Semester long Internship Report

On

FOSSEE Optimization Toolbox

Submitted by

Sharvani Laxmi Somayaji
B.Tech (Electrical and electronics engineering)
National Institute of Technology Karnataka

Under the guidance of

Prof. Ashutosh Mahajan
Industrial Engineering and Operations Research
IIT Bombay
and mentor
Mr. Rupak Rokade
FOSSEE, IIT Bombay

August 7, 2021

Acknowledgment

The semester long internship opportunity I had with FOSSEE Team, IIT Bombay, was a great chance for me to learn and experience professional software development. Therefore, I consider myself lucky to have been provided with such a wonderful opportunity. I am also grateful for having a chance to meet so many skilled and talented professionals who led me through this internship.
It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to Prof. Ashutosh Mahajan who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on the correct path.
I would like to express my deepest thanks my mentor, Mr. Rupak Rokade for his continuous support and guidance and taking part in discussions which were extremely valuable for carrying out the tasks. I would also like to thank my team members for their useful insights and discussions and everyone who helped me throughout the internship.
I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. I also hope to continue cooperation with all of you in the future.

Contents

Chapter 1

Introduction

1.1 Scilab

Scilab is a free and open-source, cross-platform numerical computational package and a high-level, numerically oriented programming language. It can be used for signal processing, statistical analysis, image enhancement, fluid dynamics simulations, numerical optimization, and modeling, simulation of explicit and implicit dynamical systems and (if the corresponding toolbox is installed) symbolic manipulations. Scilab is one of the two major open-source alternatives to MATLAB, the other one being GNU Octave.

1.2 FOSSEE Optimization toolbox

FOSSEE Optimization Toolbox (FOT) for Scilab offers several optimization routines including, but not limited to, linear optimization, integer linear optimization, unconstrained optimization, bounded optimization and constrained optimization. The function calls and outputs are similar to those available in MATLAB. These routines call optimization libraries in the backend, most of which are COIN-OR libraries. CLP is used for linear programming, CBC and SYMPHONY for integer linear programming, IPOPT (with MUMPS) for nonlinear optimization and Bonmin for integer nonlinear optimization. There are also routines for specific optimization problems like linear and nonlinear least squares, minimax, and goal programming using these solvers.

Optimization Toolbox Functions:

fot_fgoalattain Solves a multiobjective goal attainment problem.

fot_fminbnd Solves a nonlinear optimization problem on bounded variables.

fot_fmincon Solves a general nonlinear optimization problem.

fot_fminimax Solves a minimax optimization problem.

fot_fminunc Solves an unconstrained optimization problem.

fot_intfminbnd Solves a mixed-integer nonlinear optimization problem on bounded variables.

fot_intfmincon Solves a constrained mixed-integer nonlinear optimization problem.

fot_intfminimax Solves a mixed-integer minimax optimization problem.

fot_intfminunc Solves an unconstrained mixed-integer nonlinear optimization problem.

fot_intlinprog Solves a mixed-integer linear optimization problem.

fot_intquadprog Solves a mixed integer quadratic optimization problem.

fot_linprog Solves a linear optimization problem.

fot_lsqlin Solves a linear least squares optimization problem.

fot_lsqnonlin Solves a nonlinear least squares optimization problem.

fot_lsqnonneg Solves a nonnegative linear least squares optimization problem.

fot_quadprog Solves a quadratic optimization problem.

fot_quadprogCLP Solves a quadratic optimization problem with linear constraints.

fot_quadprogmat Solves a quadratic optimization problem (with input in MATLAB format).

fot_version Displays current versions of various libraries and latest git reference id.

Chapter 2

Task 1 - Exploring and implementing the "limited-memory" option of IPOPT for reducing the computation time

2.1 About the task

Role assigned: Tester.

2.1.1 Issue as stated in the CNES report

We have used "fmincon" on a test case that is more representative of "real" applications (semidirect optimal control method).
The test has been successful, and "fmincon" gave the correct results (as compared to Matlab).

Because the functions (cost + constraints) use numerical integration, computation time is critical.

For instance, with N=15 mesh points for the command, computation time is 45 seconds. But this is after making some improvements in fmincon.sci (calls to execstr, order for finite differences mainly, see below: section 3) which enable a gain of execution time of a factor 6 (computation time with the original "fmincon" would have been 6 times longer). As computation time is function of N^2, you see how fast it would increase.

For this problem, most of the computation time is spent in the function that computes the Hessian. In total the number of function calls per iteration is about 350 (only 16 for Matlab). The reason is that Matlab computes an approximate Hessian by default, which corresponds to the "limited-memory" option of Ipopt. With this option available, the computation time could be much smaller.

2.2 About the function *fot_fmincon*

fot_fmincon is a function available in the FOSSEE Optimization Toolbox which solves a general nonlinear optimization problem. The function calls Ipopt, an optimization library written in C++, to solve the Constrained Optimization problem. The function searches the minimum of a constrained optimization problem, specified by:

$\min_x$ f(x)
Subjected to:

$$
\begin{aligned}
&\text{A} \cdot x \leq b \\
&Aeq \cdot x = beq \\
&c(x) \leq 0 \\
&ceq(x) = 0 \\
&lb \leq \ x \leq ub
\end{aligned}
$$

The calls to `fot_fmincon` can be done in Scilab as follows:

```
xopt = fot_fmincon(f,x0,A,b)
xopt = fot_fmincon(f,x0,A,b,options)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq,options)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq,lb,ub)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq,lb,ub,options)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq,lb,ub,nlc)
xopt = fot_fmincon(f,x0,A,b,Aeq,beq,lb,ub,nlc,options)
[xopt,fopt] = fot_fmincon(...)
[xopt,fopt,exitflag]= fot_fmincon(...)
[xopt,fopt,exitflag,output]= fot_fmincon(...)
[xopt,fopt,exitflag,output,lambda]=fot_fmincon(...)
[xopt,fopt,exitflag,output,lambda,gradient]=fot_fmincon(...)

[xopt,fopt,exitflag,output,lambda,gradient,hessian]=
fot_fmincon(...)
```

2.3 Contribution as a tester

To begin with, I had explored the methods, terminologies and options related to non linear optimization problems. I had then explored the options available in Ipopt, MATLAB, coding in AMPL, Scilab, MATLAB and other tools like Travis CI for testing. I had also explored various examples and sources for testing the function and hessian approximation option. I had made an excel sheet and selected 16 examples from the CUTEst set of different types with varying complexities with a large number of variables and equations and translated around 10 examples from AMPL to Scilab and reviewed and tested other example codes as well. Apart from this, I have written the codes for gradients and hessians for appropriate examples and other necessary code files. I have also tried to optimize these example codes and keep them readable. I have compiled detailed documents related to the outputs for all the problems. More details are present in further sections.

2.4 Finding the appropriate method

In general, when second derivatives can be computed with reasonable computational effort, it is usually a good idea to use them, since then Ipopt normally converges in fewer iterations and is more robust. An exception might be in cases, where your optimization problem has a dense Hessian, i.e., a large percentage of non-zero entries in the Hessian. In such a case, using the quasi-Newton approximation might be better, even if it increases the number of iterations, since with exact second derivatives the computation time per iteration might be significantly higher due to the very large number of non-zero elements in the linear systems that Ipopt solve in order to compute the search direction.
Newton's method is very fast but it requires gradient and hessian matrix evaluated at each iteration.

Some alternatives that avoid calculation of the Hessian:

- Quasi-Newton methods
- finite-difference Newton methods

Algorithms that avoid calculation of the Hessian and storage of any matrix

- steepest descent
- nonlinear conjugate gradient methods
- limited-memory Quasi-Newton methods
- truncated Newton methods

Technology that helps in the calculation of the Hessian:

- automated differentiation

Methods that require no derivatives

- finite difference methods
- Nelder & Meade simplex
- pattern search

Storage and time complexities of some methods are mentioned in the table 2.1

Table 2.1: Storage and step complexities of different methods

Method	**Storage**	**Step Complexity**	**Convergence rate**
CG	$O(n)$	$O(n)$	linear
BFGS/DFP/SR1	$O(n^2)$	$O(n^2)$	superlinear (fast)
L-BFGS	$O(mn)$	$O(mn)$	(r-fast)
Newton (full Hess)	$O(n^2)$	$O(n^3)$	quadratic (v-fast)
Newton (sparse Hess)	$O(n) - O(n^2)$	$O(n) - O(n^3)$	quadratic (v-fast)

The limited-memory option available in Ipopt which is closer to L-BFGS method can be used to approximate the Hessian of the Lagrangian by a limited-memory quasi-Newton

method.

We can use this feature by setting the option hessian approximation to the value "limited-memory". In this case, it is not necessary to implement the Hessian computation method `Ipopt::TNLP::eval_h`. If we are using the C or Fortran interface, we still need to implement these functions, but they should return false or IERR=1, respectively, and don't need to do anything else. Since the Hessian of the Lagrangian is zero for all variables that appear only linearly in the objective and constraint functions, the Hessian approximation should only take place in the space of all nonlinear variables. By default, it is assumed that all variables are nonlinear, but we can tell Ipopt explicitly which variables are nonlinear, using the `Ipopt::TNLP::get_number_of_nonlinear_variables` and `Ipopt::TNLP::get_list_of_nonlinear_variables` methods, see Additional methods in TNLP. (Those methods have been implemented for the AMPL interface, so we would automatically only approximate the Hessian in the space of the nonlinear variables, if we are using the quasi-Newton option for AMPL models.) Currently, those two methods are not available through the C or Fortran interface.

2.5 Writing Examples and testing extensively

2.5.1 CUTEst

CUTEst (**C**onstrained and **U**nconstrained **T**esting **E**nvironment with safe threads) is the latest evolution of CUTE, the constrained and unconstrained testing environment for numerical optimization. It is a versatile testing environment for optimization and linear algebra solvers. The test problems provided are written in so-called Standard Input Format (SIF). A decoder to convert from this format into well-defined Fortran 77 and data files is available as a separate package. Once translated, these files may be manipulated to provide tools suitable for testing optimization packages. Ready-to-use interfaces to existing packages, such as MINOS, SNOPT, filterSQP, Knitro, and more, are available. CUTEst is available on a variety of UNIX platforms, including Linux and is designed to be accessible and easily manageable on heterogeneous networks.

I have selected 16 problems from the CUTE set of PrincetonLib. PrincetonLib is a collection of nonlinear programming (NLP) models. The purpose of the collection is to provide algorithm developers of nonlinear optimization codes with a large and varied set of both theoretical and practical test models. It also aids in the software quality assurance process by providing a set of tools to facilitate benchmarking and performance analysis. The original models are in AMPL format and collected by Robert Vanderbei and colleagues at Princeton University.

2.5.2 Translating and testing

Rosenbrock problem

I had started by taking a simple and popular nonlinear optimization problem, the Rosenbrock problem and writing the scilab code for the same as a base example. The problem is stated as follows:

Find the minimum value of the Rosenbrock function

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

with the following constraints:

$$x_1 + 2x_2 \leq 1$$

at starting point (-1, 2).

The scilab code for this function contains the following components:

```
x0=[-1,2];//initial value
A=[1,2]; b=1; //linear inequality constraint
Aeq=[]; beq=[]; //linear equality constraint
lb=[]; ub=[]; //lower and upper bounds
```

Here, the number of variables, n is 2. The number of inequality constraints, m is 1. `x0` is the initial point for x having dimension (1×2). Linear inequality constraint is of the form

$$A \cdot x \leq b$$

`A` is the matrix containing the coefficients of linear inequality constraints of size (1×2). `b` is the vector related to `A` and represents the linear coefficients in the linear inequality constraints of size (1×1). There are no equality constraints as well as lower and upper bounds for this problem.

```
//objective function
function y=f(x)
 y=100*(x(2) - x(1)^2)^2+(1-x(1))^2;
endfunction
//gradient for the objective function
function y = fGrad(x)
 y = [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
       200*(x(2)-x(1)^2)];
endfunction
```

The objective function, `f(x)` returns the value, `y` which is supposed to be minimized. In this case, the gradient of `f(x)`,

$$\nabla f(x) = \begin{pmatrix} -400 \cdot (x_2 - x_1^2) \cdot x_1 - 2 \cdot (1 - x_1) \\ 200 \cdot (x_2 - x_1^2) \end{pmatrix}$$

The function, `fGrad(x)` returns the gradient of the objective function of size (2×1). The gradient matrix is of the form

$$y_i = \frac{\partial f}{\partial x_i}$$

where i is the variable number which varies from 1 to n.

```
//non linear constraint
function [c,ceq]=nlc(x)
    c=[x(1)^2+x(2)^2-1];
    ceq=[];
endfunction
```

```
//gradient for the non linear constraint
function [cg, ceqg] = cGrad(x)
    cg = [2*x(1),2*x(2)];
    ceqg = [];
endfunction
```

Nonlinear inequality constraints are of the form

$$c(x) \leq 0$$

where c is a vector of constraints, one component for each constraint. Similarly, nonlinear equality constraints are of the form

$$ceq(x) = 0$$

Here, the number of nonliear inequality constraints, m_2 is 1. The number of nonliear equality constraints, m_3 is 0. $c(x)$ and $ceq(x)$ are defined as separate single row vectors where `c` has size (1×1) and `ceq` is an empty vector.
The function `nlc(x)`, represents the gradient of the nonlinear constraints (both equality and inequality) of the problem. The gradient matrix, cg is of the form

$$cg_{i,j} = \frac{\partial c(i)}{\partial x_j}$$

where i is the row number which varies from 1 to m_2 and j is the column number which varies from 1 to n.
It is declared in such a way that gradient of nonlinear inequality constraints are defined first as a separate matrix (cg of size $(m_2 \times n)$ or as an empty matrix), followed by gradient of nonlinear equality constraints as a separate matrix (ceqg of size $(m_3 \times n)$ or as an empty marix).

```
function y = lHess(x,obj,lambda)
    //Hessian of objective
    H = [1200*x(1)^2-400*x(2)+2, -400*x(1);
                -400*x(1), 200];
    //Hessian of nonlinear inequality constraint
    Hc = [2,0;0,2];
    //lambda.ineqnonlin: The Lagrange multipliers for
    //the nonlinear inequality constraints.
    y = obj*H + lambda(1)*Hc;
endfunction
```

The Hessian, H(x) is

$$\begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

The hessian matrix of the objective function, H is of the form

$$H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

where i and j vary from 1 to n.

The Hessian is the Hessian of the Lagrangian, where the Lagrangian

$$L(x, \lambda) = f(x) + \sum \lambda_{g,i} g_i(x) + \sum \lambda_{h,i} h_i(x)$$

where g and h are vector functions representing all inequality and equality constraints respectively (meaning bound, linear, and nonlinear constraints) .

$$g(x) \leq 0, h(x) = 0$$

The Hessian of the Lagrangian is

$$\nabla^2_{xx} L(x, \lambda) = \nabla^2 f(x) + \sum \lambda_{g,i} \nabla^2 g_i(x) + \sum \lambda_{h,i} \nabla^2 h_i(x)$$

`lambda` contains the Lagrange multipliers. The Hessian of the Lagrange Function has been pre-defined. The function, `lHess(x, obj, lambda)` represents the hessian function of the Lagrange in the form of a Symmetric Matrix with input parameters as x, objective factor and lambda. With large number of variables, the hessian matrix can easily become huge.

```
//With hessian approximation off
options = struct("MaxIter", [15000], "CpuTime", [5000], "GradObj",
fGrad, "Hessian", lHess,"GradCon", cGrad,"HessianApproximation", [0]);
```

```
//With hessian approximation on
options = struct("MaxIter", [15000], "CpuTime", [5000], "GradObj",
fGrad, "Hessian", "off","GradCon", cGrad,"HessianApproximation", [1]);
```

Various options under the options structure can be tuned by the user. The "Hessian Approximation" can be set to 0 or 1 to turn hessian approximation off or on respectively. The "CpuTime", "MaxIter" and other parameters need to be tuned according to the problem.

```
[x,fval,exitflag,output]=fot_fmincon(f,
x0,A,b,Aeq,beq,lb,ub,nlc,options)
```

Finally, the function *fot_fmincon* can be called which returns the outputs that contains the required solution, x, value of the function $f(x)$ at x, `exitflag` denoting the reason of termination of the algorithm, `ouput` containing the information about the optimization. More details can be found here.
I had tested the function with few textbook examples and compared the outputs with that obtained in MATLAB.

CUTEst problems

I have picked 16 examples of different types with varying complexities with a large number of variables and equations from this source and translated around 10 examples from AMPL to Scilab and reviewed and tested other example codes as well. In order to test the problems conveniently, I have decreased the value of variables as given in the

AMPL source code so as to keep it within the processing capacity of the testing systems.

To verify the value of the optimal solution, and the optimal value of the objective function at these reduced values, I have tested all the 16+1 examples extensively to the best of my abilities and compared the outputs with that of MATLAB, AMPL, NEOS server (Ipopt). Table 2.2 contains the 16 problems that have been selected along with the AMPL source code and documents containing detailed outputs obtained on my system as well as other online solvers. Details about the versions of the related components and other parameters in my system can be found here.

Table 2.2: CUTEst Problems with AMPL source code and detailed outputs

S. No.	Problem name	AMPL Source Code	Detailed outputs*
1	broydn3d	[problem_link_1]	[output_link_1]
2	cosine	[problem_link_2]	[output_link_2]
3	curly10	[problem_link_3]	[output_link_3]
4	dtoc1l	[problem_link_4]	[output_link_4]
5	hadamard	[problem_link_5]	[output_link_5]
6	hager1	[problem_link_6]	[output_link_6]
7	hager4	[problem_link_7]	[output_link_7]
8	liswet10	[problem_link_8]	[output_link_8]
9	mancino	[problem_link_9]	[output_link_9]
10	msqrtb	[problem_link_10]	[output_link_10]
11	penalty1	[problem_link_11]	[output_link_11]
12	power	[problem_link_12]	[output_link_12]
13	reading2	[problem_link_13]	[output_link_13]
14	sipow3	[problem_link_14]	[output_link_14]
15	tfi2	[problem_link_15]	[output_link_15]
16	ubh1	[problem_link_16]	[output_link_16]

Output details related to the rosenbrock problem can be found here.

* Details about the versions of the related components and other parameters can be found here.

The scilab codes for all the 16+1 problem files can be found in the location, *FOSSEE-Optimization-Toolbox/tests/general tests/CUTEst*

2.5.3 Limitations and challenges faced

- There is a certain computational limit up to which we can currently test the CUTEst problems on FOT (Scilab). This also depends on the capacity of the system (RAM etc). My system crashed few times on testing for certain number of variables and equations. The sparse matrices could be stored in a better manner to increase the limit.
- Some problems converged very slowly, and the outputs for some problems were varying for different solvers. There could be some issues with some problems.
- The numerical values for the optimized output could slightly vary among different systems due to differing system capacities.

- There were other challenges such as difficulty in tuning the iteration limit, tolerance limit, solver limit and other types of limits.

2.6 Executing the files

2.6.1 Steps to locate and execute the CUTEst problems

The test cases can be found in the location *FOSSEE-Optimization-Toolbox/tests/general_tests/CUTEst* in the source repository. In order to execute the CUTEst test cases, perform the following steps:

1. Open Scilab and load the FOSSEE Optimization Toolbox by running `exec loader.sce` on the Scilab console. If the toolbox has not been built, run `exec builder.sce` before the previous command.

2. Navigate to the CUTEst tests folder located at *FOSSEE-Optimization-Toolbox/tests/general_tests/CUTEst*

3. Now you can execute the individual test cases by running the corresponding files. A problem can be executed by executing the command – `exec <problem name>.sce`, where `<problem name>` can be replaced by any of the problem names mentioned above. For instance, to execute the problem `broydn3d`, execute the command – `exec broydn3d.sce`

The expected test results obtained from NEOS Ipopt/AMPL Solver are written at the top in each test file, and one can check the results obtained on the console with them in order to verify whether they are correct.

2.6.2 Steps to Run all the files at once and obtain any output

1. To run all the CUTEst problem files at once, execute the file *run_all.sce* located in the folder, *FOSSEE-Optimization-toolbox-Scilab-6/tests/general_tests/CUTEst/* . This file prints and saves the outputs of all the problems in the CUTEst folder.

2. On executing the file *run_all.sce*, the outputs for all problems (this is not included in the GitHub repository), get saved to *outputs_all.sod*. The *outputs_all.sod* obtained in my system can be found here. There are 17 problems out of which 16 are CUTEst and 1 is the rosenbrock problem. This file can be loaded on scilab to get the output for any problem.
   This file contains:
   `<problem_name>_fval`
   `<problem_name>_x`
   `<problem_name>_output`
   `<problem_name>_exitflag`
   Where `<problem_name>` is same as the filename of the problem file.
   Note: To save outputs of each problem (which is now commented), uncomment the following line in *run_all.sce*:
   `save(p+".sod","fval","output","exitflag","x");`

3. The above outputs for any problem can be obtained by entering the variable name mentioned above. Ex: To obtain the fval or objective function value of hager1

problem, enter the variable `hager1_fval` (similar format for other outputs and problems).

The outputs obtained on executing the file *run_all.sce* on a user's system can be compared with the outputs obtained on my system. I have compiled a document containing those outputs which can be found here.

2.7 Comparison of the outputs

By default, I have kept smaller values or values in the scilab problem files in the GitHub repository such that they'll give output within a short period of time for each problem. Other values can be put and tested as well. The outputs for each problem out of the 16 problems are commented and written at the top in each of the scilab code files.
Table 2.3 shows the comparison of outputs obtained on AMPL, Scilab, NEOS(Ipopt) for the 16 problems. The test cases so written, have been successfully executed in Scilab and have yielded correct results, with marginal errors in a few cases.
For more details and more precision in the numerical values, please refer to the output links for each problem in Table 2.2 and the expanded table.

Table 2.3: Outputs obtained from Scilab, AMPL and NEOS solver for the 16 CUTEst examples

S no.	**Problem**	**Input parameter value1**	**Scilab output*** **	**NEOS output (Ipopt)****	**AMPL output***	**Input parameter value2**	**Scilab output*** **	**NEOS output (Ipopt)****	**AMPL output***
1	broydn3d	N =100	0.0	0.0	0.0	N =1000	0.0	0.0	-
2	cosine	N = 10	-9.0	-9.0	-9.0	N=1000	-999.0	-999.0	-
3	curly10	N=10; K=1	-1003.1629	-1003.1629	-1003.1629	N=100; K=1	-10031.629	-10031.629	-10031.629
4	dtoc1l	n = 20; nx = 2; ny =4;	0.112686	0.1126859	0.112685	n=100; nx = 2; ny = 4;	0.4253683	4.2536827e-01	-
5	hadamard	n=4;	1.0000295	1.0	1.0	n=8;	1.0000058	1.00000015	1.0
6	hager1	N=100;	0.8807988	0.8807988	0.8807988	N=1000;	0.880797	0.880797	-
7	hager4	N=50;	2.7998374	2.7998109	2.7998109	N=500;	2.7945138	2.7945134	-
8	liswet10	N=10; K=2	0.0219117	0.02191107	0.021911	N=1000; K=2	4.9316211	4.93113009	-
9	mancino	N=60;	2.300D-09	3.253933e-27 (scaled)	1.7103E-18	N=100;	2.406D-09	7.55445E-19	5.6975E-19
10	msqrtb	P=3;	0.0	0.0	0.0	P=4,5,10;	-	0.0	0.0
11	penalty1	N=100;	0.0635	7.453036e-10 (scaled)	9.025E-4	N=1000;	2.1897404	6.43949781e-08 (scaled)	-
12	power	N=100;	1.989D-12	9.80437e-30 (scaled)	2.642409E-28	N=1000;	1.891D-10	1.4044554e-28 (scaled)	-
13	reading2	N=50;	-0.0123436	-0.01235311	-0.01235325	N=300;	-0.0124547	-0.01255234	-
14	sipow3	M=100;	0.4964553	0.49645445	0.496454456	M=10000;	0.5356508	0.5356507	-
15	tfi2	M=350;	0.6490412	0.64904119	-	M=500;	0.6490421	0.6490416	-
16	ubh1	n=5; t0=0; tf=1000;	1.244444 (with Hessian Approximation off)	1.2444444	1.244444	n=2000, t0 =0, tf=1000	(Beyond capacity)	1.11600081	-

* The student edition of AMPL is limited to 300 variables and 300 constraints and objectives (after presolve) for nonlinear problems.

** This program contains Ipopt, a library for large-scale nonlinear optimization. Ipopt is released as open source code under the Eclipse Public License (EPL). For more information visit https://github.com/coin-or/Ipopt

*** Details about the versions of the related components and other parameters of my system can be found here.

Chapter 3

Task 2 - "Integer Constraints Not Working" Issue raised on GitHub

3.1 About the task

Role assigned: documentor.

This issue, #43 on github has been raised earlier by a maintainer of the FOT from outside India. It has been observed for the function *fot_intlinprog* to have this problem of the return values to be not integers.

A linear programming problem with integer constraints is mentioned in the issue where the first 6 numbers are the optimised values using standard Scilab function, which are in floating point. The next 6 numbers are supposed to be integers using the function fot_intlinprog with integer constraints, however, x(1) is a floating point number which is not an integer.
The function does not take into account the number of integer constraints.

3.2 About the function *fot_intlinprog*

fot_intlinprog is a function available in the FOSSEE Optimization Toolbox which solves a mixed-integer linear optimization problem in intlinprog format with CBC. The function searches the minimum of a constrained optimization problem, specified by:

$$
\begin{aligned}
&\min_x && C^\top \cdot x \\
&\text{Subjected to:} && \\
& && A \cdot x \leq b \\
& && Aeq \cdot x = beq \\
& && lb \leq \ x \leq ub \\
& && x_i \in \mathbb{Z}, i \in intcon
\end{aligned}
$$

CBC, an optimization library written in C++, is used for solving the linear programming problems. The calls to *fot_intlinprog* can be done in Scilab as follows:

```
xopt = fot_intlinprog(c,intcon,A,b)
xopt = fot_intlinprog(c,intcon,A,b,Aeq,beq)
xopt = fot_intlinprog(c,intcon,A,b,Aeq,beq,lb,ub)
xopt = fot_intlinprog(c,intcon,A,b,Aeq,beq,lb,ub,options)
xopt = fot_intlinprog('path_to_mps_file')
xopt = fot_intlinprog('path_to_mps_file',options)
[xopt,fopt,status,output] = fot_intlinprog(...)
```

3.3 Contribution as a documentor

For the same example in the GitHub issue, the function now gives integer values in the output:

```
A = [-40 0 0 1 0 0 ; 0 -60 0 0 1 0 ; 0 0 -85 0 0 1 ; 0 0 0 -1 -1 -1];
B = [0 0 0 -750]';
C = [200 275 325 1.5 1.8 1.9]';
lb = [1 1 1 1 1 1]';
ub = [8 5 3 840 560 3*85]';
xopt = fot_intlinprog(C,[1 2 3 4 5 6]',A,B,[],[],lb,ub);
--> disp(xopt);
5.
5.
3.
200.
300.
250.
```

I had noted all the changes made and details about the issue. The issue was due to the reason that the function did not take into account integer constraints. `numintcons` in the code file which was only initialized to 0, is now taken into account from the input.
I had documented all the details and tested the function with few example cases taken from the github issue and MATLAB documentation.

Chapter 4

Task 3 - Spoken Tutorials

4.1 About the task

Role assigned: manager.

There were a total 7 tutorial topics decided and the corresponding tutorial script had been drafted. We were expected to verify the code file(s) involved in the tutorial and fix it if it had any errors, solve assignments, report/correct any technical issues or discrepancies in the script and slides.

4.2 Contribution as a manager

As a manager, I had made an excel sheet and divided the work among all of us. I had scheduled meetings according to the requirements and ensured that the task would be completed within time. I had worked on the first 2 subfolders, "2_Linear_Programming_using_linprog_function" and "3_Integer_Programming_using_intlinprog" in the main folder, "FOT_ST_tutorials" which focused on the functions, *fot_intlinprog* and *fot_linprog*. More details are present in further sections.

4.3 Changes made to the files in the subfolders

I have written the codes for the assignments and edited/updated the codes for the problems mentioned. I have made suitable changes to the scripts and slides. These are present in the original "FOT_ST_tutorials" folder. I have also tested the codes for the problems mentioned and few other problems and compared them by writing AMPL code with AMPL solver output. They can be found here.

4.3.1 Subfolder 2_Linear_Programming_using_linprog_function

- The solution to the assignment was slightly incorrect in the slides and the script. This was corrected and the solution code to the assignment has been written in the file *assignment_sol_opt_linprog.sce*.
- Minor changes were done to *opt_linprog.sce*
- The Scilab outputs, AMPL codes and AMPL outputs can be found here.

- The slides file, *linprog.pdf* generated by modifying *linprog.tex* file (in the slides folder) and scripts document, *2_Linear_Programming_using_linprog_function latest* were modified to match each other and the code files. Other changes such as prefixing linprog with "fot_" according to the latest scilab toolbox version and improving the content had been made.

4.3.2 Subfolder 3_Integer_Programming_using_intlinprog

- The solution to the assignment was incorrect in the slides and the script. This was corrected and the solution code to the assignment has been written in the file *assignment_sol_opt_intlinprog.sce*.

- Minor changes were done to *opt_intlinprog2.sce* according to the latest version of scilab, 0.4.1

- The solution to the example problem 1 mentioned in the slides which were present in *opt_intlinprog.sce* were re-written.

- The Scilab outputs, AMPL codes and AMPL outputs can be found here.

- The slides file, *3_Integer_Programming_using_intlinprog.pdf* generated by modifying *3_Integer_Programming_using_intlinprog.tex* file (in the slides folder) and scripts document, *3_Integer_Programming_using_intlinprog_function* were modified to match each other and the code files. Other changes such as prefixing intlinprog with "fot_" according to the latest scilab toolbox version and improving the content has been made.

Chapter 5

Other contributions

5.1 List of other contributions

- The MPS file can now be passed to the function *fot_intlinprog* without throwing any error. This issue has been fixed by making changes to the file, *sci_gateway/cpp/sci_intlinprog_mpscpp.cpp* in line 37 and line 40. [link]
- The exitflag types returned by Ipopt are more than 6 which are not mentioned in the help section. So changes to output message has been done in *fot_fgoalattain.xml*, *fot_fminbnd.xml*, *fot_fmincon.xml*, *fot_fminimax.xml* and *fot_fminunc.xml* files. [link]
- Two examples in *fot_intlinprog.xml* have been commented as they give different type of result from that mentioned in the help section previously. [link] [link]
- The help content in *fot_fgoalattain.xml*, *fot_fmincon.xml* and *fot_fminimax.xml* files are modified to fix typos and logical errors. [link] [link]
- The default option is modified in help section of *fot_fminbnd.xml* to match the one present in .sci file.
  The example 2 in *fot_intfminbnd.xml* has been corrected to give expected results(Optimal solution found). [link]
- The *changelog.txt* has been modified to include the chnages made to the latest version, 0.4.1 [link]
- Generated binaries for the toolbox, raised pull requests and compiled an article with all changes made to the toolbox from version 0.4.0 to 0.4.1
- Checked all the examples in the help section of the toolbox to the best of my abilities before raising the pull request just before generating the binaries for version 0.4.1
- Identified some issues in the code files requiring many changes.

5.1.1 Details about the binaries for toolbox version 0.4.1

- The Linux binary available on https://atoms.scilab.org/toolboxes/FOT/ contains 384 files
- The Windows binary available on https://atoms.scilab.org/toolboxes/FOT/ contains 379 files

Chapter 6

Future enhancements

- The error handling can be improved to prevent crashes or unexpected results.
- In some places the names in the perspective of a user seem to be very confusing which can be changed. Eg: 'params' could be changed to 'options'.
- Sparse matrices can be used in the functions.
- The correctness of the exitflags need to be verified. I found that the exitflags in the code files were differing from that mentioned in the help document for some functions.
- In some function codes, there is an issue with the options.
- If possible, few other functions like genetic algorithm could be added to improve the scope of the toolbox.

Chapter 7

Conclusions

I learned a lot during my internship that will help me professionally and personally. Most of my coding experience had been centered around personal projects or assignments for college. These tend to be very specific to their intended audience and generally aren't very consequential to the wider community.
In contrast, open source projects tend to have a much wider reach, so it was interesting to think about the potential scale of my contributions' impact. This made me feel like my work was consequential, and that made it feel worthwhile.

This internship program helped me gain important knowledge on various aspects of software development. I gained more proficiency in languages such as C++, Scilab, MATLAB. I learned to write and test programs in AMPL, theoretical concepts related to optimization, Travis CI, and also got a chance to explore a large codebase. This gave me exposure to the work environment, development process and much more. Personal lessons
The internship program has increased my teamwork capabilities. Team discussions and brainstorming helped me identify and solve numerous issues.

In a nutshell, this internship has been an excellent and rewarding experience. I can conclude that there have been a lot I've learnt and much more to explore. This has certainly helped in improving my coding skills in terms of testing, developing and debugging. I will strive to use the gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives.
In the end, I would like to thank and appreciate everyone who made my fellowship a superb learning and memorable experience.

Some useful links

- FOT on ATOMS: https://atoms.scilab.org/toolboxes/FOT/
- FOT: https://scilab.in/fossee-scilab-toolbox
- FOT source code: https://github.com/FOSSEE/FOSSEE-Optimization-toolbox
- Link to my forked repository: https://github.com/Sharvani2002/FOSSEE-Optimization-toolbox/tree/Scilab-6
- AMPL solver: https://ampl.com/try-ampl/start/
- NEOS Server (Ipopt): https://neos-server.org/neos/solvers/nco:Ipopt/AMPL.html

References

- https://scilab.in/
- https://in.mathworks.com/help/optim/ug/writing-scalar-objective-functions.htmlbsj1e55
- https://in.mathworks.com/help/optim/ug/fmincon.html
- https://github.com/FOSSEE/FOSSEE-Optimization-toolbox
- https://coin-or.github.io/Ipopt/SPECIALS.html
- https://coin-or.github.io/Ipopt/
- https://slidetodoc.com/unconstrained-optimization-rong-jin-logistic-regression-the-optimization/
- https://www.cs.umd.edu/users/oleary/a607/2008/607unc2hand.pdf
- https://www.math.ntnu.no/emner/TMA4180/2014v/Misc/lecture09.pdf
- https://epubs.stfc.ac.uk/manifestation/9327/RAL-TR-2013-005.pdf