executed, the outputs are written to the scilabVar* variable specificied in the output. Hence, you can see that check and resCh are retrieved from out[1] and out[0] respectively.

6.3.3 Input modifications

In this file, only these two lines modify the input.

```
//Number of variables and constraints
nVars = x0_cols;
nCons = A_rows + Aeq_rows + nonlinCon;
```

```
SmartPtr<minconNLP> Prob = new minconNLP(nVars, nCons, x0ptr,
    Aptr, bptr, Aeqptr, beqptr, A_rows, A_cols, b_rows, b_cols,
     Aeq_rows, Aeq_cols, beq_rows, beq_cols, lbptr, ubptr,
    nonlinCon, nonlinIneqCon);
```

The newly assigned variables are generally the required inputs for the solver library. They are not assigned and passed on from Scilab to reduce the API function usage.

The smart pointer `SmartPtr` is declared in the namespace Ipopt. Here we create a new instance of minconNLP method, which is defined in `root_dir /sci_gateway/cpp/sci_minconNLP.cpp`. minconNLP method is used to pass to modify the inputs in a way favourable to the solver. It is inspired from hs071_nlp.hpp in the MyExample directory in earlier versions of Ipopt. In the latest version of Ipopt, the file is `Ipopt/Ipopt/tutorial/CodingExercise/Cpp /1-skeleton/TutorialCpp_nlp.hpp`

6.3.4 Calling the library

In case of `fmincon`, Ipopt solver is called.

```
SmartPtr<IpoptApplication> app = IpoptApplicationFactory();

////////// Managing the parameters //////////

app->Options()->SetNumericValue("tol", 1e-6);
app->Options()->SetIntegerValue("max_iter", (int)*max_iter);
app->Options()->SetNumericValue("max_cpu_time", *cpu_time);
// app->Options()->SetStringValue("hessian_approximation", "
    limited-memory");

//////// Initialize the IpoptApplication and process the
    options ////////
ApplicationReturnStatus status;
status = app->Initialize();
if (status != Solve_Succeeded)
{
   sciprint("\n*** Error during initialization!\n");
      return (int) status;
    }

 // Ask Ipopt to solve the problem
 status = app->OptimizeTNLP((SmartPtr<TNLP>&)Prob);
```

A new instance of Ipopt instance is made in the first line. We are using the `IpoptApplicationFactory` since this allows us to compile this with an Ipopt Windows DLL. The respective options are passed in the following lines . Hessian approximation option is commented out as we calculate the hessian in the Scilab macro and is already initiated in the `Prob` instance.
In the following lines, the Ipopt library is initialized with the options. If the initialization succeeds, `app->OptimizeTNLP((SmartPtr<TNLP>&)Prob)` solves the problem with the given parameters

6.3.5 Output management

The following lines of code manages the output of the solver that has to passed back to Scilab.

```
//Get the solve statistics
 cpuTime = app->Statistics()->TotalCPUTime();
 app->Statistics()->NumberOfEvaluations(int_fobj_eval,
     int_constr_eval, int_fobj_grad_eval, int_constr_jac_eval,
     int_hess_eval);
 app->Statistics()->Infeasibilities(dual_inf, constr_viol,
     complementarity, kkt_error);
 rstatus = Prob->returnStatus();
 fobj_eval=(double)int_fobj_eval;

////////// Manage the output argument //////////

fX = Prob->getX();
fGrad = Prob->getGrad();
fHess = Prob->getHess();
fLambda = Prob->getLambda();
fZl = Prob->getZl();
fZu = Prob->getZu();
ObjVal = Prob->getObjVal();
iteration = (double)app->Statistics()->IterationCount();
```

The above code is self-explanatory. The solver statistics are extracted form `app` and the problem statistics are extracted from `Prob`.

6.3.6 Returning output to Scilab

These lines of code return the required outputs back to Scilab. They are similar to the Input retrieval functions and are self-explanatory.

```
out[0] = scilab_createDoubleMatrix2d(env, 1, nVars, 0);
   scilab_setDoubleArray(env, out[0], fX);

   out[1] = scilab_createDouble(env, ObjVal);

   out[2] = scilab_createDouble(env, rstatus);
   out[3] = scilab_createDouble(env, iteration);
   out[4] = scilab_createDouble(env, cpuTime);
   out[5] = scilab_createDouble(env, fobj_eval);

   out[6] = scilab_createDouble(env, dual_inf);

   out[7] = scilab_createDoubleMatrix2d(env, 1, nCons, 0);
   scilab_setDoubleArray(env, out[7], fLambda);

   out[8] = scilab_createDoubleMatrix2d(env, 1, nVars, 0);
   scilab_setDoubleArray(env, out[8], fZl);

   out[9] = scilab_createDoubleMatrix2d(env, 1, nVars, 0);
   scilab_setDoubleArray(env, out[9], fZu);
```

```
out[10] = scilab_createDoubleMatrix2d(env, 1, nVars, 0);
scilab_setDoubleArray(env, out[10], fGrad);

out[11] = scilab_createDoubleMatrix2d(env, 1, nVars*nVars,
    0);
scilab_setDoubleArray(env, out[11], fHess);
```

7

Solver Libraries

7.1 Introduction

This chapter deals with the compilation of solver and other libraries that are needed in the toolbox. It explains elaborately the tools and libraries needed to be compiled and related information

7.2 Prerequisites

Before we go ahead with library compilations, we need to setup our environment. The instructions that proceeds are meant for Linux operating systems, specifically Debian OS. Windows User are advised to install Cygwin and take necessary steps where required.Execute the following command on the terminal to install the needed packages

```
$ sudo apt-get install git subversion build-essential gfortran
```

7.3 Compiling libraries

FOSSEE Optimization toolbox directly uses the following libraries for their respective solving domains:

1. CLP
2. Symphony
3. Ipopt
4. Bonmin
5. CBC
6. ecos

But since the dependent libraries of these solvers have gone ahead with many minor releases when compared to the one included in the respective solvers, it is recommended to replace the dependent libraries in the solver library with the latest version.Hence the list of the libraries other than the one given above are :

1. CLP
2. CGL
3. OSI
4. CoinUtils

These libraries can be downloaded directly or by subversion[1] or git. The download information can be found at their respective websites.

We will be giving step-by-step instruction on compilation of the source code of all the libraries required. Many of the steps are same for most of the solvers.

7.3.1 ecos

ecos is lightweight second order conic optimization solver which is written in C. In FOT, `ecos` function uses the ecos library. The steps to download, modify and compile ecos source code are given below:

1. The ecos source code is hosted on github at this link[2]. Download using
   `$ git clone https://github.com/embotech/ecos.git ECOS`.

2. Since there are function names conflicts between ecos and scilab, we have to change the names in ecos source code before we compile. This can be done by running the following commands.

   ```
   $ cd ECOS
   $sed -ir 's/createSparseMatrix/EcosCreateSparseMatrix/g' *
   ```

3. Run the following command to compile ecos source code and generate shared library.

   ```
   $ make shared
   ```

The shared libraries will be generated in `ECOS` and the header files will be in `ECOS/include` and `ECOS/external`.

[1] subversion.tigris.org/

[2] https://github.com/embotech/ecos

7.3.2 CLP

CLP or COIN-OR Linear Programming is an open-source LP solver written in C++. In FOT, `linprog` uses CLP library. The latest version of CLP can be downloaded from the following link[3]. CLP depends on three other COIN-OR projects, namely BuildTools,CoinUtils and OSI. The steps to download, modify and compile CLP source code are given below:

1. Download using `$ svn co https://projects.coin-or.org/svn/Clp/releases /x.xx.xx CLP`.The x.xx.xx in the end of svn link should to be replaced with the respective version that is to be downloaded. Hence CLP will be downloaded into a directory called CLP.

2. Run `get.Lapack`, `get.Mumps` and `getBlas` in Lapack, Mumps and Blas sub-directory in the `CLP/ThirdParty` from the terminal.

```
$ ./CLP/ThirdParty/Lapack/get.Lapack
$ ./CLP/ThirdParty/Blas/get.Blas
$ ./CLP/ThirdParty/Mumps/get.Mumps
```

3. The dependencies on which CLP depends on has to be replaced with the latest version of the same. They can be downloaded from the following link[4]. Download them in the same directory where CLP is downloaded as they will be used by other solvers as well.

```
$ svn co https://projects.coin-or.org/svn/BuildTools/
    releases/x.xx.xx BUILDTOOLS
$ svn co https://projects.coin-or.org/svn/CoinUtils/
    releases/y.yy.yy COINUTILS
$ svn co https://projects.coin-or.org/svn/OSI/releases/z.
    zz.zz OSI
```

4. Move the current dependencies to another directory and replace it with the latest ones that was downloaded in the previous step. For example, for CoinUtils, move the `CLP/CoinUtils` to `CLP/Old.CoinUtils` and move `CoinUtils/CoinUtils` to `CLP/CoinUtils`.Please note that the new CoinUtils directory to be replaced is inside the main CoinUtils directory. The same is done for BuildTools and OSI as well.

```
$ mv CLP/BuildTools CLP/Old.BuildTools
$ mv BUILDTOOLS/BuildTools CLP/BuildTools
$ mv CLP/CoinUtils CLP/Old.CoinUtils
$ mv COINUTILS/CoinUtils CLP/CoinUtils
$ mv CLP/Osi CLP/Old.Osi
$ mv OSI/Osi CLP/Osi
```

[3]https://projects.coin-or.org/svn/Clp/releases/

[4]https://projects.coin-or.org/svn/

5. Make a directory called `build` and go into that directory. Run the `configure` script followed by `make` and `make install`.

```
$ mkdir build
$ cd build
$ ../configure
$ make
$ make install
```

The shared libraries will be generated in `CLP/build/lib` and the header files will be in `CLP/build/include`.

7.3.3 Symphony

Symphony is yet another Linear Programming solver which is an open-source written in C++. In FOT, `symphony` and `symphonymat` uses Symphony. The latest version of Symphony can be downloaded from the following link[5]. Symphony depends the following COIN-OR projects :

1. CGL
2. CLP
3. CoinUtils
4. OSI
5. BuildTools

The steps to download, modify and compile Symphony source code are given below:

1. Download using `$ svn co https://projects.coin-or.org/svn/Symphony/releases /x.xx.xx SYMPHONY`.The x.xx.xx in the end of svn link should to be replaced with the respective version that is to be downloaded. Hence Symphony will be downloaded into a directory called SYMPHONY.

2. Run `get.Lapack` and `getBlas` in Lapack and Blas subdirectory in the `Symphony/ThirdParty` from the terminal.

```
$ ./SYMPHONY/ThirdParty/Lapack/get.Lapack
$ ./SYMPHONY/ThirdParty/Blas/get.Blas
```

3. The dependencies on which Symphony depends on has to be replaced with the latest version of the same. They can be downloaded from

[5] https://projects.coin-or.org/svn/Symphony/releases/

the following link[6]. Download them in the same directory where Symphony is downloaded as they will be used by other solvers as well. Ignore the libraries which have been already downloaded.

```
$ svn co https://projects.coin-or.org/svn/Cgl/releases/x.
    xx.xx CGL
$ svn co https://projects.coin-or.org/svn/Clp/releases/y.
    yy.yy CLP
$ svn co https://projects.coin-or.org/svn/CoinUtils/
    releases/z.zz.zz COINUTILS
$ svn co https://projects.coin-or.org/svn/OSI/releases/w.
    ww.ww OSI
$ svn co https://projects.coin-or.org/svn/BuildTools/
    releases/v.vv.vv BUILDTOOLS
```

4. Move the current dependencies to another directory and replace it with the latest ones that was downloaded in the previous step.

```
$ mv SYMPHONY/Cgl SYMPHONY/Old.Cgl
$ mv CGL/Cgl SYMPHONY/Cgl
$ mv SYMPHONY/Clp SYMPHONY/Old.Clp
$ mv CLP/Clp SYMPHONY/Clp
$ mv SYMPHONY/CoinUtils SYMPHONY/Old.CoinUtils
$ mv COINUTILS/CoinUtils SYMPHONY/CoinUtils
$ mv SYMPHONY/Osi SYMPHONY/Old.Osi
$ mv OSI/Osi SYMPHONY/Osi
$ mv SYMPHONY/BuildTools SYMPHONY/Old.BuildTools
$ mv OSI/BUILDTOOLS SYMPHONY/BuildTools
```

5. Make a directory called `build` and go into that directory. Run the `configure` script followed by `make` and `make install`.

```
$ mkdir build
$ cd build
$ ../configure
$ make
$ make install
```

The shared libraries will be generated in `SYMPHONY/build/lib` and the header files will be in `SYMPHONY/build/include`.

7.3.4 Ipopt

Ipopt or Interior Point Optimizer is a non-linear programming optimization solver which is an open-source written in C++. In FOT, `fminunc`, `fminbnd` , `fmincon`,`fminimax`, `fgoalattain`, `lsqlin`, `lsqnonlin`,`lsqnonneg` and `qpipopt` uses Ipopt. The latest version of Ipopt can be downloaded from the following

[6]https://projects.coin-or.org/svn/

link[7]. Ipopt depends on only BuildTools. The steps to download, modify and compile Ipopt source code are given below:

1. Download using `$ svn co https://projects.coin-or.org/svn/Ipopt/releases /x.xx.xx IPOPT`.The x.xx.xx in the end of svn link should to be replaced with the respective version that is to be downloaded. Hence Ipopt will be downloaded into a directory called IPOPT.

2. Run `get.Lapack`,`get.Mumps` and `getBlas` in Lapack, Mumps and Blas subdirectory in the `IPOPT/ThirdParty` from the terminal.

```
$ ./IPOPT/ThirdParty/Lapack/get.Lapack
$ ./IPOPT/ThirdParty/Mumps/get.Mumps
$ ./IPOPT/ThirdParty/Blas/get.Blas
```

3. Make a directory called `build` and go into that directory. Run the `configure` script followed by `make` and `make install`.

```
$ mkdir build
$ cd build
$ ../configure
$ make
$ make install
```

The shared libraries will be generated in `IPOPT/build/lib` and the header files will be in `IPOPT/build/include`.

7.3.5 CBC

CBC is yet another Linear Programming solver which is an open-source written in C++. In FOT, `cbcintlinprog` and `cbcmatrixlinprog` uses CBC. The latest version of CBC can be downloaded from the following link[8]. CBC depends the following COIN-OR projects :

1. CGL
2. CLP
3. CoinUtils
4. OSI
5. BuildTools

The steps to download, modify and compile CBC source code are given below:

[7] https://projects.coin-or.org/svn/Ipopt/releases/

[8] https://projects.coin-or.org/svn/Cbc/releases/

1. Download using `$ svn co https://projects.coin-or.org/svn/Cbc/releases/x.xx.xx CBC`.The x.xx.xx in the end of svn link should to be replaced with the respective version that is to be downloaded. Hence Cbc will be downloaded into a directory called SYMPHONY.

2. Run `get.Lapack`, `get.Mumps` and `getBlas` in Lapack and Blas subdirectory in the `CBC/ThirdParty` from the terminal.

```
$ ./CBC/ThirdParty/Lapack/get.Lapack
$ ./CBC/ThirdParty/Mumps/get.Mumps
$ ./CBC/ThirdParty/Blas/get.Blas
```

3. The dependencies on which CBC depends on has to be replaced with the latest version of the same. They can be downloaded from the following link[9]. Download them in the same directory where CBC is downloaded as they will be used by other solvers as well. Ignore the libraries which have been already downloaded.

```
$ svn co https://projects.coin-or.org/svn/Cgl/releases/x.
    xx.xx CGL
$ svn co https://projects.coin-or.org/svn/Clp/releases/y.
    yy.yy CLP
$ svn co https://projects.coin-or.org/svn/CoinUtils/
    releases/z.zz.zz COINUTILS
$ svn co https://projects.coin-or.org/svn/OSI/releases/w.
    ww.ww OSI
$ svn co https://projects.coin-or.org/svn/BuildTools/
    releases/w.ww.ww BUILDTOOLS
```

4. Move the current dependencies to another directory and replace it with the latest ones that was downloaded in the previous step.

```
$ mv CBC/Cgl CBC/Old.Cgl
$ mv CGL/Cgl CBC/Cgl
$ mv CBC/Clp CBC/Old.Clp
$ mv CLP/Clp CBC/Clp
$ mv CBC/CoinUtils CBC/Old.CoinUtils
$ mv COINUTILS/CoinUtils CBC/CoinUtils
$ mv CBC/Osi CBC/Old.Osi
$ mv OSI/Osi CBC/Osi
$ mv CBC/BuildTools CBC/Old.BuildTools
$ mv BUILDTOOLS/BuildTools CBC/BuildTools
```

5. Make a directory called `build` and go into that directory. Run the `configure` script followed by `make` and `make install`.

[9]https://projects.coin-or.org/svn/

```
$ mkdir build
$ cd build
$ ../configure
$ make
$ make install
```

The shared libraries will be generated in `CBC/build/lib` and the header files will be in `CBC/build/include`.

7.3.6 Bonmin

Bonmin is yet another Linear Programming solver which is an open-source written in C++. In FOT, `intfminunc`, `intfminbnd`, `intfmincon` and `intfminimax` uses Bonmin. The latest version of Bonmin can be downloaded from the following link[10]. Bonmin depends the following COIN-OR projects :

1. CGL
2. CLP
3. CoinUtils
4. OSI
5. BuildTools
6. Ipopt
7. CBC

The steps to download, modify and compile CBC source code are given below:

1. Download using `$ svn co https://projects.coin-or.org/svn/Bonmin/releases /x.xx.xx BONMIN`.The x.xx.xx in the end of svn link should to be replaced with the respective version that is to be downloaded. Hence Cbc will be downloaded into a directory called BONMIN.

2. Run `get.Lapack`, `get.Mumps` and `getBlas` in Lapack and Blas subdirectory in the `BONMIN/ThirdParty` from the terminal.

```
$ ./BONMIN/ThirdParty/Lapack/get.Lapack
$ ./BONMIN/ThirdParty/Mumps/get.Mumps
$ ./BONMIN/ThirdParty/Blas/get.Blas
```

[10] https://projects.coin-or.org/svn/Bonmin/releases/

3. The dependencies on which CBC depends on has to be replaced with the latest version of the same. They can be downloaded from the following link[11]. Download them in the same directory where CBC is downloaded as they will be used by other solvers as well. Ignore the libraries which have been already downloaded.

```
$ svn co https://projects.coin-or.org/svn/Cgl/releases/x.
    xx.xx CGL
$ svn co https://projects.coin-or.org/svn/Clp/releases/y.
    yy.yy CLP
$ svn co https://projects.coin-or.org/svn/CoinUtils/
    releases/z.zz.zz COINUTILS
$ svn co https://projects.coin-or.org/svn/OSI/releases/w.
    ww.ww OSI
$ svn co https://projects.coin-or.org/svn/Ipopt/releases/w
    .ww.ww IPOPT
$ svn co https://projects.coin-or.org/svn/Cbc/releases/w.
    ww.ww CBC
```

4. Move the current dependencies to another directory and replace it with the latest ones that was downloaded in the previous step.

```
$ mv BONMIN/Cgl BONMIN/Old.Cgl
$ mv CGL/Cgl BONMIN/Cgl
$ mv BONMIN/Clp BONMIN/Old.Clp
$ mv CLP/Clp BONMIN/Clp
$ mv BONMIN/CoinUtils BONMIN/Old.CoinUtils
$ mv COINUTILS/CoinUtils BONMIN/CoinUtils
$ mv BONMIN/Osi BONMIN/Old.Osi
$ mv OSI/Osi BONMIN/Osi
$ mv BONMIN/BuildTools BONMIN/Old.BuildTools
$ mv BUILDTOOLS/BuildTools BONMIN/BuildTools
$ mv BONMIN/Ipopt BONMIN/Old.Ipopt
$ mv IPOPT/Ipopt BONMIN/Ipopt
$ mv BONMIN/Cbc BONMIN/Old.Cbc
$ mv CBC/Cbc BONMIN/Cbc
```

5. Make a directory called `build` and go into that directory. Run the `configure` script followed by `make` and `make install`.

```
$ mkdir build
$ cd build
$ ../configure
$ make
$ make install
```

The shared libraries will be generated in `BONMIN/build/lib` and the header files will be in `BONMIN/build/include`.

[11] https://projects.coin-or.org/svn/

7.4 Shared libraries

Files having extension `*.so*` are called shared or dynamic libraries in Linux. In case of windows they are `*.dll` and `*.lib`. In FOT, the shared libraries have to be compiled for only windows and linux. it has to compiled for different architectures(x64 and x32) as well. The windows dlls are kept in `ROOT_DIR/thirdparty/windows/bin/` and `ROOT_DIR/thirdparty/windows/lib/`. Both of these directories have `x64` and `x32` directories. The Linux shared libraries are in `ROOT_DIR/thirdparty/linux/lib/` which again has `x64` and `x32` directories. Copy all the shared libraries having *.so* extension generated in the previous section into the respective architecture directory.

7.5 Header files

Files having extension `*.h` or `*.hpp` are called header files. The windows header files are kept in `ROOT_DIR/thirdparty/windows/include/`. The linux shared libraries are in `ROOT_DIR/thirdparty/linux/include/`. Copy all the header files generated in the previous section into the these directory.

8

Help Files

8.1 Introduction

The FOSSEE Optimization Toolbox has an extensive help section that covers all of the functions that the toolbox currently consists of. This chapter will explain the basic structure of a standard help document, the methods to make it, and the preferred style to be used while making it. The appendix contains a short tutorial on how to create a simple help document for a toolbox function.

8.2 Basic help document structure

The sections included in a standard help document are:

- Calling Sequence: Describe the calling sequence of the function, mentioning all possible variations of inputs, and outputs.
- Input Parameters: Describe each of the Input Parameters of the function.
- Outputs: Describe each of the Outputs of the function.
- Description: Mathematical description of the type of problem being solved.
- Options: Detail the solver options available to the user.
- Misc.: Describing the exitflags, the output data structure, and other outputs of the function.
- Examples: Provide examples demonstrating the usage of the function, and cases where a solution is not possible.
- Authors: Name the authors.

To get a better idea of the structure, see the structure of a help document currently being used.

fmincon[1]

8.3 Methods of writing help documents

In this section, we touch upon the methods via which you can write the help documents.

8.3.1 Using help_from_sci

Scilab provides means to generate a help document for a macro directly from comments made in it. After defining the function, the user can write the documentation in english, and use the function `help_from_sci` to compile the help document from it. To generate a basic function template with a basic documentation skeleton, use `help_from_sci(funname,helpdir)`.

Sections available in help_from_sci

The following sections are available directly while generating documentation `via help_from_sci`.

- Calling Sequence
- Parameters
- Description
- Examples
- Authors

8.3.2 Directly via XML

This is akin to writing a simple webpage and provides the developer with all the freedom that they have while writing a webpage. The style sheet is already provided within the toolbox, so that won't prove to be a problem. This method, however, will prove to be time consuming because the developer will have to write everything from scratch, and that might prove to be time consuming.

An easier way to go about this might be to generate a basic help page skeleton using help_from_sci and then modifying the generated XML file in help/en_US as needed to make a help document according to the developer's requirements.

[1] http://www.scilab.in/scilab-toolbox- help-files/fmincon.php

8.4 Style Preferences

- Calling Sequence: The calling sequence is to be enclosed in <synopsis> tags.
- The Input parameters and the outputs will be part of a single refsection.
- The problem description, options, and misc. details will be enclosed in a single refsection. LaTeX should be used to provide the mathematical description.
- Examples: LaTeX should be used to specify the problem being solved. The code snippets are to be enclosed in a <programlisting> tag. Each example will be enclosed in a separate refsection.

8.4.1 Using LaTeX

`<latex>` tags have been provided for typesetting mathematical equations. It is advised that it be used within `<para>` tags. Basic LaTeX functionality has been included in this, but no packages are to be used. Equations can be aligned using `{eqnarray}`. There is no need to use standard LaTeX commands to begin documents. The developer can simply enter the `<latex>` tags and start working.

8.5 Additional Notes

The following observations were made while developing the current batch of help documents and the developer is advised to be familiar with them.

8.5.1 Problems faced while using LaTeX

- The developer will face some minor issues while using LaTeX in the documentation. The first and most obvious one will be that in some help files, they will run into an error if they begin a line with an ampersand. This error won't show up on the log, but will prevent the chunk of LaTeX information from rendering in the document. Currently, we do not know what is causing this error, but to work around it, the developer can add, for example, a `\hspace{1pt}` before the ampersand.
- Another thing that the developer will face in the same files that will have the aforementioned ampersand issue is that they will not be able to use multiple ampersands next to each other. The developer is advised to avoid doing so, and use `\qquad` or `hspace`.

- The developer will notice that the term 'some files' has been used. This is because the issue currently affects less than half the files. It specifically affects those files that are at the end of the sorting system, which currently is the alphabetical order.

Appendix A

Codes

A.1 FOSSEE_Optimization_Toolbox.start

```
mprintf("Start FOSSEE Optimization Toolbox\n");

[a, opt] = getversion();
Version = opt(2);

etc_tlbx  = get_absolute_file_path("FOSSEE_Optimization_Toolbox.start");
etc_tlbx  = getshortpathname(etc_tlbx);
root_tlbx = strncpy( etc_tlbx, length(etc_tlbx)-length("\etc\") );

//Load  functions library

mprintf("\tLoad macros\n");
if (getos()=="Windows") then
    pathmacros = pathconvert( root_tlbx ) + "macros_win" + filesep();
    symphony_lib = lib(pathmacros);
    clear pathmacros;
else
    pathmacros = pathconvert( root_tlbx ) + "macros" + filesep();
    symphony_lib = lib(pathmacros);
    clear pathmacros;
end

// load gateways

mprintf("\tLoad gateways\n");
[a, opt] = getversion();
Version = opt(2);
ilib_verbose(0);
if getos()=="Windows" then
    lib_path = root_tlbx + "/thirdparty/windows/bin/" + Version;
```

```
31      link(lib_path+filesep()+"IpOptFSS.dll");
32      link(lib_path+filesep()+"IpOpt-vc10.dll");
33  else
34      lib_path = root_tlbx + "/thirdparty/linux/lib/" + Version;
35      link(lib_path + "/libCoinUtils.so");
36      link(lib_path + "/libcoinblas.so");
37      link(lib_path + "/libcoinlapack.so");
38      link(lib_path + "/libcoinmumps.so");
39      link(lib_path + '/libCbc.so');
40      link(lib_path + "/libClp.so");
41      link(lib_path + "/libClpSolver.so");
42      link(lib_path + "/libOsi.so");
43      link(lib_path + "/libOsiCommonTests.so");
44      link(lib_path + "/libOsiClp.so");
45      link(lib_path + "/libCgl.so");
46      link(lib_path + "/libSym.so");
47      link(lib_path + "/libOsiSym.so");
48      link(lib_path + "/libipopt.so");
49      link(lib_path + '/libbonmin.so');
50      link(lib_path + "/libecos.so");
51  end
52  exec(pathconvert(root_tlbx + filesep() + "sci_gateway" +
        filesep() + "loader_gateway.sce",%f));
53
54  // Load and add help chapter
55
56  if ( %t ) then
57  if or(getscilabmode() == ["NW";"STD"]) then
58          mprintf("\tLoad help\n");
59          path_addchapter = pathconvert(root_tlbx+"/jar");
60          if ( isdir(path_addchapter) <> [] ) then
61                  add_help_chapter("FOSSEE_Optimization_Toolbox",
                        path_addchapter, %F);
62                  clear add_help_chapter;
63          end
64          clear path_addchapter;
65  end
66  end
67
68  // add demos
69
70  if ( %t ) then
71  if or(getscilabmode() == ["NW";"STD"]) then
72      mprintf("\tLoad demos\n");
73      pathdemos = pathconvert(root_tlbx+"/demos/
            sci_FOSSEE_Optimization_Toolbox.dem.gateway.sce",%f,%t)
            ;
74      add_demo("FOSSEE_Optimization_Toolbox",pathdemos);
75      clear pathdemos ;
76  end
77  end
78
79  clear etc_tlbx root_tlbx Version a opt lib_path;
```

A.2 FOSSEE_Optimization_Toolbox.quit

```
function quitModule()

    etc_tlbx  = get_absolute_file_path("
        FOSSEE_Optimization_Toolbox.quit");
    etc_tlbx  = getshortpathname(etc_tlbx);
    root_tlbx = strncpy( etc_tlbx, length(etc_tlbx)-length("\
        etc\") );

    //unlink libraries
    [bOK, ilib] = c_link('FOSSEE_Optimization_Toolbox');
    if bOK then
        ulink(ilib);
    end

    // Remove Preferences GUI
    if getscilabmode() == "STD" then
    removeModulePreferences(root_tlbx);
    end
   ulink();
endfunction

quitModule();

clear quitModule;
```

Appendix B

Tutorial

B.1 Toolbox Tutorial

B.2 Help Tutorial

B.2.1 Introduction

Here we see how to go about making a basic help document for a function in the FOT. Both methods of doing so have been illustrated below:

B.2.2 Using help_from_sci

- Use `help_from_sci()` to generate a basic skeleton for the documentation, along with the function. Save it as `funname.sci`
- Write the function, and the documentation as discussed in the chapter on help documents.
- Enter the command `help_from_sci(funname,helpdir)` to generate the help document from the function.
- Once done, open scilab and build the FOT. You can see the help document on using the help command.

Example

```
--> help_from_sci()
```

This will generate a function template. We save the following function in the template, and modify the filename accordingly.

```
function [y, z]=funname(a, b)
   y=a+b,z=1;
   endfunction
```

After modifying the documentation as necessary, we use the following command to generate a help document in the same directory.

```
--> help_from_sci(funname,".")
```

B.2.3 Modifying the XML

This method has its basis in the method described above, but here the developer has a lot more freedom. In this method, the developer can simply edit the XML file generated by the previous method, and implement a lot of their own ideas about the documentation.

In this method, the developer should generate a basic XML file as displayed in the previous subsection. And then go on to make changes as needed. There isn't a lot to write here because this section depends largely on the developer's comfort with XML.

To view the modifications made, simply rebuild the FOT once the modifications have been made, and use the help command to view them.

Appendix C

Assignments

1. Make a gateway file of a language of your choice that accepts two matrices and returns the
   (a) Sum of both matrices
   (b) Difference of both matrices
   (c) Multiplication of both matrices
   (d) All the above three operations. Use a third input which specifies which operation is to be performed.
2. Go through the sourcecode of lp_solve API given at this link[1]. Understand the implementation and try to make a FOT implementation of the same similar to `intlinprog`.
3. Create the same for GLPK[2].

[1] https://sourceforge.net/projects/lpsolve/files/lpsolve/5.5.2.5/lp_solve_5.5.2.5_scilab_source.tar.gz/download

[2] https://github.com/sqlite/sqlite/tree/master/sciglpk