User's Guide for JAKEFSQP Version 1.0

by P.D. Mathur and A.L. Tits

The Institute for Systems Research is supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

TR 93-37

User's Guide for JAKEFSQP Version 1.0

P. D. Mathur and A. L. Tits

Electrical Engineering Department
and
Institute for Systems Research
University of Maryland, College Park, MD 20742

1 Introduction

JAKEFSQP is a utility program that allows a user of the FSQP constrained optimization routines [1] to invoke the automatic differentiation preprocessor JAKEF [2] with minimal burden. (FSQP can be obtained from André Tits by sending him email at andre@src.umd.edu. JAKEF is available from netlib (e.g., email to netlib@ornl.gov with the subject line: `send index from jakef`).)

During its operation, FSQP requires the evaluation, at a sequence of points, of the various objectives and constraints and of their gradients. In the absence of the JAKEFSQP utility, the user may either supply subroutines that perform gradient computation or request that gradients be computed by finite differences. Manually coding the gradient subroutines is often tedious and prone to mistakes. Finite difference computation is inaccurate and CPU demanding (one function evaluation for each component of the gradient). The ever more popular technique of automatic differentiation (see, e.g., [3]) improves on both counts as it is painless, as accurate as analytical differentiation, and (in its latest implementations) very fast. Automatic Differentiation (AD) packages typically take the form of preprocessors that accept as input (appropriately edited) function evaluation subroutines and produce as output subroutines that compute the corresponding gradients. While the recent past has seen dramatic improvement in the effectiveness of available AD packages, for reasons irrelevant to this discussion, we decided as a first step to make use of JAKEF, a FORTRAN AD preprocessor developed over ten years ago.

Obviously, FSQP users can invoke JAKEF even if JAKEFSQP is not available. Yet, to do this, they have to make arrangements to pass working array definitions to the gradient evaluation subroutines, and appropriately modify the function evaluation subroutines to identify (in JAKEF's format) the independent and dependent variables. The purpose of JAKEFSQP is thus to spare the user such painstaking tasks (and likely associated mistakes) and to entirely automate the procedure. The remainder of this document contains (i) an installation guide, (ii) examples of JAKEFSQP usage, and (iii) a more detailed description of

JAKEFSQP's operation.

2 Installing JAKEFSQP

The JAKEF distribution includes two FORTRAN files: `jakef.f` and `support.f`. The user should store a copy of `support.f` with its main program (`sample`) removed in, e.g., `supp.f`. Files `fsqpd.f` and `qld.f` (as well as the sample problem files) are included in the FSQP distribution. Besides the LaTeX file for this user's manual, the JAKEFSQP distribution consists of 5 files: `jakefsqp.h`, `jakefsqp.c`, `calljkf.f`, `main1.f` and `subs1.f`.

JAKEFSQP calls JAKEF to perform automatic differentiation on the objective and constraint subroutines. This call might be system dependent. Two methods of calling JAKEF are provided in the source code. The user can select which method is used by editing the subroutine `call_jakef()` in the file `jakefsqp.c`.

Method A (preferred): This method can be used when `jakef.f` can be compiled with `jakefsqp.c`. Some users may not be able to link FORTRAN and C object files together using their linker program; these users should use method B. Running JAKEFSQP using method A involves only one executable file, while when using method B, two executables are used. Therefore, method A is faster, less memory demanding and easier to port across directories. To use this method,

- uncomment METHOD A and comment out METHOD B in file `jakefsqp.c` (this is already done in the distribution version of `jakefsqp.c`).
- compile `jakef.f` and `calljkf.f` to object files (say, `jakef.o` and `calljkf.o`) using a FORTRAN compiler, e.g.,

  ```
  f77 -c jakef.f calljkf.f
  ```
- compile `jakefsqp.c` to an object file (say `jakefsqp.o`) using a C compiler, e.g.,

  ```
  cc -c jakefsqp.c
  ```
- finally, link these three object files together using a linker or a compiler and name the resulting executable file `jakefsqp`, e.g.,

  ```
  f77 jakefsqp.o calljkf.o jakef.o -o jakefsqp
  ```

Method B: This method can be used when JAKEF has been compiled and its executable file is available. JAKEFSQP can then call JAKEF through a `system()` call in C. To support this call, the operating system must support input/output redirection (most do). Note that the file `calljkf.f` is not used. To use this method,

- compile `jakef.f` and `support.f` (in fact, only the main program, `sample`, is used from `support.f`) to create the executable JAKEF program, e.g.,

  ```
  f77 jakef.f support.f -o bin/Jakef
  ```

- uncomment METHOD B and comment out METHOD A in file `jakefsqp.c`.

- in `jakefsqp.c`, change the path name of the executable JAKEF program in the `sprintf()` statement so that it points to the JAKEF executable program, e.g.,

  ```
  sprintf(ln,"bin/Jakef < %s > %s",JAKEIN,JAKEOUT);
  ```

  $\underbrace{\texttt{bin/Jakef}}_{\text{(path name)}}$

- compile `jakefsqp.c` to create the `jakefsqp` executable program, e.g.,

  ```
  cc jakefsqp.c -o jakefsqp
  ```

JAKEFSQP uses two temporary files, `JAKEIN.f` and `JAKEOUT.f`. JAKEFSQP formats, and writes into `JAKEIN.f`, a subroutine that has to be processed. This file serves as the input to JAKEF. The output from JAKEF is collected in `JAKEOUT.f`. This cycle is repeated for all the subroutines that have to be processed.

3 Using JAKEFSQP

Supplying the user's entire program to JAKEFSQP is preferred. However, JAKEFSQP can also function if only the function evaluation subroutines are supplied; in this case, the user must modify the remainder of the program manually. Two examples are presented to demonstrate how JAKEFSQP is used in these two cases. Note that, in both cases, the gradient evaluation subroutines included in `sampl1.f` are discarded by JAKEFSQP.

I. The input for the first example is `sampl1.f`, supplied with FSQP. Since JAKEFSQP finds all the information that it needs to process this file, no user interaction is required.

   - Type the command to process `sampl1.f`:

     ```
     jakefsqp sampl1.f sampl1ad.f
     ```

     JAKEFSQP processes the entire file and puts the result in the file `sampl1ad.f`.

   - `sampl1ad.f` can then be compiled along with the rest of the object files:

     ```
     f77 sampl1ad.f fsqpd.o qld.o supp.o -o sampl1
     ```

II. The input for the second example is `sampl1.f`, but with its main program removed and stored in `main1.f` and its subroutines stored in `subs1.f`. Since JAKEFSQP does not find any calls to `fsqpd` and thus cannot figure out the names of the objective and constraint subroutines, it asks the user for help.

- Type the command to process `subs1.f`:

  ```
  jakefsqp subs1.f subs1ad.f
  ```

- JAKEFSQP prompts the user for two subroutine names (sequentially), one for the gradient of the objective subroutine and another for the gradient of the constraint subroutine.
- enter `grcn32` for the first and `grob32` (the case is not important) for the second prompt. Before proceeding, JAKEFSQP asks for confirmation. Press `RETURN` to confirm.
- JAKEFSQP processes the entire file and puts the result in the file `subs1ad.f`. This file can then be compiled along with the rest of the object files:

  ```
  f77 subs1ad.f main1.f fsqpd.o qld.o supp.o -o main1
  ```

Note that if names other that `grob32` and `grcn32` had been given to JAKEFSQP, the `external` statement and the call to `fsqpd` would have to be modified in `main1.f`.

In general, output from JAKEFSQP must be compiled along with the supporting files required by JAKEF and FSQP. If the program contains references to the gradient subroutines in places other than in calls to `fsqpd`, it is the user's responsibility to ensure that these statements now refer to the new gradient subroutines. Since JAKEFSQP tries to maintain the gradient subroutine names the program used earlier, this problem should occur rarely.

4 How JAKEFSQP Works

Based on the portion of the program supplied to it, JAKEFSQP constructs two lists of subroutine names. The first list consists of all names of subroutines in the input file that could possibly be gradient or objective/constraint subroutines. These subroutines are picked by examining the number and types of arguments of each subroutine found in the input file.

The second list is created by examining calls to `fsqpd` in the input (if such calls are not found, this list remains empty). These calls provide information about how the gradient subroutines should be named when they are created by JAKEFSQP.

JAKEFSQP then attempts to find gradient subroutine names for each objective subroutine name in the first list by looking in the second list. If the gradient name happens to be one of FSQP's built-in finite difference subroutines, the new gradient name for the objective function is automatically set to the objective subroutine name prefixed by the character G (if the length of the resulting name is greater than 6 characters, it is truncated to the length of the objective subroutine name). If a match in the second list is not found, the user is asked for the gradient name (entering no name implies that the subroutine under consideration is not a function evaluation subroutine).

After the names of gradient and objective subroutines are resolved, JAKEFSQP copies the code supplied to it to the output file. Wherever it encounters an `fsqpd` call, it makes the necessary modifications to its parameters. Also, to declare all newly created subroutines, `external` declarations are inserted into the output. Function evaluation subroutines are copied to the output, as well as to a temporary file that contains, in addition to the original subroutine, the necessary `construct` statement required by JAKEF. The temporary file is then processed by JAKEF. If JAKEF is successful in generating a gradient evaluation subroutine, its output is appended to the output file. Also, an interface subroutine is inserted into the output file.

FSQP calls the interface subroutine. This subroutine contains all the required working array declarations. The size of these arrays is arbitrarily set to $2000 + 4n$ where n is the number of operators in the objective subroutine. This formula works for all test problems; however, the user might have to change it in certain circumstances. The interface subroutine calls the gradient code produced by JAKEF.

5 Format of Input Files

- JAKEFSQP accepts `TAB` characters and expands them to spaces. The output generated does not contain any comments, extra spaces and indentation.
- The input files must comply with the restrictions placed by JAKEF. In particular, function evaluation subroutines may only call subroutines that are in the FORTRAN library. For details see Section 4 in the JAKEF user's guide.
- JAKEFSQP does not perform rigorous parsing like a compiler does and thus, will fail to detect many syntax errors. To ensure reliable operation, the input file must be compilable using a standard FORTRAN compiler.

6 Notes

- The user may wish to modify the `calljkf.f` file to change working array space for JAKEF.

- Some program parameters may be changed at the top of the files `jakefsqp.c` and `jakefsqp.h`.

References

[1] J. L. Zhou & A. L. Tits, "User's Guide for FSQP Version 3.1 – A Fortran Code for Solving Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying all Inequality and Linear Constraints," Institute for Systems Research, University of Maryland, SRC TR-92-107r2, College Park, MD 20742, 1992.

[2] B. Speelpenning, *Compiling Fast Partial Derivative of Functions Given by Algorithms*, Ph.D. Dissertation, Dept. of Computer Science, University of Illinois, Urbana-Champaign, Illinois, 1980.

[3] A. Griewank & F. Corliss, eds., *Automatic Differentiation of Algorithms: Theory, Implementation, and Applications*, SIAM (Society for Industrial and Applied Mathematics), Philadelphia, PA, 1991.