OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User

Jonathan Currie, * **David I. Wilson** **

* *Electrical & Electronic Engineering, Auckland University of Technology, New Zealand*
** *Industrial Information & Control Centre, Auckland University of Technology, New Zealand*

Abstract: For those interested in tackling industrial optimization problems, typical approaches include either purchasing a sophisticated and often specialised solver perhaps with accompanying consulting support, using an internet-based optimization server or using a MATLAB toolbox. While there are a number of open source optimization solvers that enable one to solve a wide range of continuous and discrete, linear and nonlinear, medium and large-scale optimization problems, only a few contain useable pre-compiled binaries for Windows. The initiative described in this work, OPTI, bridges this gap by providing an intuitive object-based general optimization platform that interfaces with many of those freely available, and those with low or no-cost licence requirements, high-quality optimization codes all accessible within the rapid development environment MATLAB. The user needs not compile or build the various tools, but still leverages off the advantages of using high-end desktop hardware (such as 64bit multi-core processors) and remaining in a powerful and familiar development environment.

Keywords: Optimization, Open Source, OPTimization Interface (OPTI), MATLAB

1 Introduction

The ability to optimize is becoming an increasingly important part of engineering design, particularly in this cost-conscious globally-aware environment in which we all now operate. This realisation that optimization is now not just something that is tacked on after a successful prototype is actually running, but is actually something that is considered from the outset. One only needs to look at the number of dedicated engineering design packages such as process simulation, finite element analysis, or multi-physics simulation software packages that now include an optimization module. One can also look at the increasing numbers of undergraduate engineering courses that demand optimization as part of the core engineering education.

However there is a problem: practical optimization problems, i.e. those where it is anticipated that the economic payoff to attempt a computer solution is significant, tend to be large-scale, involve binary decision variables, and show an ordinate sensitivity to formulation. This situation results in groups specializing in optimization with expensive tools applied to hand-crafted situations. This solves the problem for those with such potentially beneficial problems that they can afford to employ the optimization consultants, but the barrier is still very high for those others, the majority of whom could potentially benefit from optimization.

In our experience within an academic-based industrial consulting group, (`i2c2.aut.ac.nz`), we see a strong reluctance to use optimization on a regular and routine basis. The interesting observation is that that reluctance still persists even when the client is competent in the basics of optimization, and is confident in the underlying process models. We believe that the expense of robust high-quality optimization code, coupled with the difficulty in setting up the problem (GAMS/AMPL etc notwithstanding) is to blame. What has changed recently in our dealings with industrial clients primarily in the process, oil and gas and utility industries is the comfort and familiarity, and hence strong reliance on MATLAB. While there could well be more sophisticated proto-typing environments from a computer science perspective (such as Python or Ruby), or environments more cost effective (such as Scilab or Octave), or environments that have garnered domain specific loyalty (such as R amongst statisticians), nothing approaches the market presence of MATLAB for such a wide diverse group comprised of scientists and engineers.

We regularly talk to competent engineers and technologists whose primary interest and expertise is in their own domain. They are not optimization specialists, nor particularly comfortable in sophisticated coding environments, but they do want to solve their particular optimization problem, typically in a hurry. We term this group the 'industrial MATLAB user' as noted in the title of this paper. Often they are not particularly impressed with the assurance of a global optimum since anything better than the status quo is a saving. Finally, it is clear that probably the one reason why they have taken the time and effort to talk to us (academics) is that they have not readily found suitable optimisation toolkits on the internet.

1.1 The optimization problems of interest

The failure to find suitable optimisation toolkits is interesting because optimization problems, at least on the

surface, are succinctly described by a compact mathematical problem, i.e.

$$\min_x \left(\mathcal{J}(\mathbf{x})\right) = f(\mathbf{x})$$

subject to constraints. These constraints may be linear or nonlinear, and may involve continuous or discontinuous variables. However this problem is sufficiently well described so that a number of mathematical programming languages have evolved such as GAMS or AMPL or MPS (or the other half dozen that Wikipedia will display when queried) that allow the user to describe the problem in a common format in order to send to different solvers.

1.2 Outline

This paper describes an optimization framework that we found useful, and from analysing web traffic to our own site, and other similar initiatives we would argue is also of interest to many others. The paper does not introduce a new optimization algorithm, nor do we purport to definitively know the best algorithm for a given purpose. What we do offer though is a platform that allows the user to construct optimization problems in an environment (MATLAB) and on hardware (Windows) that with which they are probably familiar.

Section 2 describes the toolbox and three case studies show how one uses the toolbox in section 3. Finally section 4 ends with some conclusions and a description of some future potential developments.

2 The OPTI toolbox

The contribution of this paper is a flexible optimization toolbox, OPTI, that allows an object-orientated flexible interface to a number of high quality free or low-cost solvers. We made the design decision early on that the toolbox would be MATLAB based and intended for 32 and 64 bit hardware running Windows. These decisions are motivated by our industrial clients' requirements; they are uninterested in solutions running on Linux, typically do not have the motivation to gather, debug the make files and subsequently compile the various applications that are needed in other optimization collections.

We also find that not only are industrial clients willing to invest in MATLAB, but they appreciate the powerful language, the post-processing capabilities, and the ability to interface with external hardware. This tends to alleviate the need to use a specialised mathematical programming language such as AMPL, Fourer et al. [2002] or GAMS, (`www.gams.com`).

2.1 Included solvers

The key component of the toolbox is the collection of solvers designed for a wide variety of optimization problems including linear programs (LPs), quadratic programs (QPs) with optional integer variables (MILPs and MIQPs), and nonlinear optimization. The current list of solvers and a matrix of their capabilities is given in Table 1, and is expected to continue to grow with active development.

The OPTI Toolbox also interfaces to solvers or components that are not strictly open source such as QSOPT which has been kindly provided by the authors for inclusion in the toolbox, as well as binaries such as CPLEX that are accompanied with various, though not particularly onerous for academic users at least, licence restrictions.

We have included hooks to these solvers in our collection, (but not the solver themselves) because, while not strictly open source, they are available to the motivated user to download and register.

Since the toolbox is built around an object, the generic solver can inspect the object to automatically decide the most appropriate algorithm, or check that the requested algorithm is suitable for the optimization problem. We have included our own favourites as defaults in the toolbox, but of course the user is most welcome to overrule these opinions.

2.2 Compiling considerations for the Optimization Libraries

Since most of the solvers in Table 1 are readily available, they can in principle be interfaced with MATLAB. The reality is however less than half of the solvers provided `make` files for compiling with Visual Studio, and fewer still provided working binaries with Windows. Consequently one must not only compile from source the various codes in various languages, but also link them together and interface them all to the Matlab gateway routines. One of the motivations of developing the OPTI toolbox is to relieve the user from these messy and non-trivial details.

A second complication is that the MATLAB MEX files must be compiled for the specific version of MATLAB that are being used, meaning one cannot mix 32bit and 64bit versions. In order to achieve maximum performance and compatibility, we have compiled all solvers from source to both 32bit and 64bit libraries, and compiled win32 and win64 MATLAB interfaces, all supplied with the toolbox.

2.2.1 FORTRAN vs C++ The included solvers listed in Table 1 may be written in FORTRAN, C, C++ or a mixture of all three. An example is the parallel linear equation solver MUMPS, from `graal.ens-lyon.fr/MUMPS/` described in Amestoy et al. [2000], with the core numerical routines implemented in FORTRAN, while the interface API, top level functions and third party graph partitioning library METIS, (`glaros.dtc.umn.edu/gkhome/views/metis`), are coded in C. This causes an added compilation problem, requiring the user to either use `f2c` to convert the code to C (may be inefficient and require the `f2c` library to be linked), or the user to have a separate FORTRAN compiler such as the Intel FORTRAN compiler. The route taken will vary how the FORTRAN functions are called, noting FORTRAN functions will be capitalized, while `f2c` may append a leading underscore. Further problems may also arise if using C++ which 'mangles' the function names if not declared within an 'extern C' block. This can lead to problems if C++ functions are called from FORTRAN.

Most of the compiling issues described above are solved using the right combination of `#define` and preprocessor definitions when compiling and linking, provided the source code is written to accommodate Visual Studio and Windows platforms. Nonetheless determining the correct combination of compiler directives is time consuming and error-prone, especially with limited Windows compilation documentation and detracts from the core task of industrial optimization.

2.2.2 The MEX Interface In order to use any external solver with MATLAB, a MATLAB Executable or MEX file is required. The MEX file implements a standard gateway

Table 1. Optimization solvers incorporated in OPTI *optimisation toolbox. O — Open Source, A — Academic License, $ — Commercial*

Solver	LP	MILP	QP	MIQP	NLS	NLP	MINLP		
BONMIN							×	O	`https://projects.coin-or.org/Bonmin`
CBC		×						O	`https://projects.coin-or.org/Cbc`
CLP	×		×					O	`https://projects.coin-or.org/Clp`
CPLEX	×	×	×	×				A	`www-01.ibm.com/software/integration/optimization/cplex-optimizer`
GLPK	×	×	×					O	`www.gnu.org/software/glpk/glpk.html`
IPOPT						×		O	`www.coin-or.org/Ipopt/`
L-BFGS-B						×		O	`users.eecs.northwestern.edu/~nocedal/lbfgsb.html`
LEVMAR					×			O	`www.ics.forth.gr/~lourakis/levmar/`
LM_DER					×			O	`www.netlib.org/minpack/`
LP_SOLVE	×	×						O	`lpsolve.sourceforge.net/5.5`
MATLAB	×		×		×	×		$	`www.mathworks.com/products/optimization/`
NL2SOL					×			O	`people.sc.fsu.edu/~jburkardt/f_src/nl2sol/nl2sol.html`
NLOPT						×		O	`ab-initio.mit.edu/wiki/index.php/NLopt`
OOQP	×		×					O	`pages.cs.wisc.edu/~swright/ooqp/`
OPTI (built-in)	×	×	×	×			×	O	`www.i2c2.aut.ac.nz`
QSOPT	×							A	`www2.isye.gatech.edu/~wcook/qsopt/`

routine between MATLAB and the solver library, such that it can be called as if it were a built-in MATLAB function. The MEX file is typically written in C or C++, and must be compiled using a MATLAB compatible compiler. While some libraries supply a pre-written MEX interface, or others have a MEX interface supplied by a 3rd party, the task of configuring all the required source files, header files, libraries, preprocessor definitions and configuration settings can be a daunting process.

Therefore a clear advantage of our toolbox supplying pre-built solver binaries effectively skips this compilation step, allowing the user to optimise immediately. However for future solver versions or bug fixes, we intend to provide detailed instructions on how to compile each solver using Visual Studio, and a MATLAB script to automatically include all required header files and libraries and compile the MEX interface in one function call.

For solvers that do not provide a MEX interface we have written one to implement the core functionality of the solver, allowing its use in the toolbox. We have also encountered a few solvers providing either out-of-date, or incorrect MEX interfaces which we have modified to allow their use. Typical problems include problems with sparse matrix indicing or depreciated API calls.

2.3 Setting Solver Options

Solving a large-scale or complex optimization problem will often require the user to customize the solver options in order to obtain faster convergence, and/or a satisfactory solution. Options can include simple adjustments from maximum iterations or tolerances, through to scaling step algorithms and nonlinear variable identification. The OPTI Toolbox provides two means of setting options for all solvers, a common low-level interface, and a solver specific advanced settings interface.

The low-level interface is called `optiset` and allows the user to specify general options such as maximum iterations, display settings, convergence tolerances and timeout settings. These settings are deemed general to most solvers and the `optiset` function allows simple viewing and modifying of the current options. The solver specific interface is a function which exists for each solver, such as `ipoptset` for the nonlinear solver IPOPT. It allows customization of advanced options such as scaling algorithm, Hessian evaluation, linear constraints, and derivative testing. The resulting structure is then passed to `optiset` for inclusion in the problem definition.

2.4 Reading Common Optimization Files

While MATLAB provides a powerful modelling language for defining and simulating complex problems, quite often it is advantageous to utilize existing models created in other standard formats such as MPS or AMPL. Our toolbox includes routines for reading both MPS and LP models which were adapted from the COIN-OR Utilities project, and for parsing AMPL .NL models which was adapted from the AMPL Solver Library from Netlib. These routines generate a general MATLAB structure with appropriate fields completed including function handles for the arbitrary nonlinear functions, which can be passed directly to the OPTI constructor or used by any MATLAB function.

2.5 Alternative Optimization Collections

As acknowledged, collecting optimization solvers and providing a common interface is not unique, but we believe we have added specific functionality which does distinguish this approach. Perhaps the most similar project to the OPTI Toolbox is OpenOpt, (Kroshko, D. L. [2011]), which is a Python based optimization framework. Using many of the same solvers as well as those included in the Python framework (e.g. NumPy and SciPy), OpenOpt can solve a larger range of optimization problems. However other than the solvers supplied with Python, and the supplied OpenOpt solvers, the user is still required to build and compile OPTI supplied solvers such as GLPK and IPOPT.

An alternative package is NLOPT, (Johnson [2011]), which is a C/C++ based nonlinear optimization package which contains over 20 global and local optimization algorithms, including both derivative and derivative free algorithms. NLOPT includes interfaces to number of languages including MATLAB and Python. While a precompiled 32bit Windows DLL is provided, and CMake scripts for automated Visual Studio compiling, the user will still have to

build and link the MEX file for it to run with MATLAB. The OPTI Toolbox provides pre-built versions of NLOPT for 32 and 64bit MATLAB, including a modified API for simpler constraint handling and algorithm selection.

3 Optimization Case Studies

As we have acknowledged, one of the aims of the toolbox was to reduce the complexity of setting up and then solving an optimization problem by providing an easy to use and flexible problem constructor, and, if required, automatically identifying the type of problem to solve. In this section we introduce three case studies and illustrate how one could solve them using the toolbox.

3.1 MPS Formatted Mixed Integer Linear Program

A common problem format for LPs and MILPs is the Mathematical Programming System (MPS) format. For this case study we are using the MILP problem `bienst1.mps` from the benchmark collection available from Mittelmann, H. [2011]. The single line

```
prob = coinRead('bienst1.mps')
```

reads in the MPS file using the COIN-OR Utilities MPS reader and automatically creates a general MATLAB structure containing the optimization problem description. Sparse matrices will be generated for the linear constraints, and if a quadratic objective is specified, for the QP H matrix as well. To build an `opti` object, the constructor is given the problem structure

```
Opt = opti(prob)
```

which will check the inputs for errors, determine the problem type, and return a MATLAB `opti` object with problem description

```
-------------------------------------------------------
Mixed Integer Linear Program (MILP) Optimization
 min f'x
 s.t. Ax <= b
      Aeqx = beq
      lb <= x <= ub
      xi = Integer / Binary
-------------------------------------------------------
Problem Properties:
# Decision Variables: 505
# Constraints: 1165
 # Linear Inequality: 448 [896 nz]
 # Linear Equality: 128 [1288 nz]
 # Bounds: 561
 # Integer Variables: 28
-------------------------------------------------------
```

To actually solve the optimization problem, the user simply passes the object to the `solve` routine

```
[x,fval,exitflag,status] = solve(Opt)
```

which when solved using the GNU LP solver GLPK, returns an optimal solution of 46.75 in 133s. If the user wishes to compare this result with for example the CPLEX implementation, then one simply overrides the default solver and resolves.

```
opts = optiset('solver','CPLEX');
Opt = opti(prob,opts);
[x,fval,exitflag,status] = solve(Opt)
```

In order to obtain a list of the available MILP specific solvers currently installed, the following command can be entered:

```
checkSolver('all_MILP')
-----------------------------------------------
OPTI MILP SOLVERS:
CPLEX - IBM ILOG CPLEX
CBC - COIN-OR Branch and Cut
GLPK - GNU Linear Programming Kit
LP_SOLVE - LP_Solve
OPTI - OPTI Supplied Solver
-----------------------------------------------
```

3.2 Quadratic Programs from Model Predictive Control

In our recent work, Currie and Wilson [2010, 2011] we have developed the jMPC Toolbox for generating and simulating linear Model Predictive Controllers (MPC) within MATLAB, Currie [2011b]. As an optimal control strategy, MPC requires an optimization problem to be solved at each sample. Essentially the MPC algorithm delivers a future sequence of control moves, $\mathbf{\Delta u}$ over the immediate future control horizon, N_c, such that a cost function involving a weighted sum of output deviations and input movements is minimized. The objective function is constrained by linear plant dynamics and possible linear output, input, and input rate constraints. In this formulation, the objective function and constraints are

$$\min_{\mathbf{\Delta u}} j = \frac{1}{2}\mathbf{\Delta u}^T\mathbf{H}\mathbf{\Delta u} + \mathbf{f}^T\mathbf{\Delta u}$$
$$\text{subject to: } \mathbf{A}\mathbf{\Delta u} \leq \mathbf{b}$$

which is a standard Quadratic Program (QP). The characterisation of QPs from an MPC problem is that they are always convex, continuous but can have large dimensions and exhibit ill-conditioning. For unstable systems with long horizons, the magnitude of the matrix elements can require large dynamic ranges.

Furthermore since the QP is to be solved at each sample time, and we are aiming to control high speed applications that demand kilo-hertz sampling rates, we need to carefully test potential QP solvers across a wide range of models, horizons and possible weighting functions.

One of the utilities provided with the jMPC Toolbox, `mpc_qps`, randomly generates a dynamic plant model with a customizable number of inputs, outputs and states, and formulates the relevant MPC QP with selected prediction and control horizons. This problem can then be used to test various QP solvers for speed in order to obtain high speed MPC. Using the OPTI Toolbox, the generated MPC QP can be solved using the following code:

```
QP = mpc_qps(Np,Nc,n_in,n_out,n_states);
prob = optiprob('H',QP.H,'f',QP.f,'ineq',QP.A,QP.b)
Opt = opti(prob)
[x,fval] = solve(Opt)
```

which builds an `opti` problem, passes it to the constructor, and then solves it using the best QP solver available.

Another feature of the OPTI Toolbox is the ability to automatically generate the contour plot with constraints for any two-dimensional optimization problem. For example Fig. 1 shows the objective contours and associated constraints of a simple 2D QP with additional integer constraints such as might be encountered in communication MPC applications where there are a mixture of continuous and logical constraints as noted in Axehill [2005]. Such utility plots can be useful for the new user to better visualise their optimization problems.

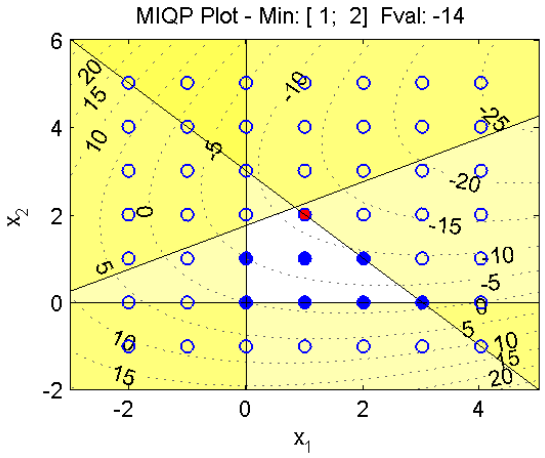

Figure 1. *An automatically generated contour plot of the integer QP optimization problem*

3.3 Steam Utility System Nonlinear Program

Our research centre has also been involved in the optimization of utility systems and has developed the JSteam Toolbox, Currie [2011a], for building and simulating industrial utility systems within MATLAB. At the core of the JSteam toolbox is a hand-optimized dynamically linked library (DLL) written in C++, allowing high speed thermodynamic and unit operation calculations, making it suitable for higher level optimization.

Because of the requirement of the thermodynamic engine DLL to generate the objective function, solving via an online optimizer such as NEOS is not possible due to the dependency on local libraries. Furthermore, in many industrial applications it may be impractical, or even downright forbidden to link to external web-based applications. A better solution is to use a local optimizer installed on the user's PC, possibly interfaced using C or C++ directly to the optimizer, or to a local version of GAMS. Alternatively the user could use MATLAB to handle calling the library functions (via a MEX file or loaded via the .NET interface), and OPTI Toolbox can use a simple function handle just as any other NLP.

An example utility system is presented in Figure 2. The system contains three headers, as well as a mix of common equipment such as turbines and steam users. It also models the complete energy balance of the system, including a demineralized water return and complete deaerator energy model. The two flows exiting the dual stage turbine (red arrows, M_1, M_2) are chosen as the decision variables, to minimize the following cost function:

$$\begin{aligned} &\min_x \left(\mathcal{J}(\mathbf{x})\right) = M_{\mathrm{FG}} C_{\mathrm{FG}} - G_E P_E + M_{\mathrm{DW}} C_{\mathrm{DW}} \\ &\text{subject to: } 0 \leq M_1 \leq M_{1\max} \\ &\qquad\qquad 0 \leq M_2 \leq M_{2\max} \end{aligned}$$

where M_{FG} is the mass flow of fuel gas to the steam boilers, C_{FG} is the cost the fuel gas per unit of mass, G_E is the generated electricity from the turbines, P_E is the price paid for generated electricity, M_{DW} is the mass flow of demineralized water required, and C_{DW} is the cost of demineralized water per unit of mass. The system is bound constrained between no exit flow from a stage in the turbine, and the maximum flow set by the equipment limits.

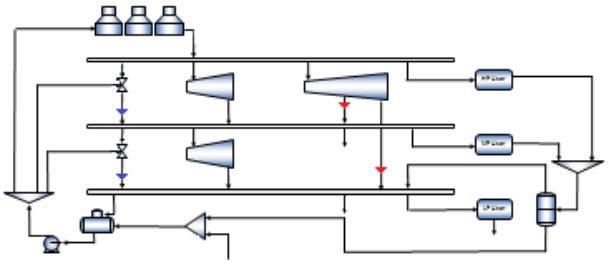

Figure 2. *Nonlinear steam utility system with two decision variables*

The original operating condition had the dual stage turbine operating with minimal flows, and the letdown valves (blue arrows) were making up the existing steam flow required downstream. From inspection the optimum exists when the letdown flows are zero, and the dual stage turbine flows are set to meet the downstream demand, which also generate electricity. This problem exhibits strong nonlinear behaviour because once the dual stage turbine flows exceed the required demand, they cause an excess of steam which is vented, at a cost of fuel gas to generate the vented steam. Using the JSteam Toolbox, the model can be easily built using a `JSteam` object, and saved as a MATLAB function (in this case `Opt3Hdr.m`) to be used by the optimizer:

```
% Nonlinear optimization problem
JStm = JSteam(); % Setup JSteam
optfun = @(x) Opt3Hdr(JStm,x(1),x(2));
% Upper & lower bound Constraints
lb = [0,0];
ub = [75,75];
% Build Problem
prob = optiprob('obj',optfun,'bounds',lb,ub);
% Configure Solver & Build Object
nopts = nloptset('algorithm','LN_NELDERMEAD');
opts = optiset('solver','nlopt','solverOpts',nopts);
Opt = opti(prob,opts)
% Solve optimization problem
x0 = [5,25];
[x,fval,ef,stat] = solve(Opt,x0)
```

As the above problem does not provide a function for the objective function gradient, we have used the derivative free Nelder-Mead solver included in the NLOPT collection. The solver returns in less than 2 seconds with an optimal solution which as estimated, minimized the flows of the letdown valves to the tolerance of the solver specified at 10^{-8}. These types of optimization problems are actively studied given the significant economic savings possible. Several authors have approached this problem using MILP, (Aguilar et al. [2007]), and MINLP (Grossmann [1985], Bruno et al. [1998]), formulations, and the above example shows that a similar integer constrained problem could be posed and solved using the OPTI Toolbox.

A utility function is also provided to automatically benchmark the included solvers for a given problem type. For example to benchmark the supplied NLP solvers, one simply types

```
optiBench('NLP')
```

which will run all available NLP solvers using standard settings across the first 50 in the Hock-Schittkowski collection, Hock and Schittkowski [1981], and generate a

benchmark plot shown in Figure 3. Note only one of the many NLOPT solvers has been used in the benchmark and others could well perform better on individual problems within the collection. We note in passing that IPOPT from A.Wächter [2002] clearly sets the standard for these sorts of problems.

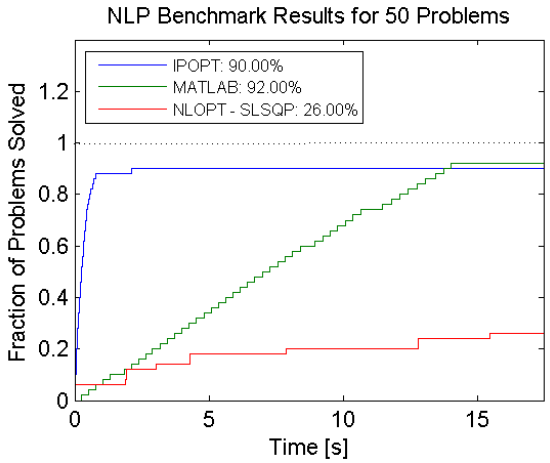

Figure 3. *A comparison of nonlinear optimisers applied to the 50 Hock-Schittkowski test problems. The plot shows the time taken versus the cumulative fraction of solved problems for three different solvers.*

4 Conclusions and an outlook

The development of this optimization toolbox based around a flexible object data structure containing the solvers, gateway routines, various utility functions for data exchange and plotting was motivated by our own needs for developing optimal controllers, and developing utility optimization software for industrial clients. We do not suggest that this initiative is particularly novel, but we would argue that it is unique in areas that are important and largely overlooked by similar initiatives. These include features such as that it is designed for 32 and 64 bit Windows, it does not require the user to compile or assemble the software, and it uses the MATLAB development environment. We also make no particularly strong claims about the quality of our various optimizers, but we do note that some of our included solvers enjoy a high regard in the optimization community.

We have chosen three different illustrative optimization case studies representing the typical types of industrial optimization problems that our research group is readily asked to perform. These include large and sparse LP/MILPs, QPs for high speed controllers, and nonlinear optimization in the steam utility domain. What was important in this paper was not the optimization problem per se, but the ease with which the user can formulate the problem, make changes to the objective function or solver options, or even the solver, and the advantages of embedding this in such a power development environment such as MATLAB for model development, and pre- and post-processing. It is worth noting that the number of hits as recorded on various internet pages dealing with our collection and similar MATLAB-based optimization collections is in the order of tens of thousands. This group, we termed the 'industrial MATLAB user', is clearly looking for an optimization toolkit with a low entry barrier.

Acknowledgments

Financial support for this project from the Industrial Information and Control Centre, School of Engineering, AUT University, New Zealand is gratefully acknowledged.

References

O. Aguilar, S. J. Perry, J.-K. Kim, and R. Smith. Design and optimization of flexible utility systems subject to variable conditions: Part 1: Modelling framework. *Chemical Engineering Research and Design*, 85:1136–1148, 2007.

P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):501 – 520, 2000.

A.Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, USA, 2002.

Daniel Axehill. *Applications of Integer Quadratic Programming in Control and Communication*. PhD thesis, Linköping, Sweden, 2005.

J. C. Bruno, F. Fernandez, F. Castells, and I. E. Grossmann. A rigorous MINLP model for the optimal synthesis and operation of utility plants. *Chemical Engineering Research and Design*, 76:246–258, 1998.

J. Currie. JSteam Toolbox. A MATLAB Steam Modelling Toolbox, 2011a. Available from `www.i2c2.aut.ac.nz`.

J. Currie. jMPC Toolbox. A MATLAB Model Predictive Control Toolbox, 2011b. Available from `www.i2c2.aut.ac.nz`.

Jonathan Currie and David I. Wilson. Lightweight Model Predictive Control intended for embedded applications. In *9th International Symposium on Dynamics and Control of Process Systems (DYCOPS)*, pages 264–269, Leuven, Belgium, 5–7 July 2010.

Jonathan Currie and David I. Wilson. Interpolated Model Predictive Control: Having Your Cake and Eating it Too. In *Australian and New Zealand Annual Chemical Engineering Conference, Chemeca*, Sydney Australia, 18–21 September 2011.

Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole, 2002.

I. E. Grossmann. Mixed-integer programming approach for the synthesis of integrated process flowsheets. *Computers and Chemical Engineering*, 9:463–482, 1985.

W. Hock and K. Schittkowski. Test examples for nonliner programming codes. In *Lecture Notes in Economics and Mathematical Systems*, number 187. Springer, 1981.

Steven G. Johnson. The NLopt nonlinear-optimization package. Available from `ab-initio.mit.edu/nlopt`, 2011.

Kroshko, D. L. OpenOpt. Available from `openopt.org`, 2011.

Mittelmann, H. MILP Benchmark MPS Files. Available from `plato.asu.edu/ftp/milp`, 2011.