Increased OPF code development efficiency by integration of general purpose optimization and derivative computation tools

Tina Orfanogianni
Tina.Orfanogianni@eeh.ee.ethz.ch

Rainer Bacher
Rainer.Bacher@eeh.ee.ethz.ch

Swiss Federal Institute of Technology (ETH), CH 8092 Zurich, Switzerland

Abstract This paper describes a new approach for solving OPF problems, using a combination of "black-box" tools for optimization and automatic code-generation for first order derivatives. The efficiency of the new code will be evaluated in terms of maintainability, flexibility and speed. These code properties are relevant and useful considering the increasing need for easy-to-handle software tools, to integrate new models and objectives derived from the new open energy market environment, with only standardized, minimal changes for the software developer.

Keywords - OPF, optimal power flow, optimization, automatic differentiation, MINOS, ADIFOR

1 Introduction

Economic operation of power systems has traditionally been achieved with the support of the *Optimal Power Flow* (OPF) program. This is an optimization problem of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & F(x) \\ \text{subject to} \quad & g(x) = 0 \\ & h(x) \leq 0 \\ & l \leq x \leq u \end{aligned}$$

$F(x)$ is the objective function representing the mathematical formulation of the goal, that usually is minimum generation cost (*economic dispatch*) and minimum loss operation. The equations $g_i(x) = 0$ and the inequalities $h_i(x) \leq 0$ and $l \leq x_i \leq u$ form the set of constraints and represent physical limitations as well as security and reliability restrictions. The constraints can be *network* and *modeling* constraints (Kirchhoff law or power mismatch equations at the nodes and modeling of network components) as well as *operational* constraints (upper and lower limits on: voltage magnitude, active and reactive power generation, tap magnitude and tap angle, current or power flow of the lines). Constraints of the first type are usually equalities, while constraints of the second type are inequalities, or take the form of bounded variables. The variables of the optimization problem are identified as *control* variables or *state* variables. Control variables represent quantities that can be influenced by the operator (power generation, tap settings) while state variables *describe the state* and they usually come as part of the solution of the OPF (line currents, load voltages). Quantities that are constant throughout the problem are *parameters* (loads, line impedances etc.). The role of the type of each variable is essential here, and we will see how quantities change type. A parameter that in a version of the OPF becomes a variable increases the degree of freedom of the optimization problem.

The OPF is in general a nonlinear optimization problem with nonlinearities both in $F(x)$ and $g(x)$, $h(x)$. The exact formulation of the problem and the selection of variables will determine the degree of nonlinearity, while the number of constraints can be very large even for medium size networks. The mathematical challenge does not stop here. Every constraint is a function of only a small subset of the variable set (partially separable functions), therefore there is a large benefit to consider the problem as *sparse*. For these reasons the **OPF** is solved as a **large-scale sparse nonlinear optimization problem**. The solution process usually involves solving the optimality (Karush-Kuhn-Tucker) conditions for nonlinear constrained optimization that contain first and second derivatives of the objective and constraint functions. A number of algorithms (augmented Lagrangian, sequential quadratic programming, reduced gradient) can be applied for its solution. If Jacobian and Hessian matrices of constraints and gradient and Hessian of objective are hand-coded then the resulting code can be fast (execution time). However the insertion of a new feature from the software programmers point of view is not trivial. Correct and exact programming requires keeping track of the interdependence of variables and having thorough knowledge of the sparsity structure of the derivatives. Calculation as well as positioning of the Jacobian terms while adding new variables or new equations, is a particularly error-prone programming task.

As mentioned, the objectives of the traditional OPF have been minimum production cost and minimum-loss operation. With the open energy market environment the motivations have changed. Minimum-loss and secure operation of networks is still desired, but the focus of planning has been shifted to more economic, flexible and controllable operation. Maximum power transfer between regions, elimination of loop flows or setting of the active power flow along desired paths are some examples of new possible goals under the new conditions. At the same time FACTS controllers allow more flexibility in the power grid by providing smooth and rapid control over certain network elements. This flexibility amounts to an increased number of control variables in optimization (series impedance with series FACTS devices) and less strict constraints (increased line ratings).

Under these conditions there is a motivation in an **academic environment** to create an **optimization software tool** that offers a flexible and easy-to-augment code, where the new features (new device models, adding or elimination of constraints, changing a parameter to a variable etc.) can be easily inserted. Ideally the user should be relieved from the tiring task of reconsidering the sparsity structure of matrices (dependent on the order of variables and equations) every time a new feature is added. The main purpose of this paper is to present an environment for solving OPF problems having the properties described above, by using a combination of a general-purpose optimization package (MINOS) and an automatic code differentiation tool (ADIFOR).

Using both parts separately is not new to the power systems engineering field: A combination of MINOS and the OPF problem has been described by various authors [1, 2, 3, 4, 5]. ADIFOR alone has been applied to power system problems in [6, 7, 8].

In this paper, however, the combination of both MINOS and ADIFOR for various purposes is described. Thus, this paper does not reinvent OPF problem solutions nor does it want to emphasize that MINOS is the fastest optimization environment to solve OPF problems.

The main emphasis of this paper is the efficiency of the presented code in terms of certain **software quality properties** that in the context of this paper are interpreted as follows:

• **OPF problem formulation flexibility:** The ability in an OPF code to switch between different optimization algorithms or experiment with alternative formulations of functions.

• **OPF optimization code maintainability:** The ability to *easily* extend, modify and reuse parts of an existing code (e.g. inserting a new constraint or objective function, change parameters, bounds, variables etc.)

• **OPF optimization code development and program execution speed:** The paper distinguishes between *development* (programming) speed and *execution* (CPU time) speed.

2 MINOS optimization tool

2.1 Standard form and algorithm

MINOS (Modular In-core Nonlinear Optimization System) is a FORTRAN based package designed to solve large-scale optimization problems in continuous variables, expressed in the standard from

$$\min_{x,y} \quad F(x) + c^T x + d^T y \tag{1a}$$

$$\text{s.t.} \quad f(x) + A_1 y = b_1 \quad (m_1 \text{ rows}) \tag{1b}$$

$$A_2 x + A_3 y = b_2 \quad (m_2 \text{ rows}) \tag{1c}$$

$$l \leq \begin{bmatrix} x \\ y \end{bmatrix} \leq u \quad (n+m \text{ variables}) \tag{1d}$$

where the vectors c, d, b_1, b_2, l, u and the matrices A_1, A_2, A_3 are constant (see [9]). All functions of x are considered twice differentiable with bounded Hessians. MINOS distinguishes between nonlinear (1b) and linear (1c) *general constraints* and constraints imposing bounds on variables (1d). Inequalities on general constraints can be rewritten as equalities with the introduction of *logical* (slack) variables with appropriate bounds according to the following principle (assume $g_i(x)$ is a general constraint function):

$$l_i \leq g_i(x) \leq u_i \longmapsto \begin{cases} g_i(x) + s_i = 0 \\ -u_i \leq s_i \leq -l_i \end{cases} \tag{2}$$

Slack variables are introduced also to the equality constraints, which are considered as inequalities with $l_i = u_i$. Problem (1) has m general constraints (m_1 nonlinear and m_2 linear, $m_1 + m_2 = m$), n *structural* and m slack variables. The vectors $x \in \mathcal{R}^{n_1}$ and $y \in \mathcal{R}^{n_2+m}$ contain the variables that are nonlinearly and linearly used (nonlinear and linear variables). The last m components of y form the vector s of slack variables and the last m columns of A_1, A_3 form the identity matrix I.

Depending on the structure of the problem MINOS applies different algorithms.

1. $F(x) = 0, f(x) = 0$: For linear programs MINOS employs a primal simplex method.

2. $f(x) = 0, F(x) \neq 0$: For the solution of a linearly constrained nonlinear optimization problem MINOS employs a reduced-gradient algorithm. This is a feasible point method and can be viewed as a generalized simplex method. The variables are separated in three sets: basic, superbasic and nonbasic variables. MINOS implements a quasi-Newton algorithm for optimizing the superbasic variables, by updating a dense approximation of the reduced hessian and applying a line search algorithm.

3. $f(x) \neq 0, F(x) \neq 0$: In this case MINOS uses a projected augmented Lagrangian method ([10]). A sequence of *major* iterations are performed, in which a linearly constrained subproblem is solved with the reduced gradient approach (minor iterations).

2.2 Interface to the OPF

The authors have used the subroutine version of MINOS. The interface in this case is a simple FORTRAN subroutine call

```
      call  minoss(start, m, n, nb, ne, nname,
     &   nncon, nnobj, nnjac,  iobj, objadd,
     &   names, a, ha, ka, bl, bu, name1, name2,
     &   hs, xn, pi, rc, inform, mincor, ns,
     &   ninf, sinf, obj, z, nwcore)
```

where all problem data is passed as parameters. The most important input parameters are:

start: specifies how a starting point is to be obtained.
m,n: number of constraints and structural variables
ne: number of nonzero entries of the Jacobian matrix of constraints
nncon: number of nonlinear constraints
nnjac, nnobj: number of nonlinearly used variables ("nonlinear variables") in the constraints and objective function
a,ha,ka: the Jacobian matrix stored in compressed sparse column format

The software programmer should note that in a nonlinear problem (as the OPF problem) a certain partitioning of the constraints should take place, so that the nonlinear constraints are the first `nncon` components of the set of constraints (1b). Likewise, the nonlinear variables of the objective (contained in $F(x)$) occupy the first `nnobj` places of the variable vector, followed by the `nnjac` nonlinear variables of the constraints (variables contained in vector $f(x)$) and the rest `n+m-nncon-nnobj` are the linear variables. The partitioning of the constraint and variable set is under the responsibility of the developer and can be a non trivial task, depending on the dimension and complexity of the problem.

For linear problems this is the main input the user should provide. For a nonlinear problem at most two additional user-written subroutines are essential. If nonlinearities in the objective function exist ($F(x) \neq 0$ in (1a)), a FORTRAN subroutine `funobj` will calculate the objective function at the iterate x_k and the gradient $\nabla F|_{x_k} = \{ \frac{\partial F}{\partial x_j}\Big|_{x_k}, j = 1 \ldots nnobj\}$ of the *nonlinear part* with respect to the first *nnobj* nonlinear variables. Likewise, if the constraints have a nonlinear part ($g(x) \neq 0$) in (1b) then a FORTRAN subroutine `funcon` will compute the constraints at a given point x_k and the Jacobian submatrix $\frac{\partial g}{\partial x}\Big|_{x_k}$ of the *nonlinear part* of constraints with respect to the first *nncon* nonlinear variables. MINOS can compute all or some of the derivatives with finite differences, but to achieve efficient code (execution time), the user should provide all derivatives.

To evaluate the relative efficiency of an OPF code using the MINOS package (OPF/MINOS) in comparison to a totally hand-coded OPF (OPF/HC) the following observations can be made:

• The OPF/HC code is more flexible in specifying features in the optimization kernel, while in the OPF/MINOS code the algorithm, features and options are confined to the MINOS capabilities.

• The OPF/MINOS code is easier to extend and to maintain. The power flow and network related formulations are the same in both codes, but in the first case the user is relieved from second order derivative calculations (MINOS approximation) and in an extreme case can omit first order derivative calculation as well. However, in large problems, like the OPF, Jacobian sparse structure and Jacobian terms should be provided.

• Depending on the algorithm chosen in the OPF/HC and the size of the problem the execution speed varies significantly, and it is likely that with a robust and efficient hand-coded optimization kernel the OPF/HC is faster. The development time, however, to code a robust optimization kernel in comparison to the time to build up the interface to MINOS is significantly larger.

3 ADIFOR applied for first order derivatives in optimization

As pointed out before, MINOS expects from the programmer to provide formulas for the first order derivatives, that are contained in the subroutines `funcon` and `funobj`. This is FORTRAN user-written code and usually contains formulas for the calculation of the functions (constraint, objective) and some or all partial derivatives. For every constraint function the user has to supply formulas for all nonzero partial derivatives and the indices pointing to their place in the sparse Jacobian matrix. This is an increasingly difficult task, especially when the code gets larger or if it is maintained by more than one software programmers. It restricts the ease in maintainance of the code since there is more than one code segment relevant to the insertion of a new feature or function. In the case of integrating a new constraint, the necessary steps in a hand-coded OPF are: the formulation of the contraint function, definition of bounds, calculation of partial derivatives and positioning in the Jacobian matrix. In such a structure of interdepending code sections, propagating errors or just failure to update some of the necessary parts result in bugs difficult to detect.

To relieve the software programmer from the task of derivative calculation and building-up of the Jacobian structure the technique of automatic differentiation was used. The software package ADIFOR (Automatic Differentiation for FORTRAN) allows automatic code differentiation for FORTRAN77 programs [11]. The user needs only to supply the source code (standard FORTRAN77 code that can contain `if` statements, `do` loops, common blocks, `implicit none` and `include` statements) and to specify the names of the variables that correspond to the dependent and independent variables. ADIFOR performs

a dependency analysis and determines the *active* variables, i.e. the variables that have an associating derivative object [12]. Assume a function f with an $n-$vector $\boldsymbol{x}$ as independent and an $m-$vector $\boldsymbol{y}$ as dependent variables. The ADIFOR-generated code will compute

$$\boldsymbol{g\_y} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \times \boldsymbol{g\_x}^T \right)^T = (\boldsymbol{J} * \boldsymbol{S})^T \tag{3}$$

where $\boldsymbol{J}$ is the Jacobian matrix and $\boldsymbol{S}$ is the *seed matrix* ([11, 8]). By proper initialization of $\boldsymbol{S}$ all elements of the transposed Jacobian or the transpose of a Jacobian-vector product can be obtained. The autogenerated code is an augmentation of the original source code, which must be available in subroutine form (FORTRAN77). A call to the generated subroutine evaluates *both* the function value *and* the value of derivatives (a call to the original hand-coded subroutine is not needed).

A major advantage of ADIFOR that renders it ideal for power system problems is the *sparsity handling*. When linked with a given C-language library, called SparsLinC, nonzero entries of the Jacobian matrix are detected and calculated during code derivation time. By use of pointers to the sparse representation of the variables the Jacobian matrix can be extracted in a compressed sparse row format.

The authors have implemented a power-flow code, that uses ADIFOR-generated code for the calculation of the Jacobian in every Newton-Raphson iteration. The reader is referred to [8] for a detailed description of the advantages and features of the power flow code. In this environment the source code (FORTRAN77 subroutine `power_flow`) contains the power flow equations (network modeling equations, Kirchhoff current or power equations) and several features (fix area MW export, regulating transformers, ability to switch between polar and rectangular formulation etc.) that all can be applied to the optimization. The equality constraints of the OPF are the power flow equations, so that the existing source code could be used with only minor modifications as the subroutine `constraints.f`. This FORTRAN77 subroutine contains the mathematical formulations of all constraint functions *and* the objective function.

The subroutine call to the ADIFOR generated code

```
call g_constraints
```

and a subsequent call to the extraction routine of SparsLinC will give: the objective value, its gradient vector, the constraints values and the sparse jacobian values and structure in compressed sparse row format. A format converter was used to convert it to the compressed sparse column format that MINOS needs. An additional part is necessary, since MINOS expects from `funcon` and `funobj` the calculation of the *nonlinear* terms: a for-loop over all calculated derivatives will substract the linear part.

At this point a drawback of the combination of the two packages should be noted. MINOS wants the calculation of the objective and constraint (as well as their derivatives) in different subroutines, `funcon` and `funobj`. However, ADIFOR cannot separate the generated code in two independent subroutines: the interdependence path and application of the chain rule for derivative calculations are parts of the automatic differentiation technique and cannot be detached. This amounts to an unnecessary overload, since every time MINOS calls `funobj` the whole Jacobian is built anew before calculating the objective and its gradient.

For these reasons this optimization tool is slower than a similar tool with hand-coded derivatives in terms of executing (CPU) time. However the flexibility and modularity that has been achieved translates in programming-time speed. To give an insight of the properties of ADIFOR code, the following extract of the `constraints` subroutine (the FORTRAN subroutine that the user has to hand code) is studied in more detail.

```
   do i = 1, ntie_data
          ktie = tie_br_indx(i)
c   The flow on the tie line
c-----------------------------
       if (power_mismatch) then
          Ptie = P_2p_1(ktie)
       else
          Ptie = e_2p_1(ktie) * I_2p_1_e(ktie) +
  &            f_2p_1(ktie) * I_2p_1_f(ktie)
       endif
c  Add the tie flow to the appropriate area
c-----------------------------------------
       SPexp_area(area_index(tie_metered_areano(i))) =
  &      SPexp_area(area_index(tie_metered_areano(i))) +
  &      Ptie

       SPexp_area(area_index(tie_nonmet_areano(i))) =
  &      SPexp_area(area_index(tie_nonmet_areano(i))) -
  &      Ptie
    enddo
```

This do-loop calculates the MW export of all areas by adding the flows on the tie lines. `e_2p_1,f_2p_1` are the vectors of real and imaginary part of the voltage (in rectangular coordinates) of terminal 1 of lines (`_2p_` stands for the two-port model of lines), `I_2p_1_e, I_2p_1_f` the vectors of real and imaginary part of the current of terminal 1 of lines, `P_2p_1` the vector of real line flow (MW) measured on terminal 1, and `SPexparea` the vector of total MW export of areas. If the option `power_mismatch` is activated (use of the power-mismatch and not current-mismatch equations) then `P_2p_1` is independent variable, otherwise the vector `I_2p_1_e` is independent variable.

A constraint or objective function can be formulated, by means of the intermediate (neither independent nor end-dependent) variable vector `SPexp_area`. An example of a constraint function would be to force the import of an area `iarea`, specified by the user, to a fixed value by inserting the constraint:

```
NLConstraintVector(nlcon) = - SPexp_area(iarea)
NLbu(nlcon) = -ImportValue
NLbl(nlcon) = -ImportValue
```

An objective function could be to minimize or maximize the MW export of area `iarea`.

```
c    if maximize export of area iarea
    FObjective = - SPexp_area(iarea)
c    if minimize export of area iarea
    FObjective =  SPexp_area(iarea)
```

The dependence path of the variable `SPexp_area` back to the independent variables is generated and updated during ADIFOR code generation and derivative information is propagated with the chain rule. This capability allows great flexibility in defining complex formulations as simple functions of intermediate variables. In the following section we will see how a new objective can be inserted with minimal *standardized* steps in certain parts of the code.

4 Adding features and modifying the structure

In this section we will describe the process of inserting a new constraint and an objective function in the OPF code with MINOS optimization package and an automatic differentiation tool.

4.1 Standardized steps to include a new constraint

The following FORTRAN code extract is from the `constraints` subroutine and adds a new constraint posing an upper limit on the MVA flow of lines (the line MVA rating).

```
   if (flow_ratings) then
      do i = 1, num_2p_data
         NLConstraintVector(nlcon) =
&           P_2p_1(i)**2 + Q_2p_1(i)**2
         NLbu(nlcon) = 0.0
         NLbl(nlcon) = -branch_MVA_rating_1(i)**2
         nlcon = nlcon + 1
      enddo
   endif
```

We can distinguish the following steps:

1. Mathematical formulation of the constraint as a function of independent or/and intermediate variables. In this case the variable vectors `P_2p_1, Q_2p_1` are intermediate variables that have been formulated as a function of independent variables higher up in the code.

2. Depending on the degree of the nonlinearity of the function storage to the corresponding vector. The vectors `NLConstraintVector` and `LNConstraintVector`, for nonlinear and linear constraints, are locally defined and at the end of the code are copied in the right order to the globally defined variable vector `x` (the input parameter to `minoss`).

3. Specification of the upper and lower limits according to (2). The vectors `NLbu,NLbl,LNbu,LNbl` are copied in the right order to the input parameters `bu, bl`.

4. Automatic calculation of first order derivatives by execution of ADIFOR for `constraints` subroutine and recompilation with the newly generated code.

Inequality constraints posing bounds on independent variables are formulated during initialization of the ADIFOR-seed matrix (see section 4.2).

4.2 Standardized steps to include a new independent variable

Inserting a new variable is a relatively easy procedure. The following do-loop introduces the variable `X_l` (line reactance) in the seed matrix (independent variable), using the `dspsd` SparsLinC subroutine, and specifies upper and lower bounds.

```
do i = 1, num_lines
   call dspsd(g_X_l(i),ipos,1.d0,1)
   bu(ipos) = X_max(i)
   bl(ipos) = X_min(i)
   ipos = ipos + 1
enddo
```

Observe that `X_l` is a constant parameter in conventional power system calculations and by considering it a variable, the degree of freedom of the optimization is increased. Observe that this is relevant (although a simplified model) to the series FACTS devices for control of power flow along a transmission path. From the programmer's point of view the new partial derivatives of the objective and *all* constraint functions have to be recalculated and may not be trivial to obtain. With the OPF/MINOS/AD code (OPF/MINOS approach with derivatives calculated with the automatic differentiation technique) the user does not have to care about Jacobian calculations. All modifications needed are restricted in the seed matrix initialization file:

1. Definition and initialization of the new variable in the seed matrix (using the appropriate SparsLinC library call).

2. Specification of bounds of variable.

3. Storage in vector `x` of the linear and nonlinear variables.

4. Execution of ADIFOR to update the seed matrix and regenerate the derivative code. Compilation of the newly generated code.

The developer should pay attention when storing the variable in vector `x` (Step 3. above). The separation of variables in linear/nonlinear is not so trivial as in the case of general constraints. A variable which is linearly used with the existing constraints could be nonlinearly used in a newly inserted constraint. As already mentioned, this is under the responsibility of the developer, and it is the only task restricting the modularity of the whole process. However, from experience in power system problems, the high nonlinearity in classical compact formulations (with voltages and reactive generation as the main independent

variables) relieves the user from this responsibility, since he/she deals with exclusively nonlinear equations and variables.

4.3 Standardized steps to include a new objective

In the following, a number of existing objectives will be listed to provide a better insight into the modularity and flexibility of the OPF software tool. The code extracts are simplified for illustration purposes. All additions and modifications are again restricted to one file, the main subroutine `constraints`.

• Losses of areas

```
if (LossMin) then
   if (NetworkLoss) then
      FObjective = SPsysLoss
   elseif (AreaLoss) then
      do i = 1, HowManyAreasMin
         FObjective = FObjective +
&              SPLoss_area(AreaLossMin(i))
      enddo
   endif
endif
```

The user has the option to choose between minimization of losses of the whole network or of specific areas. The corresponding variables `SPsysLoss` and `SPLoss_area` have been defined and formulated higher up in the code.

• Power transfer between areas

```
if (mw_transfer_between_areas) then
   if (maxmin .eq. 'max') then
     FObjective = FObjective - SeTieAreaFrom(tiearea)
   elseif (maxmin .eq. 'min') then
     FObjective = FObjective + SeTieAreaFrom(tiearea)
   endif
endif
```

In this case we wish to optimize the MW power transfer between two user-specified areas. The variable `tiearea` denotes the group of the tie lines that connect these two areas and `SeTieAreaFrom` is the sum of real power flowing on the tie lines measured at area `AreaFrom`. Keeping in mind that MINOS by default minimizes, we set the appropriate sign on the variable `SeTieAreaFrom(tiearea)` and store it to `FObjective`. This part has been easily extended to include optimization of reactive power flow and optimization of more than one pair of areas.

• Total power import/export of area

```
if (mw_area_export) then
   if (maxmin .eq. 'max') then
      FObjective = FObjective - SPexpArea(iarea)
   elseif (maxmin .eq. 'min') then
      FObjective = FObjective + SPexpArea(iarea)
   endif
endif
```

By specifying one of the above options (`LossMin`, `mw_transfer_between_areas`, `mw_area_export`) as true, one of the above function is activated. Multiobjective optimization can be achieved by activating more than one option, whereas all individual functions are accumulated in `FObjective`.

Considering the above examples it is easy to assess the benefits of the OPF/MINOS/AD development approach:

• The OPF/MINOS/AD approach is more flexible in alternative formulations of functions (e.g. one may switch from current to power mismatch equations, or from polar- to rectangular- coordinate formulations).

• The largest benefit in the OPF/MINOS/AD code is that it is easy to extend and to maintain. Including a new feature requires changes in certain code sections irrespective of the problem structure.

• For these reasons development speed increases significantly in comparison to the OPF/MINOS or OPF/HC codes. Execution speed suffers mainly because of the approximation of second order derivatives and the unnecessary rebuilding of the Jacobian with every call of `funobj`.

5 Conclusions

This paper presents the benefits and drawbacks of a) developing and b) solving the OPF problem using a general purpose optimization tool and a black-box automatic differentiation tool. The general question when implementing an OPF is: Will the power system OPF developing community get rid of dedicated optimization development or not ? From the OPF program execution point of view, hand-coding and using the latest state-of-the-art sparsity techniques will always result in the fastest code (execution time). Users of production grade OPF programs have traditionally been mainly interested in high execution speed. Computer hardware speed was such that there was a high needs to implement efficiency in the software in order to get the necessary speed. However, also due to the appearance of electric power markets, new ownerships and responsibilities of independent-system-operators and associated changing objectives there is also higher end-user interest in OPF code which can be adapted quickly to new problem formulations and to a wider set of electric power system elements such as FACTS. Traditional OPF code development teams which have put highest emphasis in fine-tuning OPF code towards highest efficiency, have to make large investments to satisfy end-user requirements related to the new fast implementation of new objective functions, new network model characteristics and new functional inequality constraints.

The presented approach has distinctly different qualitities related to these problems: The properties of this OPF software tool are evaluated in terms of flexibility (FL), maintainability (MN), execution speed (CPU) and development speed (DS), and are compared to the hand-coded OPF code. The following table summarizes the results of the comparison with regard to each criterion: With the OPF/MINOS/AD approach the code tends to be slower in

execution speed, while there is high flexibility and a large saving on maintainance effort and development time.

	FL	MN	CPU	DS
OPF/HC	+	+	+++	+
OPF/MINOS	++	+	++	++
OPF/MINOS/AD	+++	+++	+	+++

Clearly, hand-coded (HC) OPF is best in terms of execution code efficiency.

A combination of OPF and MINOS (OPF/MINOS) alone leads to certain advantages in development speed, since the hand-coding of the optimization algorithm is not necessary any more. Execution speed will be affected in the negative sense, i.e. longer execution times can be expected. Flexibility and maintainability are only marginally improved as compared to hand-coding an OPF.

The approach of this paper (OPF/MINOS/AD) clearly improves the development problems of a hand-coded approach: Algorithmic flexibility (FL) is increased together with maintainability (MN) of the resulting code. This last point is clearly justified because the code complexity for the OPF problem is limited to coding the OPF problem formulation itself. No dependent code for any derivative terms need to be hand-coded which leads to much faster development speed (DS) for any functional additions to the code. Program execution time decreases when compared to hand-coded OPF programs: For large networks we noticed a factor of about 5-10 compared to highly efficient OPF programs. This decrease of speed comes mainly from the fact that MINOS requires separate calls to first order derivatives for both the non-linear functional constraints (both equalities and inequalities) and the objective function. Since the objective function depends on intermediate variables computed in the code parts belonging to the functional constraints, certain compute-intensive parts are computed twice whenever MINOS calls the objective function part. These problems could be avoided, if more hand-coding is applied to the OPF/MINOS/AD approach. In this paper, hand-coding of code dependencies was "not allowed" due to the known problems in later code enhancements and maintenance.

Future work will concentrate on improving execution speed without loosing the strenghts of the chosen approach in terms of flexibility, maintainability and development speed.

References

[1] J. De La Fuente and J. Lumbreras. *A new implementation of an optimal power flow system based on a general purpose nonlinear programming program*. IEEE Power Industry Computer Applications Conference, pages 422–428, 1987.

[2] M. Muchayi and M.E. El-Hawary. *Wheeling rates evaluation using optimal power flows*. Conference Proceeding of IEEE Canadian Conference on Electrical and Computer Engineering, pages 389–392, 1998.

[3] D. Chattopadhyay, K. Bhattacharya, and J. Parikh. *Optimal reactive power planning and its spot-pricing: an integrated approach*. IEEE-Transactions-on-Power-Systems, pages 2014–2020, 1995.

[4] R.A. Ponrajah and F.D. Galiana. *The minimum cost optimal power flow problem solved via the restart homotopy continuation method*. IEEE-Transactions-on-Power-Systems, pages 129–148, 1989.

[5] R.C. Burchett, H.H. Happ, and K.A. Wirgau. *Large scale optimal power flow*. IEEE-Transactions-on-Power-Apparatus-and-Systems, PAS-101(10):3722–3732, Oct 1982.

[6] A. Ibsais and V. Ajjarapu. *The application of automatic differentiation in the continuation power flow*. Proceedings of the Twenty Sixth Annual North American Power Symposium. Kansas State Univ, Manhattan, KS, USA, pages 329–337, 1994.

[7] A. Ibsais and V. Ajjarapu. *The role of automatic differentiation in power system analysis*. IEEE Transactions on Power Systems, 12(2):592–597, May 1997.

[8] Tina Orfanogianni and Rainer Bacher. *Using Automatic Code Differentiation in Power Flow Algorithms*. IEEE Transactions on Power Systems, 14(1), Feb 1999.

[9] Bruce A. Murtagh and Michel A. Saunders. *MINOS 5.4 User's Guide (preliminary)*. Technical report, Departement of Operations Research Stanford University, 1983. Revised Jan. 1987, Mar. 1993.

[10] Bruce A. Murtagh and Michel A. Saunders. *A Projected Lagrangian Algorithm And Its Implementation For Sparse Nonlinear Constraints*. Mathematical Programming Study, 16:84–117, March 1982.

[11] Christian Bischof, Alan Carle, Paul Hovland, Peyvand Khademi, and Andrew Mauer. *ADIFOR 2.0 User's Guide*. Technical Report ANL/MCS-TM-192, Mathematics and Computer Science Division, Argonne National Laboratory, http://www.mcs.anl.gov/adifor/, August 1995.

[12] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. *The ADIFOR 2.0 System for the Automatic Differentiation of FORTRAN 77 Programs*. Technical Report ANL/MCS-P481-1194, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

Tina Orfanogianni received the Dipl.El.-Eng. degree in electrical engineering in 1995 from the National Technical University of Athens (NTUA), Greece. She is currently doing her PhD in the area of power system optimization using FACTS devices. She can be reached at Internet e-mail: Tina.Orfanogianni@eeh.ee.ethz.ch

Rainer Bacher received the Dipl.El.-Ing. degree in electrical engineering in 1982 and the Dr.sc.techn. degree in 1986, both from the Swiss Federal Institute of Technology (ETH) in Zürich, Switzerland. In 1993 he was appointed assistant professor of Energy Management Systems at the department of electrical engineering at the ETH Zürich. He can be reached at Internet e-mail: Rainer.Bacher@eeh.ee.ethz.ch