# A New Application Programming Interface and a Fortran-like Modeling Language for Evaluating Functions and Specifying Optimization Problems at Runtime

Fuchun Huang
School of Engineering and Science, Victoria University
Melbourne, Australia

***Abstract*—A new application programming interface for evaluating functions and specifying optimization problems at runtime has been developed. The new interface, named FEFAR, uses a simple language named LEFAR. Compared with other modeling languages such as AMPL or OSil, LEFAR is Fortran-like hence easy to learn and use, in particular for Fortran programmers. FEFAR itself is a Fortran subroutine hence easy to be linked to user's main programs in Fortran language. With FEFAR a developer of optimization solver can provide pre-compiled, self-executable and directly usable software products. FEFAR and LEFAR are already used in some optimization solvers and should be a useful addition to the toolbox of programmers who develop solvers of optimization problems of any type including constrained/unconstrained, linear/nonlinear, smooth/nonsmooth optimization.**

***Keywords-programming language; Fortran computing language; Fortran subroutine; Application programming interface; Runtime function evaluation; Mathematical programming; Optimization problem; Optimization modeling language.***

## I. INTRODUCTION

In this paper we introduce to readers a new application programming interface for evaluating functions and specifying optimization problems at runtime. FEFAR is a Fortran subroutine For Evaluating Functions At Runtime. It can be linked to a programmer's main program to provide a way to evaluate a function or solve an optimization problem at runtime. The functions or optimization problems must be written in a new language called LEFAR I developed together with FEFAR, for evaluating functions or specifying optimization problems at runtime. LEFAR is similar to but much simpler than Fortran, and that is a big merit as it is easy to learn and use and yet powerful enough and complex enough to be able to specify any functions and optimization problems at runtime. FEFAR and LEFAR are already used in USsolver [1] and UNsolver [2], two binary machine code programs for solving unconstrained smooth or nonsmooth optimization problems.

This paper is organized as following. Section II gives the background and related works, section III explains the FEFAR interfaces and parameters, and give some examples, section IV explains the rules of the LEFAR language with comparison to Fortran language, and section V gives some other resources and a near future work scope.

## II. BACKGROUND AND RELATED WORKS

Advanced computing languages such as Fortran [3] and C are compiled language [4]. Unlike interpreted languages [5] such as the S language [6] in SPLUS and R, and the MATLAB language [7], compiled languages must first compile the main program and all other subroutines and functions into a binary machine code program. Programmers of such compiled languages often want to be able to evaluate functions at runtime after the source code of the program has been compiled into a binary machine code program. They may also want to be able to specify/describe optimization problems at runtime by using a modeling language similar to, or, ideally the same, advanced computing language they use, such as Fortran. They may want to keep the source codes of their programs to themselves for commercial reasons but still want others to be able to use or test their software products by giving them a self-executable, directly usable binary machine code program. On the user's end, the binary machine code programs are self-executable and usable immediately; hence the users, in particular the ordinary users but not programmers, are eased from the troubles of finding or purchasing a compiler and compiling the source code programs into binary machine code programs. There are some available modeling languages such as AMPL [8] and OSiL [9] for modeling and specifying optimization problems at runtime though, it is still better to have another modeling language and application programming interface that meet the abovementioned needs better, and that is why I have developed FEFAR and LEFAR, a new application programming interface and modeling language for evaluating functions and specifying optimization problems at runtime, in particular for Fortran programmers. FEFAR is a Fortran subroutine For Evaluating Functions At Runtime. It can be linked to a programmer's main program to provide a way to evaluate a function or solve an optimization problem at runtime. The functions or optimization problems must be written in a new language called LEFAR I developed together with FEFAR, for evaluating functions or specifying optimization problems at runtime. LEFAR is similar

to but much simpler than Fortran, and that is a big merit as it is easy to learn and use. Another reason I made it simple is to shorten the processing and running time of LEFAR codes at runtime, subject to yet being powerful enough and complex enough to be able to specify any functions and optimization problems.

## III. FEFAR INTERFACES

In the following we use gfortran [12] to illustrate how to use FEFAR, but other compilers work as well. There are several interfaces of FEFAR: FEFAR1.obj, FEFAR2.obj, FEFAR3.obj, etc. The following is a simple test program of FEFAR1.obj linked to the main program at compiling and linking stage.

```
>type FEFARtest.f90
program FEFARtest
real*8 :: f
real*8,dimension(1000) :: b
integer*4 :: k
do;
call FEFAR1(1,b,k,f);
call FEFAR1(2,b,k,f);
call FEFAR1(3,b,k,f);
end do;
end program
>
>gfortran FEFARtest.f90 FEFAR.obj
>a.exe
Input the file name of the function:
rosenbrock.far
    f=          24.1999999999999957
   x1=          -1.2000000000000000
   x2=           1.0000000000000000
>
```

The file "rosenbrock.far" is written in LEFAR language to evaluate the following Rosenbrock function [13] at given x values:

$$F(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$
$$\overline{x}_1 = -1.2, \qquad \overline{x}_2 = 1.0.$$

LEFAR code (that is, the content of the file rosenbrock.f95) is the following:

```
function: Rosenbrock
real                :: f
real, dimension(2) :: b
integer             :: if123

if(if123==1)
   b(1)=-1.2
   b(2)=1.0
end if

if(if123<=2)
f=100.0*(b(2)-b(1)**2)**2+(1.0-b(1))**2
end if

if(if123==3)
   print,"f=",f
   print,"x1=",b(1)
   print,"x2=",b(2)
end if
end function
```

FEFAR1 is a Fortran subroutine of the following structure:

```
subroutine FEFAR1(if123,b,k,f)
integer*4 :: if123
real*8, dimension(1000) :: b
integer*4 :: k
real*8 :: f
…
end subroutine FEFAR1
```

FEFAR2 is a Fortran subroutine of the following structure:

```
subroutine FEFAR1(if123,b,k,f,g)
integer*4 :: if123
real*8, dimension(1000) :: b,g
integer*4 :: k
real*8 :: f
…
end subroutine FEFAR1
```

FEFAR3 is a Fortran subroutine of the following structure:

```
subroutine FEFAR1(if123,b,k,f,kg,g)
integer*4 :: if123
real*8, dimension(1000) :: b,g
integer*4 :: k
real*8 :: f
integer*4 :: kg
…
end subroutine FEFAR1
```

In calling each FEFAR, a file name is prompted to be inputted at runtime. The value of the integer variable "if123" will be passed to the first integer variable in the runtime file to control which statements in the runtime to be executed or not. For example, initial value assignment statements only need to be executed once, and most other statements need to be executed each time the subroutine is called. Another example is "if123==3" can be used for only displaying values without executing many other statements. Of course, the integer variable "if123" can take any integer value for more complex controls. The argument "b" is a real data type array of dimension 1 for getting initial values of the function from the source code at runtime or setting from the main program the next step *x* values of the function for solving an optimization problem . The argument "f" is a real variable of the value of the function. In FEFAR2 and FEFAR3 argument "g" is a real data type array of dimension 1 for getting array values of dimension 1 returned from the runtime function, with the only difference being that FEFAR3 returns an integer "kg" for the actual number of array values calculated and returned from the runtime function file which are to be used by the main program

while FEFAR2 assumes implicitly kg=k. Argument "g" can be used to return gradients of a differentiable function, for example. It can also be used to return values of constraints.

## IV. RULES OF LEFAR LANGUAGE

LEFAR is a very simple language similar to but far simpler than Fortran language, so in many cases below we just give the Fortran equivalence of most LEFAR statements.

### A. Data types

There are only three data types: real, integer, and logical, and they are equivalent to Fortran's real(len=8) (or equivalently real*8), integer(len=4) (or equivalently integer*4) and logical(len=2) (or equivalently logical*2).

### B. Constants

Real constants are specified like 2.0, 2.1, -2.0, where the decimal symbol '.' is necessary, so is the '0' following the '.' even if there are no other decimal digits. Scientific notations such as 2.0D-1 and 2.0E-1 are not used in LEFAR. The following are not allowed: 2., .1, -2., -.3, 2.0D-1. Correct ways are: 2.0, 0.1, -2.0, -0.3, 0.2. Logical constants are .true. and .false., like in Fortran. The mathematical constant π (sometimes written as pi or PI) which is the ratio of any circle's circumference to its diameter is .PI. in LEFAR. For example, *y=sin(.PI./2)* assigns value 1.0 to *y*.

### C. Intrinsic functions

Intrinsic functions in LEFAR have the same rules as those in Fortran. Currently implemented functions are: *abs(), exp(), log(), log10(), cos(), acos(), sin(), asin(), tan(), atan(), max(), min(), sqrt(), dble(),*and *int().* Depending on demands other functions can be easily added to LEFAR.

### D. Arrays

Arrays are specified and used the same way as in Fortran.

### E. Operators, mathematical expressions and the assignment

All operators in Fortran work the same way in LEFAR. The assignment and mathematical expressions have the same rules as Fortran. For example, x(2,1)=2*(3.4+5)**2-5*x(1,2)**2 is valid in LEFAR and evaluated the same way as in Fortran. Additionally, "^" is also used for exponentiation, the same as "**".

### F. Do loop

There is only one construct of do looping:

```
do while(expr)
⋮
end do
```

which is equivalent to Fortran's DO WHILE(expr) … END DO construct.

### G. If construct

IF-THEN constructs are

```
if(expr)then
 ⋮
end if

if(expr)then
⋮
else
⋮
end if

if(expr)then
⋮
else if(expr)then
⋮
else if(expr)then
⋮
⋮
else
⋮
end if
```

They are equivalent to Fortran's IF-THEN constructs. In LEFAR, however, the word "then" is not necessary, hence the following are also valid:

```
if(expr)
 ⋮
end if

if(expr)
⋮
else
⋮
end if

if(expr)
⋮
else if(expr)
⋮
else if(expr)
⋮
⋮
else
⋮
end if
```

### H. data-end data construct

```
data(x)
⋮
end data
```

by which listed values between are read into x starting from the most right array index then the second array index from the right until the first array index from the left hand side.

*I. datafile-end datafile construct*

```
datafile(x)
'filename'
end datafile
```

by which values in the file 'filename' (a path can be included) are read into x starting from the most right array index then the second array index from the right until the first array index from the left hand side.

*J. "print" statement*

'print' outputs values to the monitor screen. For example,

```
print, "i, b(i):", i, b(i)
```

is similar to the following Fortran statement:

```
print*,'i, b(i):',i, b(i)
```

In 'print' statement, character strings must be put in double quotation marks "…", not single quotation marks.

*K. Other statements and rules of LEFAR*

- 'exit', 'cycle', 'stop', 'return' statements work the same as in Fortran.
- LEFAR statements use lower-case letter only. For example, 'print', but not 'PRINT'.
- LEFAR variables are case-sensitive.
- A LEFAR statement line can be up to 200 characters long.
- Like in Fortran, Any line starting with '!' is treated as a comment line.
- A function file can have up to 1000 lines.
- There is no way to continue a one-line statement (not a construct) in another line.
- All variables must be declared. There are no implicit rules. The way to declare variables and arrays are the same as in Fortran 95.

*L. Rules of the runtime function*

Generally the file may have the following structure:

```
function: function_name
real :: fmin
real, dimension(k) :: x
integer   :: if123
[declare other working variables]

if(if123==1)
⋮
end if

if(if123<=2)
⋮
end if

if(if123==3)
⋮
end if
end function
```

where 'fmin' is the value of the function to be returned to the main program, 'x' is the vector input of the function passed to and from the main program, and 'if123' is a working integer variable passed from the main program. **Important**: the first 'real ::' variable is the one to be returned, the first 'real, dimension(k) ::' vector is the input variable of the function, where 'k' is a positive integer like 2, 3, etc., which is the dimension of the function, while the first 'integer ::' variable is a special integer variable come from the main program (that is, the value of the variable is set in the main program and passed to the function for controlling which blocks to be executed). They can use different names such as 'f', 'b', 'iw'. Generally, within the block "if(if123==1) … end if" are statements to be processed only once. For example, 'data … end data' statements, to specify initial values for an optimization problem, etc. Within the block "if(if123<=2) … end if" are statements to be processed repeatedly like in optimization program. Within the block "if(if123==3) … end if" are statements to be processed only once in the final stage. For example, after a minimum x* has been found, it can be used in this block to evaluate values of other variables or functions depending on it. Two other rules are:

- In Fortran, ';' is used to put and separate two statements in one line. In LEFAR, however, there is no way to separate two one-line statements in one line.
- There should not be a ';' nor any other separator at the end of any statement.

A side note of the above two rules, they make the processing and running time of the codes shorter.

## V. RESOURCES AND FUTURE WORK

More codes, examples and future work are available at http://sites.google.com/site/SoftSome. Future work may include a C computing language version of FEFAR for easy linking of optimization solver programs in C language to FEFAR.

## ACKNOWLEDGMENT

The author thanks Professor Yosihiko Ogata, Professor Satoshi Ito, Professor and Director Tomoyuki Higuchi and The Institute of Statistical Mathematics, Tokyo, Japan, for supporting my research visit to the institute from the 5th of January to the 28th of February in 2012.

## REFERENCES

[1] F. Huang, Some test results of UNsolver: a solver for solving unconstrained non-smooth optimization problems, http://sites.google.com/site/VicSolver, 2012.

[2] F. Huang, Some test results of USsolver: a solver for solving unconstrained smooth optimization problems, http://sites.google.com/site/VicSolver, 2012.

[3] Wikipedia, "Fortran", http://en.wikipedia.org/wiki/Fortran, retrived on April 16, 2012.

[4] Wikipedia, "Compiled language", http://en.wikipedia.org/wiki/Compiled_language, retrived on April 16, 2012.

[5] Wikipedia, "Interpreted language", http://en.wikipedia.org/wiki/Interpreted_language, retrived on April 16, 2012.

[6] Wikipedia, "S (programming language", http://en.wikipedia.org/wiki/S_(programming_language), retrived on April 16, 2012.

[7] Wikipedia, "MATLAB", http://en.wikipedia.org/wiki/MATLAB, retrived on April 16, 2012.

[8] R. Fourer, D.M. Gay, and B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming. Scientific Press, San Francisco, CA, 1993.

[9] R. Fourer, J. Ma and Kipp Martin, "OSiL: An Instance Language for Optimization," Computational Optimization and Applications, Computational Optimization and Applications, 2010.

[10] Wikipedia, "gfortran," http://en.wikipedia.org/wiki/Gfortran, retrived on April 16, 2012.

[11] L. Luksan and J. Vlcek, "Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization," Technical report No. 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2000.

## AUTHORS PROFILE

**Dr Fuchun Huang** is a Senior Lecturer in the School of Engineering and Science at Victoria University, Melbourne, Australia. He was awarded a PhD degree by The Graduate University of Advanced Studies, Tokyo, Japan, and has published papers on computational statistics, in particular Monte Carlo methods, pseudo-likelihood and generalized pseudo-likelihood methods, and developed solver software for solving smooth and nonsmooth optimization problem. He is a member of The Japan Statistical Society.