2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

Algorithm for very fast computation of Least Absolute Value regression

Amin Nobakhti, *MIEEE MIET*, Hong Wang *SMIEEE*
Control Systems Centre
School of Electrical and Electronics Engineering
The University of Manchester
Manchester M60 1QD
nobakhti@iee.org

Tianyou Chai *FIEEE*
Centre of Automation
Northeastern University
Shenyang, P.R. China
tychai@mail.neu.edu.cn

***Abstract*— The Least Squares (LS) problem has been popular in industrial modeling applications due to its speed, efficiency and simplicity. However, the LS solution is known to be unreliable when the data distribution is not Gaussian and is flat-tailed and such data anomalies occur frequently in the industry. The Least Absolute Value (LAV) problem overcomes these difficulties but at the expense of greatly increasing the complexity of the solution. This was partly addressed when it was shown that the LAV problem could be formulated as a Linear Programme (LP). However, the LP formulation is not suitable for implementation in all types of applications. In this paper, a very fast direct search algorithm is developed to solve the general dimension LAV problem using only elementary operations. The algorithm has been shown to be significantly faster than the LP approach through several experiments.**

I. Introduction to data driven modeling

Data driven modeling is an important tool in industrial control and monitoring projects. From forecasting, to condition monitoring and soft-sensor applications, data driven modeling techniques are the superior choice for many routine requirements in the industry.

There are practical reasons why analytical modeling techniques are not popular in the industry. First, they are extremely labor intensive and time consuming projects and unless there are strong reasons, funds will not be available for this amount of resources. In addition, unless they are meticulously updated, they generally have a very short life. An industrial plant may be subject to daily, if not hourly changes. In case of a typical paper machine operating with 5 shifts, there can be multiple changes made to the machine in each of these shift. Changes could be as minor as a new agitator for a mixing tank, to a major new steam system for the dryers. There also frequent modifications to the process, such as new chemical additions, piping reconfigurations etc. Evidently it is a cumbersome task to want to update a model every time there is a process change.

Finally there are also more fundamental reasons such as the difficulty of modeling some of the processes involved. Paper making for example involves several complex mechanical and chemical processes which are heavily interacting. This is especially the case at the Wet End of the machine where important factors such as filler and ash retention are sharply affected by both the performance of the chemical reactions taking place, and also the physical characteristics of the Wire and the set up of the forming foils and vacufoils. Some of these phenomena cannot be easily medelled. These and other reason have led to the popularity of simple data driven modeling in industry. These have the advantage of being able to continuously and rapidly update themselves with the process which could change either due to a change to the process, or because the plant has been shifted to make another grade of product (Indeed, A typical paper machine produces a large variety of products. Products are classified according to their Basis Weight, grade of finish, and coating. Together they may lead to over a hundred combinations of product grammage/grade/finish. A run of a given product/grade may last only as little as 1 hour).

The de facto modeling method of choice in these situations is the least squares regression . The dominance and popularity of the least squares regression can be ascribed, at least partially, to the fact that the theory is simple, well developed and documented. The least squares regression is optimal and results in the maximum likelihood estimators of the unknown parameters of the model if the errors are independent and follow a normal distribution with mean zero and a common variance [1]. The least squares regression is very far from optimal in many non-Gaussian situations, especially when the errors follow distributions with longer tails[2]. For the regression problems, Huber [3] stated that "just a single grossly outlying observation may spoil the least squares estimate, and, moreover, outliers are much harder to spot in the regression than in the simple location case". The outliers occurring with extreme values of the regressor variables can be especially disruptive.

This sensitivity of the LS to bad data is a major weakness in terms of industrial applications. Here, very large disturbances can happen for a variety of reasons which could be as simple as a technician unplugging a sensor, to a chemical addition line filter not being correctly primed, to a sensor which was very badly calibrated during a routine maintenance. There are also more complex reasons for unforeseen and very large process disturbances. For example the Stock Preparation section could be running a trial with a new chemical, which ends up severely affecting the performance of another chemical used in the Wet End. Regardless, the point is that very large disturbances, bad data, and large anomalies occur frequently in an industrial environment and the modeling technique needs to exhibit a certain degree of robustness to these. A method which is related to the LS method is the Least Absolute Value problem which as we will show does have inherent bad data rejection properties.

Least Absolute Value (LAV) regression overcomes these drawbacks and provides an attractive alternative [4]. It is less sensitive than least squares regression to extreme errors, has implicit mechanics to reject bad data and does not require a normal distribution of data which is very unrealistic in practical situations [5]. To see why the LAV approach offers this implicit bad data rejection property, let x_i be an arbitrary number and define m_2 by the value of m which minimizes the sum of the squared differences (i.e. the least squares solution) between m and x,

$$m_2 := \arg\min_m \sum_{i=1}^{N} (m - x_i)^2 \tag{1}$$

It is straightforward to find the minimum by setting the partial derivative of the sum with respect to m equal to zero. We obtain,

$$0 = \sum_{i=1}^{N} 2(m_2 - x_i) \tag{2}$$

or,

$$m_2 = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3}$$

Notice that in the LS case, the solution is the mean of the data observation values. In this case if any of the x_i are very large (say due to a measurement error) it will directly effect the solution. To see the same effect in the LAV case, define now m_1 to be the solution obtained by minimizing the least absolute value as,

$$m_1 := \arg\min_m \sum_{i=1}^{N} \mid m - x_i \mid \tag{4}$$

To find the minimum, the partial derivative with respect to m is set equal to zero,

$$0 = \sum_{i=1}^{N} sgn(m_1 - x_i) \tag{5}$$

The work is supported by the UK EPSRC RAIS Scheme and the NSF China (60534010, 60828007), and the 111 Project (B08015).

where,

$$sgn(x) = \begin{cases} +1, & x > 0 \\ -1, & x < 0 \\ 0, & x = 0 \end{cases} \quad (6)$$

The solution indicates that m_1 should be chosen so that m_1 exceeds x_i for $N/2$ terms; m_1 is less than x_i for $N/2$ terms; and if there is an x_i left in the middle, m_1 equals that x_1. This defines m_1 as the median (for N even, the solution is an interval). Relating the mean and the median to the LS and LAV problems helps to easily see why the LAV approach is inherently more robust. For example suppose that there are three observations from an experiment $\{1, 1.1, 0.9\}$. The LAV solution in this case is 1 and the LS solution is also 1. However, if due to equipment error, or a measurement anomaly, the observations became $\{1, 100, 0.9\}$, then the LAV solution is *still* 1, but the LS solution will change to ≈ 10. This is because in the LAV case, as long as the *bad* data remains in the same side of the median, it has zero effect on the actual value of the median, but in the LS case, any change will directly affect the LS solution.

II. The LAV regression problem

Let the linear regression model with n observations be identified as shows below,

$$y_i = \sum_{i=1}^{m} x_{ij}\beta_j + \epsilon_i, \quad i = 1, 2, \ldots, n, \; j = 1, 2, \ldots, m \quad (7)$$

Let $(x_{i1}, x_{i2}, \ldots, x_{im}, y_i)$ be the ith observations and let $b_0, b_1, \ldots, b_m$ estimated by minimizing the overall absolute values of the differences between the values of $\hat{y}$ and y be the estimates of $\beta_0, \beta_1, \ldots, \beta_m$. So the LAV method is given as follows,

$$\min \left(\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid \right) \quad (8)$$

The LAV regression problem actually predates the least squares solution [6] [7]. It's use however was not popular for many years because unlike least squares, the solution is difficult and not straightforward. It was not until the implementation of the linear programming algorithms on the digital computer that LAV estimates could be obtained for problems of reasonable size [8]. Charnes et al [9] appear to be the first to point out that (8) could be rewritten as a linear programming problem. Subsequently several other LP versions of the LAV problem were developed by [10],[11], [12], [13]. The best known model for the linear regression model is the primary linear regression model developed by [10] and [14]. In this model, the aim is to minimize the overall absolute difference between observations and estimation values, in other words, minimizing the overall error terms. In this respect , goal programming is used to develop the linear programming model which will be used to minimize the overall total positive and negative deviations. That is achieved by [15],

$$\min \left(\sum_{i=1}^{n} (d_i^+ + d_i^-) \right) \quad (9)$$

subject to,

$$y_i - (b_0 + \sum_{i=1}^{m} x_{ij} b_j + d_i^+ - d_i^-) = 0, \quad (10)$$

$$i = 1, 2, \ldots, n \quad (11)$$

$$j = 1, 2, \ldots, m \quad (12)$$

A. A direct search approach

Although as demonstrated the LAV problem is inherently robust to bad data, its calculation is a nontrivial problem. Even in case of the LP formulations which is now regarded as the fastest and most convenient formulation of the LAV problem, there are still difficulties. In particular it is often the case that the modeling and estimation needs to be implemented on older legacy systems or within a limited programming environment. Under these circumstances it is difficult to solve the LP either using the simplex method or using some sort of interior point algorithm.

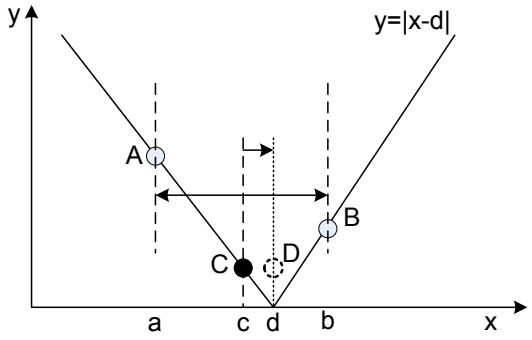

Fig. 1. Simple bisection approach to find the LAV minimum

Indeed, the background to this particular work was a project on online estimation of the Base Sheet Ash measurement at a paper mill in the UK. The modeling and estimation programme had to be implemented on a legacy Honeywell DCS system from the 1980s using the Measurex Programming Language (MPL). This imposed severe memory limitations and no access to more advanced mathematical operators. This led to the development of the algorithm presented in this paper which is able to calculate the LAV solution with a much reduced memory requirement and using only elementary operations. We envisage that the algorithm be useful whenever fast online LAV estimation is required, in embedded systems, or when the LAV problem needs to be implemented in an environment not designed for mathematical programming.

To see the motivations for the choice of algorithm made here, consider Figure 1 which shows the simplest case of the LAV problem. This is a one dimensional problem where the task is to find x which minimizes $y = \mid x - d \mid$. Note that the LAV problem in any dimension is always convex and this feature is useful in developing a simple search method. One way to find the minimum is a direct bisection search heuristic. This might start off by making two initial guesses; points A and B at points $x = a$ and $x = b$ respectably. We can now find the point which corresponds to the mid-point between these two points, in this case, point C at $x = c$. This new point C can the be evaluated, if it leads to a lower function potential, it will replace point A (since it has a higher potential than point B) and the process is repeated between points C and B. Evidently, due to the convexity of the function, if this process is repeated enough times, the solution will converge to point D.

There are however two potential bottlenecks. If both initial choices are on the same side of the minima, then the bisection will converge to the point closest to the minima and not the minima itself. To circumvent this problem, one therefore needs to generate many initial choices (investigate several points in parallel). Secondly, a pure bisection approach is likely to lead to many iterations. Consider what would happens if the new solution from the bisection (i.e. point C) was multiplied by some sort of a correction or weighting factor. In this case one could potentially reach D in just two steps. The reader familiar with optimization techniques should now realize that the features we are looking for, are in fact some of the principal properties of Evolutionary Algorithms, and in particular a recently proposed algorithm called Differential Evolution (DE). As the name suggests, DE works principally by calculating the 'difference vector' between its population members and is an extremely efficient algorithm. Based on a modified version of the DE, i.e. RADE [16] [17], a new simplified algorithm is developed here for fast computation of the LAV solution.

III. Differential Evolution

One of the most successful family of direct search stochastic optimization techniques has been the Evolutionary Algorithm (EA) class of optimisers. Differential Evolution, introduced recently by Storn and Price [18], may also be classed within the family of Evolutionary Algorithms. Nonetheless, they remain principally different to both the Genetic Algorithms (GA) and Evolution Strategies (ES). Amongst the DE's advantages are its simple structure, ease of use and speed of convergence. DE has consistently been ranked as the best search algorithm in several case studies. In [18], DE is compared with two very powerful optimisers; the annealed version of the Nelder and Mead Simplex algorithm and the Adaptive Simulated Annealing (which is claimed to outperform GAs). The DE outperformed both of these algorithms and it was the only one to converge for all of the test problems. In [19], the DE was compared with the simulated annealing M-SIMPSA algorithm [20] and Genetic Algorithms , and was again found to be the only one converging on all problems and providing a better solution when other converged as well. Lin et al [21] applied several MINLP case problems and compared the DE with the MIBBGRG (Mixed Integer Branch and Bound using the Generalized Reduced Gradient algorithm), the MINSLIP (Mixed Integer Nonlinear Sequential Linearisation Programming Algorithm) and the MDSA (Mixed Discrete Simulated Annealing). They found that the DE was the only one able to simultaneously solve all the problems to optimality and satisfy the constraints. In [22] Lampinen and Zelinka report that the DE outperformed several algorithms, including the Branch and Bound, using Sequential Programming, Sequential Linearisation Algorithm, Simulated Annealing, GAs , Evolution Programming and ESs, when tested on Sandgren's problem set.

The population of a DE is subject to three operators of mutation, crossover and selection. Price and Storn proposed several variants of the basic DE which are denoted using the notation $DE/vec/num/mode$. vec is the vector to be mutated, which is either a randomly chosen vector ,'rand', or 'best'=$\vec{x}^{*k}$; the best vector of the current generation. num is the number of difference vectors used in the mutation, which is either 1 or 2 and $mode$ is the method of crossover used. For independent binomial experiments of the genes, this is set to 'bin'. Studies have shown that in general, the two most effective DE strategies are $DE/rand/1/bin$ and $DE/best/2/bin$. The latter is typically slower in converging however is more likely to converge on the global optimum because it will generate several folds more perturbation vectors and thus maintain a higher population diversity [23]. The initial population of the DE is generated uniformly to span the initial boundary of $\vec{x}$,

$$P_i^{(0)} \leftarrow \vec{x}_i^{(0)} = \vec{x}^L + \vec{\rho}^n \,|_0^1\, (\vec{x}^U - \vec{x}^L), \quad i = \{1, \ldots, n_p\}. \tag{13}$$

Subject to,

$$\vec{x}^L \prec \vec{x}^U \in \mathbb{R}^n, \tag{14}$$

where $\vec{x} \prec \vec{y}$ iff $\forall x_i \in \vec{x}$ and $\forall y_i \in \vec{y}, x_i < y_i$.

A. Mutation

Unlike EAs, in DE the mutation amount is derived from a difference vector which is calculated using members of the current populations. For the standard $DE/rand/1/\ldots$ strategy, this is as follows,

$$\vec{u}_i^{k+1} = \vec{x}_{best}^k + F(\vec{x}_s^k - \vec{x}_t^k), \tag{15}$$

$$\text{for } \vec{d} = \{s, t\}. \tag{16}$$

For the $DE/\ldots/2/\ldots$ strategy, four vectors are used,

$$\vec{u}_i^{k+1} = \vec{x}_r^k + F(\vec{x}_s^k - \vec{x}_t^k + \vec{x}_u^k - \vec{x}_v^k), \tag{17}$$

$$\text{for } \vec{d} = \{r, s, t, u, v\}. \tag{18}$$

Where F is the weighting factor and d_j are randomly chosen integers such that $d_l \neq d_k \neq i,\ \forall k, l$. Since the DE operates a greedy selection scheme, $\vec{x}_{best}^k$ also equals the best ever solution found so far. The inherent 'self-adaptation' of the DE is seen. As the population converges to an optimum, any randomly chosen difference vector will becomes smaller in magnitude. Eventually when all members converge to a single solution, the difference vector will be zero and the mutation operator will be disabled all together. Therefore, the actual amount of mutation at iteration k is not only determined by F but also by the population diversity. Indeed [24] has shown that the DE's mutation behavior closely matches that of an ES's correlated self-tuning mutation vectors.

B. Crossover

Once the mutated trial vector $\vec{u}_i^{k+1} = (u_{i1}^{k+1}, \ldots, u_{in}^{k+1})$ is created, it will undergo binomial genewise crossover. For each gene of the trial vector a random number is generated, if this is lower than the crossover rate C_r set by the user, then the gene of the new trial vector is used, if not the gene of the original trial vector $\vec{x}_i^k = (x_{i1}^k, \ldots, x_{in}^k)$ is kept,

$$u_{ij}^{k+1} = \begin{cases} x_{ij}^k, & if \quad \rho \,|_{0_{ij}}^1 < C_r \\ u_{ij}^{k+1}, & else \end{cases}$$
$$i = \{1, \ldots, n_p\}$$
$$j = \{1, \ldots, n\} \tag{19}$$

C. Selection

The selection in DE is deterministic and simple: The resulting trial vector will only replace the original parent if it has a lower objective function value,

$$\vec{x}_i^{k+1} = \begin{cases} \vec{x}_i^k, & if \quad f(\vec{u}_i^{k+1}) > f(\vec{x}_i^k) \\ \vec{u}_i^{k+1}, & else \end{cases}$$
$$i = \{1, \ldots, n_p\} \tag{20}$$

This is the same for all variants of the DE. Although the selection pressure is only one, the best individual of the next generation will be at least as fit as the best individual of the current generation. Similar to a $(\beta, \beta) - ES$, the spread is one, but the best individual can get better whilst the least remain the same. Therefore, there is less loss of diversity than in truncation selection used in ES and this insures a relatively large selection variance.

D. DE Control Variables

DE has three control parameters ; n_p, the population number, F the mutation weighting factor and C_r the cross over rate. The difficulty in use of DE arises in that the choice of these is mainly based on empirical evidence. Moreover many of these guidelines simply state a good *initial starting* value and it is invariably the case that trials and errors will be involved to fine tune these.

The guideline for the number of population is $2n \leq n_p \leq 20n$, and in general the larger n_p, the more robust will be the search, with increased computational cost. The choice of C_r is initially recommended to be $0.3 \leq C_r \leq 0.9$. A large C_r will lead to large perturbations and a slow convergence speed, on the other hand a low C_r will lead to rapid loss of diversity. If $C_r = 1$, the algorithm becomes rotationally invariant and the crossover operator will not yield any new solutions. C_r is typically considered as a 'fine tuning' parameter. The parameter which has by far the largest effect on the search is the mutation weighting factor F. The initial range for this is $F \in [0, 2]$, with $F \in [2/n_p, 1)$ being the recommended range. Its not recommended to set F exactly equal to one as it will mean each potential trial point appears twice, hence the number of different potential candidate solutions is reduced. Its also readily seen that if F is small, it will lead to extensive 'exploitation' and thus a much increased likelihood of misconvergence. On the other hand a large F will lead to over exploration and a significantly reduced convergence speed.

E. Adaptive DE algorithms

As mentioned although the standard DE is inherently adaptive, its sensitivity to the control parameters is well known (see [25] for a general treatment and [23] for the stagnation problem). On the one hand numerous studies have shown that both the convergence

rate and convergence speed of the DE is heavily dependant on the control parameters, especially F. On the other hand, because of the special mutation mechanism in DE, if for whatever reason (such as incorrect choice of F) the DE population looses diversity, then the search will completely stop as mutation value becomes zero. Evidently it is desirable to not have to chose these parameters, and if possible for the algorithm to adaptively determine these.

There have been a handful of generic adaptive DEs developed in the past. Some algorithms such as [26] or [27] rely heavily in human intervention. For example, Zaharie [27] proposes a feedback update rule for F that is designed to maintain the diversity of the population at a given level (and thus stop the search converging prematurely). However the method requires the user to tune a γ parameter which determines the update law for F. Therefore, as the author points out themselves, although the algorithm seems to perform better than the self-adaptive DE proposed in [28], in actual fact the problem merely changes from that of choosing F to choosing γ and there is little real advantage from a practical sense.

Recently the authors developed a self-adaptive algorithm which takes into account the inadequacies of the previous methods and involves no feedback law. This removes the need to have to determine a generic rule, or replace tuning of F with tuning of some other parameters (for example the mean and variance of a distribution). Unlike the standard DE, in the proposed algorithm each $\vec{x}_i$ has its own unique value of F_i^k which is subject to change during the evolution. The Random Adaptive Differential Evolution (RADE) [17] implements a high level ES type of optimization for F. This can be thought of a one parameter ES, where the cost of the i^{th} member, is equivalent to the accumulated objective function value improvement of x_i over the past α iterations. Then an elitist selection will select the best 50 percent members (which will survive into the next round), and new offsprings for the next generation are created at random. Crucially, this means that the F search and the main optimization, run on different time scales. Every one generation of the F search, is equivalent to α generations of the main search. The new F values are created randomly to ensure a high diversity throughout the search. This means, at all times, there will be some members with high values of F (which will exhibit exploratory behavior and are used to escape local minima), some with small F (which will locate the optimum from a proximity near by) and the mean value for F somewhere between the two bounds.

RADE is a generic optimization algorithm which is applicable for the generic class of nonlinear and discontinuous problems. However, in this work, some of the special features of the LAV problem are taken into consideration to develop a faster algorithm based on RADE which will be referred to as fADE

IV. A Fast adaptive DE for LAV computation: fADE

Although the LAV problem is not a smooth function, one of its key geometric features is that its convex. The use of a stochastic search algorithm for solving a convex problem may seem inefficient at first, but as will be shown there are some distinct advantages in this case.

The convexity of the cost function greatly reduces the effectiveness of the crossover operator. In this case, the crossover operator will only yield a better result if the two parent vectors form a difference vector which is not orthogonal with the main function axis (See Figure 2). Clearly this is a special case and the computational cost of performing the crossover operation outweighs the potential benefits arising from the above geometry. Therefore in the fADE algorithm, the crossover operator is removed completely.

The second modification made to RADE is regarding the size of the population. Recall that in the general case of multi-modal function, an EA needs to maintain a large population to avoid entrapment in the function's local minima. However when the function is convex this is no longer the case. However, if a small population is used at the start of the search, there is a chance that there will not be sufficient distribution of initial choices around the global minima causing the search to stagnate. As mentioned the special feature of all DE variant algorithms is that they can stagnate *even on a slope*.

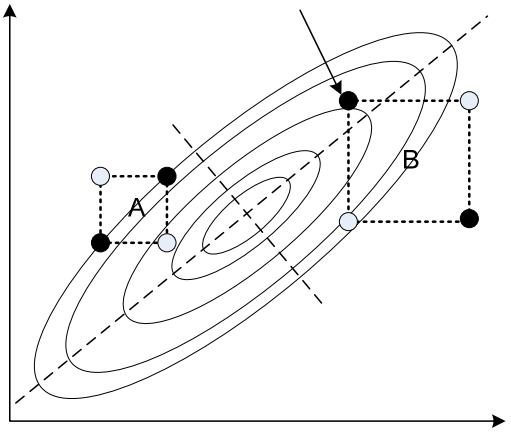

Fig. 2. Showing the crossover operation in case of convex cost function. The parent vectors are shown in gray and the trial vectors are shown in black. Note that of the 4 crossover operations, only one has lead to an improvement in this case

In light of this, the second modification is to dynamically reduce the size of the population. The benefits of this are that it ensures good initial coverage of the surface, but it quickly reduces the size of the population to reduce unnecessary computation at the later stages. A simple heuristic is used to achieve this. Namely at each iteration, the worst member of the population is removed and the size of the population is reduced by one. This approach is not recommended in all cases. All EA algorithms operate under the assumption that each member of the population might posses a set of 'good' genes which in later generations and through recombination with other members will lead to an overall better solution. However, once again the fact that we know aprioir the surface is convex allows us to do this. Namely, in case of the generic multi-modal function, since we never know how many minima exist, a 'bad' member (i.e. poor objective function value), might in fact be closer to the actual global minima than a 'good' member which has just happened to find a local minima. However, since in the LAV case we know there is only one minimum, the member whose objective function value is better, is also guaranteed to be closer to the minimum. In other words, the worst member has no useful genetic information and can be safely discarded. For an m dimensional problem, a lower bound of m members has been defined so that the size of the population will never drop below the dimension of the problem.

In light of the above, we present the following pseudo code for the fADE algorithm.

begin
$k,\ \Gamma^{adapt} \leftarrow 0$
$\alpha,\ \alpha_u \leftarrow 5$
$\gamma^* \leftarrow \infty$
for $(i = 1, \ldots, n_p)$
  % Initialize the initial population
  $P^{(0)} \leftarrow \vec{x}_i^{(0)} = \vec{x}^L + \vec{\rho}\,|_0^1\,(\vec{x}^U - \vec{x}^L)$
  % Evaluate initial members
  $\gamma_i^{(0)} \leftarrow f(\vec{x}_i^{(0)})$
  if $(\gamma_i^0 < \gamma^*)$ **do begin**
    $\gamma^* \leftarrow \gamma_i^0$
    $\vec{x}_{best} \leftarrow \vec{x}_i^0$
  end
  % Assign random F to each member
  $F_i^{(0)} \leftarrow \rho\,|_0^1$
end
% repeat until reached k_{max} or f_{tol}
while (**not** termination condition) **do begin**
  for $(i = 1, \ldots, n_p)$ **do begin**
    $\vec{u}_i^{k+1} \leftarrow \vec{x}_{best} + F_i(\vec{x}_s^k - \vec{x}_t^k + \vec{x}_u^k - \vec{x}_v^k)$
      $(s = \rho \vdash_0^{n_p}, t = \rho \vdash_0^{n_p}, v = \rho \vdash_0^{n_p}, u = \rho \vdash_0^{n_p}$

```
            and ∀k, l, d_k ≠ d_l, where d⃗ = {s, t, u, v})
        % Select if fitter than parent
        if (f(u⃗_i^{k+1}) < γ_i^k) do begin
            x⃗_i^{k+1} ← u⃗_i^{k+1}
            % Update cost vector
            γ_i^{k+1} ← f(u⃗_i^{k+1})
        else do begin
            x⃗_i^{k+1} ← x⃗_i^k
            γ_i^{k+1} ← γ_i^k
        end
        % Update best vector and least cost
        if (γ_i^{k+1} < γ*) do begin
            γ* ← γ_i^{k+1}
            x⃗_best ← x⃗_i^{k+1}
        end
    end
    % Update accumulative adaption vector
    Γ^adapt ← (Γ^adapt + Γ^k − Γ^{k+1})
    % Adaptation loop
    if k = α_u + α do begin
    % Sort vector in decreasing order
    Γ^str = sort(Γ^adapt)
    % Calculate new performance threshold
    T_r = γ^str_⌊n_p/2⌋
        % new F for below threshold performance
        for (i = 1, ..., n_p) do begin
            if γ_i^adapt < T_r do begin
                F_i = ρ |_{F^L}^{F^U}
            end
        end
    Γ^adapt ← 0
    α_u ← α_u + α
    end
% Remove worst performing member, reduce pop
for (i = 1, ..., n_p) do begin
    if γ_i = max(Γ) && n_p > m do begin
        x⃗_i^{k+1} = empty
        n_p = n_p − 1
    end
end
k ← k + 1
end
```

Remark 1: α is the adaptation update interval. If too large, bad choices of F will remain in the population for a long time and adversely effect the search. If on the other hand α is too small, then temporal transient behaviors will affect the adaptation process. The α value of 5 has been found to be the most suitable value. F^L and F^U define the range within which F varies. Although this may be changed for each problem, the values of $F^L = 0.1$ and $F^U = 0.9$ has been found to be suitable for most problems. These settings are fixed and are not required to change for different experiments.

V. Benchmark results

In this section, the regression coefficients of the LP and fADE algorithms are estimated for benchmarking purposes. Each experiment was repeated 10 times and the results given here are the average of those 10 runs. The experiments were performed using MATLAB 7.3 installed on an Intel Pentium Core 2 Due 1.83 GhZ system with 2Gb of RAM. The CPU times (in seconds) and total memory allocation in Kb (in MATLAB) are recorded. In case of the LP problem, MATLAB standard function `linprog` was used (fixed to `LargeScale` option to always use the interior point algorithm).

In all cases, n is the number of observation data points, and m is the dimension of the problem. In case of the LP problem, an interior point (IP) algorithm was used to solve the problem. For the scale of problems considered here, the IP algorithm was substantially faster than the Simplex method which is the other common approach to solve the LP. For fADE, the initial population was always fixed at $10m$ and the algorithm had a 100% convergence rate for all the experiments; Then again this is a convex problem. For both algorithms the same function tolerance was used. The results are given in Tables 1 and 2 and are shown graphically in Figure 3 and 4.

	Proposed Algorithm	LP (Interior Point)
m=2 n=1000	CPU=0.0344 Mem=24	CPU=0.2641 Mem=16040
m=3 n=1000	CPU=0.0688 Mem=32	CPU=0.2672 Mem=16048
m=4 n=1000	CPU=0.1266 Mem=40	CPU=0.2813 Mem=16056
m=5 n=1000	CPU=0.1812 Mem=48	CPU=0.2875 Mem=16064
m=6 n=1000	CPU=0.2953 Mem=56	CPU=0.3016 Mem=16072
m=7 n=1000	CPU=0.5078 Mem=64	CPU=0.3172 Mem=16080
m=8 n=1000	CPU=0.6656 Mem=72	CPU=0.3234 Mem=16088
m=9 n=1000	CPU=0.8250 Mem=80	CPU=0.3359 Mem=16096
m=10 n=1000	CPU=1.1516 Mem=88	CPU=0.3516 Mem=16104

TABLE I
BENCHMARK RESULTS FOR VARYING PROBLEM DIMENSION

	Proposed Algorithm	LP (Interior Point)
m=5 n=500	CPU=0.0859 Mem=24	CPU=0.1922 Mem=7884
m=5 n=800	CPU=0.1266 Mem=38.4	CPU=0.2266 Mem=10291
m=5 n=1100	CPU=0.1047 Mem=52.8	CPU=0.3172 Mem=19430
m=5 n=1400	CPU=0.2047 Mem=67.2	CPU=0.4359 Mem=31450
m=5 n=1700	CPU=0.2266 Mem=81.6	CPU=0.5703 Mem=46349
m=5 n=2000	CPU=0.2594 Mem=96	CPU=0.7328 Mem=64128
m=5 n=2300	CPU=0.2375 Mem=110.4	CPU=0.9125 Mem=84787
m=5 n=2600	CPU=0.2484 Mem=124.8	CPU=1.1000 Mem=108330
m=5 n=2900	CPU=0.3406 Mem=139.2	CPU=1.3250 Mem=134750
m=5 n=3200	CPU=0.3681 Mem=153.6	CPU=1.5703 Mem=164040

TABLE II
BENCHMARK RESULTS FOR VARYING NUMBER OF OBSERVATIONS

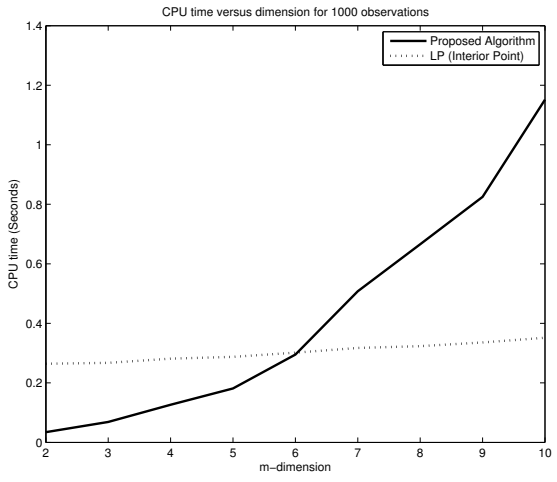

Fig. 3. Showing the CPU time versus the LAV problem dimension for fixed number of observations(1000)

Notice that fADE has strikingly different properties when compared to the LP algorithm. The LP is much less sensitive to the

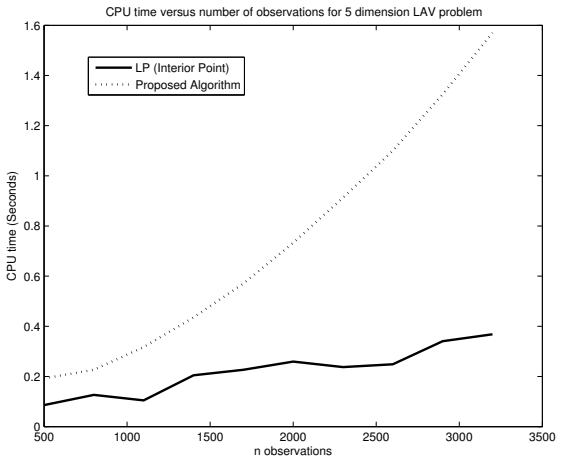

Fig. 4. Showing the CPU time versus the number of observations a fixed number of regression coefficients (5)

dimension of the problem, than fADE. In fact, it is fairly obvious that the time complexity of the LP algorithm rises linearly with m, whereas in case of fADE, it rises exponentially. However, the situation is completely reversed when the number of observations are increased. In this case, fADE is much less sensitive and has a slow and almost linear increase in its time complexity. But the LP solution takes exponentially longer to compute. It is also noteworthy to underline the huge difference in memory requirements. For example, in case of 5 regression coefficients and 2000 observations, the proposed algorithm uses 96Kb of memory, versus 64Mb required for the data matrices of the LP formulation (This is simply the allocation space required for the input data, and does not include the temporary memory required during computation).

Clearly fADE has an overall advantage in terms of memory usage. We also argue that for the application it was developed, it is also superior in terms of computational efficiency. Namely, when estimating in real life industrial applications, one is generally dealing with a few coefficients, but many observations. For example in case of the Base Sheet Ash model which was mentioned earlier, the model only has 4 coefficients (Headbox Ash, Headbox Consistency, Whitewater Ash, Whitewater consistency). In terms of the number of observations, the paper machine's DCS system will log plantwide data every 5 seconds. This translates to over 17280 observations per 24 hours. This will take the LP approach an estimated 2 minutes to compute the LAV model, but only 1.3 seconds using fADE. This is a very significant speed advantage, especially when noting that the Interior Point solver used for the LP problem is already one order of magnitude faster than the Simplex method which itself is reputed to be a fast algorithm. The fact that the LP bogs down when the observations are increased in not altogether a surprise. Recall that in case of the LP formulation, each new observation adds a new equation and plane to the problem. However in case of the proposed direct search approach, a new observation just means an extra multiplication in the cost function evaluation.

The speed of the proposed algorithm will make it possible to implement LAV problems online, in the same way that been traditionally done for many years with LS models. Moreover, the algorithm is very light on memory usage and is able to compute the LAV solution using only elementary operations of addition and multiplication (although a (pseudo) random number generator is required). Although the proposed algorithm is based on an EA, it is important to recall that significant modifications were made to it to increase its speed for the LAV problem. It is therefore important to realize that the proposed algorithm can no longer be considered a generic optimization algorithm in the same way as a standard EA can. It might well be that due to the removal of the crossover operator and the dynamic reduction of the population, the proposed algorithm will no longer be able to solve a multi-modal problem (although it doesn't have to be, since other powerful EA algorithms, such as Evolution Strategies (ES) only use gene mutation).

REFERENCES

[1] S. Narula and J. Wellington, "The minimum sum of abseloute errors regression: A state of the art survey," *International Statistical Review*, vol. 50, pp. 317–326, 1982.

[2] J. Rice and J. White, "Norms for smoothing and estimation," *SIAM Review*, vol. 6, pp. 243–256, 1964.

[3] P. Huber, "Robust regression: Asymptotics, conjectures and monte carlo," *Ann. Statist*, vol. 1, no. 799-821, 1973.

[4] D. Andrews, "A robust method for multiple linear regression," *Technometrics*, vol. 16, pp. 523–531, 1974.

[5] J. Claerbout and F. Muir, "Robust modeling with erratic data," *Geophysics*, vol. 38, no. 5, pp. 826–844, October 1973.

[6] E. Denoel and J. Solvay, "Linear prediction of speech with a least absolute error criterion," *IEEE Transactions on Acoustics, speech and signal processing*, vol. 33, no. 6, pp. 1397–1403, 1985.

[7] C. Eisenhart, *Roger Joseph Boscovitch*, 1961, ch. Bascovitch and the combination of observations.

[8] R. Armstrong and M. Kung, "A dual algorithm to solve linear least absolute value problems," *J. Opi Res. Soc.*, vol. 33, pp. 931–936, 1982.

[9] A. Charnes and H. A. Cooper, W.W. and, *Introduction to Linear Programming*. Wiley, 1953.

[10] A. Charnes and F. R. Cooper, W.W. and, "Optimal estimation of executive compensation by linear programming," *Manage, Sci.*, vol. 1, pp. 138–151, 1955.

[11] R. Gonin and A. Money, *Nonlinear LP-norm estimation*. Marcel Dekker, New York, 1989.

[12] H. Li, "Solve least absolute value regression problems using modified goal programming techniques," *Computers Op. res.*, vol. 25, no. 12, pp. 1137–1143, 1998.

[13] J. Gentle, S. Narula, and V. Sposito, *Statistical Data Analysis Based on the L_1-norm and related methods*. North-Holland, Amsterdam, 1987, ch. Algorithms for unconstrained L_1 linear regression, pp. 83–94.

[14] J. Ignizo, *Introduction to goal programming*. Sage, Newbury Park, California, 1995.

[15] E. Cerezci and F. Gokpinar, "A comparison of three linear programming models for computing least absolute value estimates," *Hacettepe Journal of Mathematics and Statistics*, vol. 34, pp. 95–102, 2005.

[16] A. Nobakhti and H. Wang, "Co-evolutionary self-adaptive differential evolution with a uniform-distribution update rule," *Proceedings of the IEEE CCA/CACSD/ISIC - Munich - Germany*, 2006.

[17] ——, "A simple self-adaptive differential evolution algorithm with application on the alstom gasifier," *Applied soft computing*, vol. 8, no. 1, pp. 350–370, 2007.

[18] R. Storn and K. Price, "A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[19] B. Babu and R. Angira, "A differential evolution approach for global optimization of minlp problems," *Proceedings of 4th Asia Pacific Conference on Simulated*, 2002.

[20] M. Cardoso, R. Salcedo, S. Azevedo, and D. Barbosa, "A simulated annealing approach to the solution of minlp problems," *Computers and Chemical Engineering*, vol. 21, no. 12, pp. 1349–1364, 1997.

[21] Y. Lin, K. Hwang, and F. Wang, "Co-evolutionary hybrid differential evolution for mixed-integer optimization problems," *Eng. Opt.*, vol. 00, pp. 1–20, 2001.

[22] J. Lampinen and I. Zelinka, "Mixed variable non-linear optimization by differential evolution," *Proceedings of Nostradamus*, 1999.

[23] ——, "On stagnation of the differential evolution algorithm," *Proceedings of MENDEL*, 2000.

[24] R. Storn, *Designing digital filters with differential evolution*. McGraw-Hill, London, 1999, ch. In Corne, D., Dorigo, M., and Glover, F. (editors), New Ideas in Optimization, pp. 109–125.

[25] R. Gamperle, S. Muller, and P. Koumoutsakos, *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*. WSEAS Press, 2002, ch. A Parameter Study for Differential Evolution, pp. 293–298, a. Grmela, N.E. Mastorakis (eds).

[26] J. L. Lampinen, "Adaptive parameter control of differential evolution," *R. Matouek, P. Omera (eds.), Proc. of Mendel 2002, 8th Internat. Conference on Soft Computing, pp. 19-26*, 2002.

[27] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," *R. Matouek, P. Omera (eds.), Proc. of Mendel 2003, 9th Internat. Conference on Soft Computing, Brno, pp. 41-46*, 2003.

[28] H. Abbass, "The self-adaptive pareto differential evolution algorithm," *Proc. 2002 congress on Evolutionary Computation*, pp. 831–836, 2002.