Web-based Application Programming Interface to Solve Nonlinear Optimization Problems

João Matias, Aldina Correia, Pedro Mestre *Member, IAENG,* Carlos Fraga, Carlos Serôdio

***Abstract*—Nonlinear Optimization Problems are usual in many engineering fields. Due to its characteristics the objective function of some problems might not be differentiable or its derivatives have complex expressions. There are even cases where an analytical expression of the objective function might not be possible to determine either due to its complexity or its cost (monetary, computational, time, ...). In these cases Nonlinear Optimization methods must be used. An API, including several methods and algorithms to solve constrained and unconstrained optimization problems was implemented. This API can be accessed not only as traditionally, by installing it on the developer and/or user computer, but it can also be accessed remotely using Web Services. As long as there is a network connection to the server where the API is installed, applications always access to the latest API version. Also an Web-based application, using the proposed API, was developed. This application is to be used by users that do not want to integrate methods in applications, and simply want to have a tool to solve Nonlinear Optimization Problems.**

***Index Terms*—Nonlinear Programming, Derivative-free, Web Services, Java, API**

I. Introduction

The main objectives of our work is to study, implement and compare Optimization Algorithms for Nonlinear constrained and unconstrained optimization problems, without the use of derivatives or approximations to them.

A Web Application was developed to solve non-linear problems where the objective function might be non smooth, non linear, non continuous, non convex and with many local minima. Since derivative-based methods are not the most suitable to deal with such kind of problems, so derivative-free optimization methods must be used.

Such problems exist in many real-life situations such as: in problems where the values of the objective function or its constraints are the result of a slow and complex deterministic simulation; when the objective function values are data gathered from experiments; when problems have complex analytical expressions or do not have an analytical expression at all; when the objective function has noise; etc.

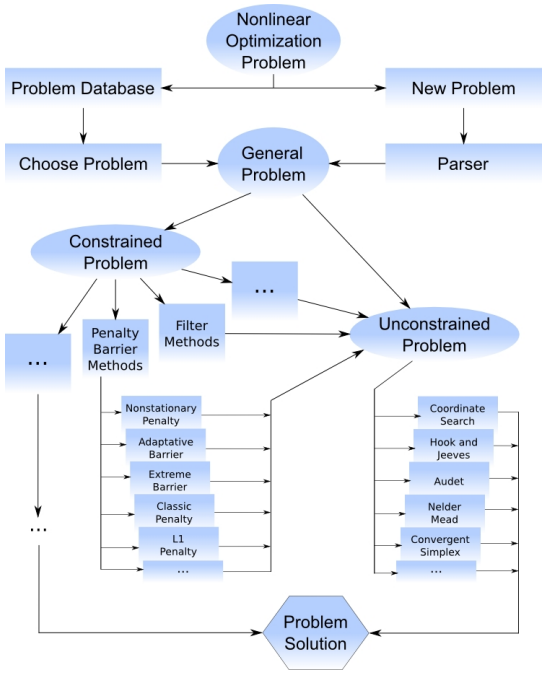

Figure 1. API Block Diagram

The studied methods only need information about the objective function. They advance towards to the optimal based on the comparison of the objective function values in several points. These methods can also be used when the derivatives have discontinuities, when are difficult to determine or when its calculation demands a high computational effort. Problems might have constraints which may also have the mentioned above characteristics, therefore these methods must also be applied to them.

An API (Application Programming Interface) containing the implementation of some the methods and algorithms has been implemented. It is to be used both by the developed software application presented on this paper and by programmers who want to include the developed methods in their projects. It is then possible to integrate the developed methods in other applications such as engineering software packages. This API can be used, as traditionally, installing it locally, or it can be remotely accessed, without the need for its installation, over the Internet using Web Services.

A major advantage of using Web Services is that they allow client applications to be developed in any programming language, despite the fact that Java Technology was used to implement the API.

Manuscript received March 05, 2010.

J. Matias is with CM-UTAD - Centre for the Mathematics, University of Trás-os-Montes and Alto Douro, Vila Real, Portugal, email: j_matias@utad.pt

A. Correia is with ESTGF-IPP, School of Technology and Management of Felgueiras Polytechnic Institute of Porto, Portugal, aldinacorreia@eu.ipp.pt

P. Mestre is with CITAB - Centre for the Research and Technology of Agro-Environment and Biological Sciences, University of Trás-os-Montes and Alto Douro, Vila Real, Portugal, email: pmestre@utad.pt

C. Fraga is with UTAD - University of Trás-os-Montes and Alto Douro, Vila Real, Portugal, email: al24252@utad.eu

C. Serôdio is with CITAB - Centre for the Research and Technology of Agro-Environment and Biological Sciences, University of Trás-os-Montes and Alto Douro, Vila Real, Portugal, email: cserodio@utad.pt

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

II. API Structure

The developed API, and consequently all applications based on it, allows the user to choose either to use a problem stored in the database or to define a problem to be solved. If the first option is chosen, then the user can choose among one of the 25 unconstrained problems or one of the 18 constrained problems already available.

If the second option is selected, the user must then choose between constrained and unconstrained problems, define the objective function and the constraints (if any) and define the initial point. Supplied data is then interpreted by an expression parser and a new problem is generated.

In Fig. 1 are presented all the options available to the user. After choosing the type of problem to be solved, the problem is generated, and then the user can choose one of the available methods.

A. Problems with Constraints

If a problem with constraints is chosen, i.e., a problem of the form:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ s.t. & c_i(x) = 0, i \in \mathcal{E} \\ & c_i(x) \leq 0, i \in \mathcal{I} \end{array} \quad (1)$$

where:

- $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*;
- $c_i(x) = 0,\ i \in \mathcal{E}$, with $\mathcal{E} = \{1, 2, ..., t\}$, define the problem *equality constraints*;
- $c_i(x) \leq 0,\ i \in \mathcal{I}$, with $\mathcal{I} = \{t+1, t+2, ..., m\}$, represent the *inequality constraints*;
- $\Omega = \{x \in \mathbb{R}^n : c_i = 0,\ i \in \mathcal{E} \wedge c_i(x) \leq 0,\ i \in \mathcal{I}\}$ is the set of all feasible points, i.e., the *feasible region*.

the user can then choose between: Penalty and Barrier Methods; Filters Method.

1) Penalty and Barrier Methods: Penalty and Barrier Methods have been created to solve problem P defined in (1), by solving a specially chosen sequence of problems without constraints. The original problem is transformed into a sequence of unconstrained problems (*External Process*) which are solved using methods typically used to solve unconstrained problems (*Internal Process*). Fig. 2 shows the process diagram block.

In these methods a new objective function, Φ, is obtained, which contains information about the initial objective function, f, and the problem constraints, thus the optimality and feasibility are treated together. A succession of unconstrained problems that depend on a positive parameter, r_k, which solutions $x^*(r_k)$ converge to the initial problem solution x^*, is built.

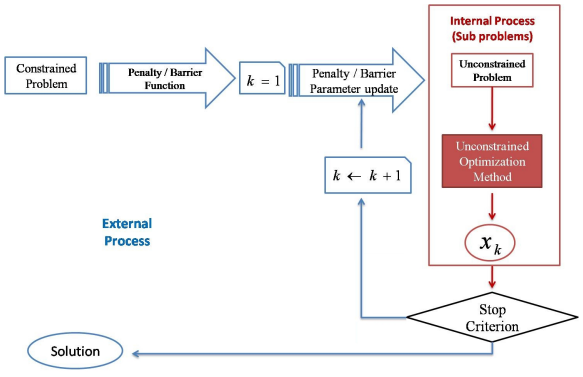

Figure 2. Penalty and Barrier Methods Implementation Diagram Block

Penalty and Barrier Methods, as presented in Fig. 2, are built by two processes:

- External Process - where a succession of unconstrained problems is created;
- Internal Process - where the unconstrained problems are solved.

The new sequence of problems to be solved at each iteration k, that replaces problem P, $B(r_k)$ is defined by:

$$\Phi(x_k, r_k) : \min_{x_k\in\mathbb{R}^n} f(x_k) + r_k p(x) \quad (2)$$

where p is a function that penalises (penalty) or refuses (barrier) points that violates the constraints.

Of the existing Penalty/Barrier functions we implemented the following: External Barrier Function; Progressive Barrier Function; Classical Penalty Function; Static/Dynamic Penalty Function; ℓ_1 Penalty Function.

These methods are adequate for solving problems where a feasible approximation to the solution is needed, however the initial point must also be feasible.

Barrier method has as main objective to dissuade the points x of any approximation to the feasible region border. The External Barrier Function, widely used with Direct Search Methods with feasible points, for example by Audet et. al., [1], [2], [3], [4], [5], [6] is defined by:

$$\Phi(x) = \begin{cases} f(x) & \text{se} \quad x \in \Omega \\ +\infty & \text{se} \quad x \notin \Omega \end{cases} \quad (3)$$

This function works well with the Direct Search Methods to deal with constraints, because these methods use the objective function value only for comparison in the studied points. So if point falls outside or it approaches the feasible region border $\Phi = +\infty$, it is then rejected.

The need for a feasible initial point caused the development of a new version of this method, [5], which can also be chosen by the user in the application. In this second version a relaxable constraints violation measurement function is used $b : X \subset \mathbb{R}^n \to \mathbb{R}_+$, where a maximum is imposed to the violation value, and points that have a value above this limit are rejected. Authors call *Progressive Barrier* to this approach, and MADS-PB (*Mesh Adaptive Direct Search - Progressive Barrier*) to the method. One advantage of this method is its ability of starting the process with points that violates the relaxable constraints and accepts both testing points and interactions with *feasible* violation to this constraints.

In the method the feasible region is defined as

$$\Omega = \{x \in X : c_i(x) \leq 0, i = 1, 2, ..., m\} \subset \mathbb{R}^n,$$

with X the set of non-relaxable constraints (defined by the equality constraints). The relaxable constraints $c_i(x) \leq 0$ are treated using the function $b_X : \mathbb{R}^n \to \mathbb{R}$:

$$\Phi_X(x_k) = f(x_k) + b_X \quad (4)$$

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

where

$$b_X = \begin{cases} \sum_{i=1}^{m} \left(\max\left(c_i\left(x_k\right),0\right)\right)^2 & if \quad x \in X \\ +\infty & if \quad x \notin X \end{cases} \tag{5}$$

Therefore, $b_X(x_k) = 0$ if and only if x verifies all relaxable constraints of the problem, i.e., if $x_k \in \Omega$, and $0 < b_X(x_k) < +\infty$ if x_k violates any of the relaxable constraints. This approach is called progressive barrier since it is attributed a maximum value for the violation at each iteration $h_{k_{max}}$, that is progressively updated (iteration by iteration) using a relation of non dominance, used in the Filters Method, between the tested points and with $h_{k_{max}} \rightarrow 0$ when $k \rightarrow +\infty$.

Other methods of this type are the Penalty Methods. Penalty functions penalise the constraints violation, allowing that infeasible points may occur in the iterative process, although penalised, instead of creating a barrier in the border of the feasible region.

Classic Penalty Functions include the following type of functions:

$$\begin{aligned} \Phi_X(x_k) = & \quad f\left(x_k\right) + r_k p\left(x_k\right) = \\ = & \quad f\left(x_k\right) + r_k \sum_{i=1}^{m} \left[\max\left\{0, c_i\left(x_k\right)\right\}\right]^q, \end{aligned} \tag{6}$$

with $q \geq 1$, and: if $q = 1$, in (6), the function $p(x)$ is called linear penalty function; if $q = 2$, (6) is called a quadratic penalty function, where $r_k \rightarrow +\infty$.

Static Penalty Methods were proposed by Homaifar et al.[7]. In these methods a family of violation levels for each constraint type is used. Each violation level imposes a different penalty. The disadvantage of this method is the number of parameters to be selected, which rapidly increases with the number of constraints and violation levels. A penalty vector is selected for the whole process.

With the penalty vectors $\alpha \in \mathbb{R}^t$ e $\beta \in \mathbb{R}^{m-t}$ it can be built, for problem (1), a Penalty Problem for each iteration k, with $\rho \geq 1$:

$$\min_{x \in \mathbb{R}^n} \Phi_k(x, \alpha, \beta) \tag{7}$$

with

$$\begin{aligned} \Phi_k(x_k, \alpha, \beta) = & \quad f(x_k) + \sum_{i=1}^{t} \alpha_i \left|c_i(x_k)\right|^\rho + \\ + & \quad \sum_{i=t+1}^{m} \beta_i [\max(0, c_i(x_k)]^\rho. \end{aligned} \tag{8}$$

This Penalty Method can be Exact or Inexact. If $\rho = 1$ in (8), it is an Exact Penalty Method, if $\rho > 1$ it is an Inexact Penalty Method, [8].

As an alternative to the search for the penalty parameters by trial-and-error, there are the Dynamic Penalty Methods[9], that gradually increment the penalties in (7) with Φ_k defined in (8). They find the global minima $\tilde{x}$ of (7), for each penalty combination and they stop when $\tilde{x}$ is a feasible solution to P, (1).

Many variants of these methods exist. One of them, which is widely known, is the *Non-stationary Method* that solves a sequence of problems of the same type as (7) with Φ_k defined at (8) and $\rho > 1$, updating the penalty parameters at each iteration k.

ℓ_1 Penalty Method was initially proposed by Pietrzykowski[10], and it has been studied and used by many authors, for example Gould et. al. in [11] and Byrd et. al. in [12], furthermore, it has been the base for many penalty methods.

This is a local minimization Exact Penalty Method and it solves at each iteration k the problem:

$$\min_{x_k \in \mathbb{R}^n} \ell_1^{(k)}(x_k, \mu) \tag{9}$$

with

$$\begin{aligned} \ell_1^{(k)}(x_k, \mu) = & \quad f(x_k) + \mu \sum_{i=1}^{t} |c_i(x_k)| + \\ + & \quad \mu \sum_{i=t+1}^{m} \max[c_i(x_k), 0], \end{aligned} \tag{10}$$

and $\mu \rightarrow +\infty$.

2) Filter Method: To solve a constrained Nonlinear Optimization Problem (NLP), it must be taken in account that the objective is to minimize the objective function and the constraints violation, that must be zero or tend to zero. This involves two concepts: *optimality* (which has the propose of minimize the objective function f) and the *feasibility* (which is intended to minimize the constraints violations c_i).

In the Penalty/Barrier methods, the optimality and feasibility are treated together, however in the Filters Method the concept of bi-objective optimization dominance is used, considering optimality and feasibility separately. In each iteration two phases exist: the first is the feasibility phase and the second is the optimality phase.

The Filters Method was introduced by Fletcher and Leyffer em [13] to globalize SQP (Sequential Quadratic Programming) methods and with the motivation to avoid the difficulty in the penalty parameters and/or the Lagrange multipliers estimation.This method considers the NLP (1) as a bi-objective program, and it has as main goal the minimization of both the objective function (optimality) and a continuous function h that aggregates the problem m constraint functions values (feasibility).

Priority must be given to h since it is not reasonable to have as a problem solution a infeasible point, i.e., that does not comply with the constraints.

Therefore, the function h must be such that:

$h(x) \geq 0$ with $h(x) = 0$ if and only if x is feasible.

We can then define h as:

$$h(x) = \left\|C_+(x)\right\|, \tag{11}$$

where $\|.\|$ is the norm of a vector and $C_+(x)$ is the vector of the $t + m$ values of the constraints in x, i.e, $c_i(x)$ for $i = 1, 2, ..., t + m$:

$$C_+\left(x\right) = \begin{cases} c_i\left(x\right) & \text{if} \quad c_i\left(x\right) > 0 \\ 0 & \text{if} \quad c_i\left(x\right) \leq 0 \end{cases}$$

Considering the norm 2, for example, it is obtained:

$$h\left(x\right) = \left\|C_+\left(x\right)\right\|_2 = \sqrt{\sum_{i=1}^{t+m} \max\left(0, c_i\left(x\right)\right)^2}.$$

The Filters Method define a *forbidden region*, memorizing

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

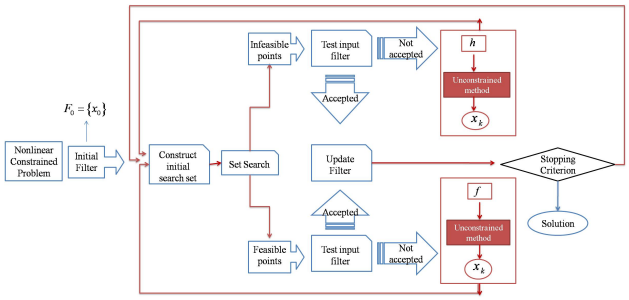

Figure 3. Block diagram of the implemented Filters Method

pairs $(f(x_k), h(x_k))$, with good performance in the previous iterations, avoiding *dominated* points (as defined by the current Pareto rule) by the points of this set, in the next iterations.

a) Definition: A point $x \in \mathbb{R}^n$ *dominates* $y \in \mathbb{R}^n$, and is written as $x \prec y$, if $f(x) \leq f(y)$ and $h(x) \leq h(y)$.

b) Definition: A *filter*, $\mathcal{F}$, is a finite set of points where no pair of points, x and y of the set $\mathcal{F}$, as a relation $x \prec y$, i.e., the filter is made by points such none dominates the other.

A point is accepted by the filter if and only if it is not dominated by other point belonging to the filter and its inclusion eliminates from it all the points that it dominates. We can say that a filter is a dynamic set. A filter works then as a criterion for the iteration acceptance.

In this method a succession of Filters is made $F_0 \subset F_1 ... \subset F_k$, constituted by $(f(x_k), h(x_k)) \in \mathbb{R}^2$ pairs.

Karas [14], to define a temporary pair for the filter, uses the equality $(f(x_k), h(x_k)) = (f(x_k) - \alpha h(x_k), (1-\alpha)h(x_k))$. This modification avoids the acceptance of pairs too close to previous iterations.

Auddet and Dennis, in [1], used for the first time the Filters Method together with the Direct search Methods, namely with Pattern Search Method, showing some convergence results.

Fig 3 presents the diagram block of the Filter Methods implemented in the API.

B. Unconstrained Problems

Both for the Penalty/Barrier and the Filters methods it is needed, in the internal process, to solve unconstrained problems like:

$$\underset{x \in \mathbb{R}^n}{minimize}\ f(x) \tag{12}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function.

Our API and application offers to the user or programmer the following five algorithms to solve such problems:

- A coordinated search algorithm;
- Hooke e Jeeves algorithm;
- A version of Audet et. al. algorithms;
- The Nelder-Mead algorithm;
- A Convergent Simplex algorithm.

The first three are Pattern Search Methods (described, for example, by Conn et. al. in [15], Chapter 7 - *Directional Direct-Search Methods*). These methods determine possible optimal points using fixed directions during the iterative process: starting from a iteration x_k, the next iteration will be found by searching in a pattern or a grid of points, in the directions d, at a distance δ_k (called *step length*).

Last two methods are Simplex Methods (described, for example, by Conn et. al. in [15], Chapter 8 - *Simplicial direct-search methods*). These methods are characterized by starting from an initial simplex and modifying the search directions at the end of each iteration, using movements of refection, expansion and contraction to the inside and the outside, together with the shrunk step towards the best vertex.

III. Options, Variables and Parameters

To solve a problem, regardless of the chosen options, there are parameters that should be defined, while others are set internally, without user intervention.

A. Parameters chosen by the user

For the methods and algorithms to work, besides the problem expression and the initial point, also aditional parameters are needed. Some of them are specific to some methods while others are generic. This last set of parameters, which can be changed by the user, are presented in this subsection.

Input data for the Penalty/Barrier Methods are:

1) Problem to be solved;
2) Initial Point;
3) The penalty/barrier function to be used (1 to 6) - phi_to_use;
4) Initial parameters for the penalty/barrier function;
5) Maximum number of external process iterations-k_{max};
6) Tolerance for the distance between two iterations - $T1$;
7) Tolerance between two values of the objective function in two consecutive iterations - $T2$;
8) Minimum step length - $T3$;
9) Method to be used in the internal process - MET_i;
10) Possible change to change the process parameters of the chosen internal process - $SouN\_i$;
11) Maximum value of the constraints violation - h_{max};
12) Updating factor for the penalty/barrier parameters - γ;

Values defined by default in the API are: : $k_max = 40$, $\alpha = 1$, $T1 = 0.00001$, $T2 = 0.00001$, $T3 = 0.001$ e $\gamma = 2$.

In the filters methods input data are, the points above, 1), 2), 5), 6), 7), 9), 10) and the

- The maximum initial value for the constraints violation - h_{max};

To use the unconstrained methods, previously described in the internal process, it is also needed to define the following parameters:

- Maximum number of internal iterations - $kmax$;
- Initial step length - α:
- Tolerance for the distance between two iterations;
- Tolerance between two values of the objective function in two consecutive iterations;
- Minimum step length;

B. Returning results

The unconstrained methods implemented here, have the following return values:

- Number of objective function evaluations;

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- Last values calculated at the Stop Criteria;
- The found solution;
- Value of the objective function at the found solution;

The Filters Methods return the following parameters:

1) Number of internal process iterations; - k;
2) Number of objective function evaluations;
3) Last iteration;
4) Value of the objective function at the last iteration;
5) Best feasible solution (if any);
6) Value of the objective function at the best feasible solution;
7) Iteration at which the best feasible solution was found;
8) Best infeasible solution (if any);
9) Value of the objective function at the best infeasible solution;
10) Iteration at which the best infeasible solution was found;
11) Value of the constraints violation at the best infeasible solution;
12) Set of non dominated solutions.

Penalty/barrier methods algorithms return the same of the above results, 3), 4), 5), 6), 7), 8), 9), 11) and

- Number of external process iterations;
- Number of Penalty/Barrier function evaluations;
- Value of Penalty/Barrier function at the last iteration;
- Penalty/Barrier function value at the best infeasible solution;
- Iteration were was found the best infeasible solution;
- Constraint violation value at the best infeasible solution;

IV. API Implementation

To implement the API Java technology was chosen because it is a multi-platform technology which has official support (by Sun Microsystems), for the most used Operating Systems. In this type of applications, besides platform portability also performance is a parameter to take in account.

Java has been benchmarked against other programming languages used in this kind of applications, such as C and FORTRAN [16], and it was concluded that it has a good relative performance. In finite element analysis[17] it even has a performance comparable to C. So no performance constraints are expected in our API.

As result this work a set of classes that can be used by Java based applications were implemented. These classes implement the various methods and algorithms discussed in the previous sections.

The API can be accessed in two different ways, one using the standard procedure of installing the `.jar` file containing all the developed classes the developer computer, or remotely accessing the API trough the Local Area Network (LAN) or over the Internet. This last method allows developers and end clients to access always to the latest API version.

A. Using the API

To include the API in Java applications developers only need to include in the classpath the developed API `class` or `.jar` file. A Class exists for each algorithm and method above mentioned. Fig. 4 shows a sample of Java code where the Audet algorithm is used to minimize an expression.

```
String expr = ``(x0-2)^2 + (x1+2)^2'';
String initPoint = ``x0=0.0 x1=0.0'';
AudetProgram audet =
       new AudetProgram(expr,initPoit);
audet.run();
double[] result = audet.getLastResult();
```

Figure 4. Java code to access the locally installed API - in this example the application executed the Audet algorithm.

Input parameters (problem expression and initial point), in this example, are sent to the algorithm using the class constructor. To solve the problem method `run()` is called and results can be obtained by invoking the `getLastResult()` method. This last methods returns a double array which has the problem dimension.

At any moment `setInitialPoint()` and `setExpression()` methods can be invoked to change the initial point or the problem expression, respectively. Besides these methods, the implemented API also includes methods to set and obtain the parameters and results above mentioned in section III.

B. Remote Access using Web Services

Access to the API as presented in the previous section can only be made if the API is installed on the computer where methods are needed. Besides that, it can only be accessed using Java, since this is the programming technology used in its implementation.

To enable remote access to the implemented methods they were made available using Web Services. By using this technology it is possible to access to them, not only using Java, but also other programming languages used in scientific applications [18] such as FORTRAN, C, C++ and the .NET Framework (C# for example).

Although RMI (Remote Method Invocations) has better performance in Java applications than Web Services[19], they are used in because of its wide compatibility. Also, the time spent in communications is much smaller than the time needed to run the optimization methods.

Using the tools provided by the programming technology, user must import the WSDL (Web Service Definition Language) file from the URL (Unified Resource Locator) `http://server.address:port/NLOSolver?wsdl`, using the correct server IP address and port number. After generating all the locally needed files, the API can then be used. Sample code showing how to access the remote API using Java is presented in Fig. 5 and Fig. 6 shows sample C# code to access the remote API.

Methods available in the remote API include: `connect`, to create a new session (needed to deal with multiple access); `runMethod`, to run a specific method or algorithm, for example: `runAudet`, `runNelder`; `getMethodLastResult` to fetch the result of a method last run, for example `getAudetLastResult`; `disconnect` to end the session and free all the resources allocated on the server side.

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
NLOSolverService service =
                  new NLOSolverService();
NLOSolver nlos =
              service.getNLPSolverPort();
String sessionID = nlos.connect();
      String expr = "(x0-2)^2 + (x1+2)^2";
String initPoint = "x0=1 x1=0";
nlos.runAudet(sessionID, expr, initPoint);
List<Double>  result =
     nlos.getAudetLastResult(sessionID);
```

Figure 5. Java code to access to the remote API using Web Services - in this example the application executed the Audet algorithm.

C. Web Application

For solving problems a Web-based application, based on the API, was developed. This application, accessible using a web browser, allows users to solve constrained and unconstrained problems, either by choosing one of the problems previously stored in database or by specifying the problem expression and constraints (in constrained problems).

This application was developed using Java Server Pages (JSP) and it interacts directly with the API. The purpose of this application is to allow users, that do not want to integrated the methods on their applications and simply want to solve a problem. Fig 7 shows the user interface with some of the options available when the user chooses to solve a constrained problem.

V. Conclusion and Future Work

A Java-based API containing methods for solving constrained and unconstrained Nonlinear Optimization Problems has been implemented and successfully tested using local and remote applications. Direct access to the API can be done using Java. Remote access to it can be done using any programming language,as long as it has support do Web Services. Example applications using Java and C# remotely accessing the API has been presented. Also an Web-based application based on the developed API has been presented.

As future work we intend to add the Augmented Lagrangian Method to the available method to solve constrained problems. Further future developments include the storage of the interaction log and various results obtained from method execution in database for future comparison on the methods performance solving various problems, which will be very useful for example for lectures.

References

[1] C. Audet and J. E. D. Jr., "Analysis of generalized pattern searches," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 889–903, (2002).

```
NLPOolverClient nlps =
                  new NLOSolverClient();
String sessionID = nlos.connect();
String expr = "(x0-2)^2 + (x1+2)^2";
String initPoint = "x0=1 x1=0";
nlos.runAudet(sessionID, expr, initPoint);
double[] result =
      nlos.getAudetLastResult(sessionID);
```

Figure 6. C# code to access the remote API using Web Services - in this example the application executed the Audet algorithm.

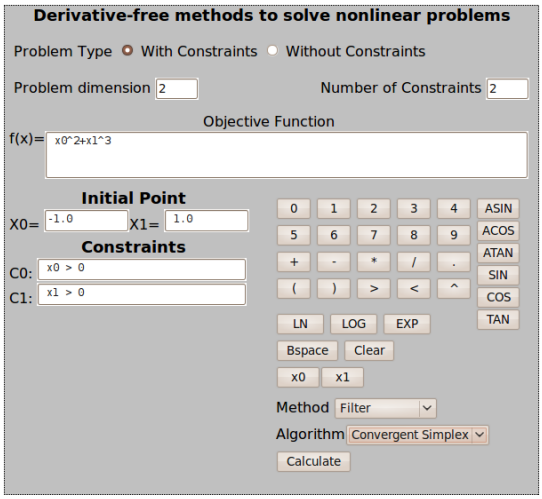

Figure 7. Web-based Application

[2] C. Audet, "Convergence results for pattern search algorithms are tight," *Optimization and Engineering*, vol. 2, no. 5, pp. 101–122, (2004).
[3] C. Audet, V. Béchard, and S. L. Digabel, "Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search," *J. Global Opt.*, no. 41, pp. 299–318, (2008).
[4] C. Audet and J. E. D. Jr., "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, no. 17, pp. 188–217, (2006).
[5] ——, "A mads algorithm with a progressive barrier for derivative-free nonlinear programming," Les Cahiers du GERAD, École Polytechnique de Montréal, Tech. Rep. G-2007-37, (2007).
[6] C. Audet, J. E. D. Jr., and S. L. Digabel, "Globalization strategies for mesh adaptative direct search," Les Cahiers du GERAD, École Polytechnique de Montréal, Tech. Rep. G-2008-74, (2008).
[7] A. Homaifar, S. H. V. Lai, and X. Qi, "Constrained optimization via generic algorithms," *Simulation*, vol. 62, no. 4, pp. 242–254, (1994).
[8] D. P. Bertsekas, *Nonlinear Programming*. Belmont, Massachusetts: Athena Scientific, (1999).
[9] F. Y. Wang and D. Liu, *Advances in Computational Intelligence: Theory And Applications (Series in Intelligent Control and Intelligent Automation)*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., (2006).
[10] T. Pietrzykowski, "An exact potential method for constrained maxima," *SIAM Journal on Numerical Analysis*, vol. 6(2), pp. 299–304, (1969).
[11] N. I. M. Gould, D. Orban, and P. L. Toint, "An interior-point l_1-penalty method for nonlinear optimization," Rutherford Appleton Laboratory Chilton, Tech. Rep., (2003).
[12] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Steering exact penalty methods for nonlinear programming," *Optimization Methods & Software*, vol. 23, no. 2, pp. 197–213, (2008).
[13] R. Fletcher, S. Leyffer, and P. L. Toint, "On the global convergence of an slp-filter algorithm," Dundee University, Dept. of Mathematics, Tech. Rep. NA/183, (1998).
[14] E. W. Karas, A. A. Ribeiro, C. Sagastizábal, and M. Solodov, "A bundle-filter method for nonsmooth convex constrained optimization," *Mathematical Programming*, vol. 1, no. 116, pp. 297–320, (2006).
[15] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Philadelphia, USA: MPS-SIAM Series on Optimization, SIAM, (2009).
[16] J. M. Bull, L. A. Smith, C. Ball, L. Pottage, and R. Freeman, "Benchmarking Java against C and Fortran for scientific applications," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 3-5, pp. 417–430, March-April 2003.
[17] G. P. Nikishkov, Y. G. Nikishkov, and V. V. Savchenko, "Comparison of C and Java performance in finite element computations," *Computers & Structures*, vol. 81, no. 24-25, pp. 2401–2408, September 2003.
[18] R. A. van Engelen, "Pushing the SOAP Envelope WithWeb Services for Scientific Computing," in *Proceedings of the International Conference on Web Services (ICWS)*, 2003, pp. 346–352.
[19] M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko, "Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL," *Journal of Systems and Software*, vol. 79, no. 5, pp. 689–700, May 2006, quality Software.

ISBN: 978-988-18210-8-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)