User's Guide for ADIFFSQP Version 0.9[1]

by M.D. Liu and A.L. Tits

T.R. 97-32

User's Guide for ADIFFSQP Version 0.9[1]

Released March, 1997

Mingyan D. Liu and André L. Tits

Institute for Systems Research
University of Maryland, College Park, MD 20742

1 Introduction

ADIFFSQP is a utility program that allows the user of the FFSQP [1] constrained nonlinear optimization routines to invoke the computational differentiation (or automatic differentiation: AD) preprocessor ADIFOR2.0 [2] conveniently.

When solving optimization problems, FFSQP requires the evaluation of various objectives and constraints, and of their derivatives. The user may either provide his/her own derivative evaluation subroutines or ask FFSQP to use its intrinsic finite differencing option to evaluate the derivatives. Both alternatives have drawbacks. Manually coding the derivative subroutine is often tedious and prone to mistakes. On the other hand, finite differencing is inaccurate and CPU demanding. The rapid development of the technique of automatic differentiation has shown a lot of advantages. Automatic differentiation is as accurate as analytical differentiation. It evaluates partial derivatives accurately and cheaply. In particular, it has been demonstrated that, in reverse mode, the evaluation of a gradient requires no more than five times the effort of evaluating the underlying function alone [3]. Current AD packages, such as ADIFOR2.0, typically take the form of preprocessors that accept function evaluation subroutines as input and generate corresponding derivative evaluation subroutines.

Obviously users can directly invoke ADIFOR2.0 to generate the derivative evaluation subroutine without ADIFFSQP. But to do this, they have to create a "composite file", a "script file", and a main program calling the function evaluation subroutine (Section 2.1), as required by ADIFOR2.0. After the derivative evaluation subroutine is generated by ADIFOR2.0, users still have to implant it into the main program calling FFSQP and create a "driver" subroutine invoking this ADIFOR-generated subroutine, since it cannot be used directly.

Our purpose is to save users the time and trouble of doing the above mentioned work and repeating it every time there is a new problem. ADIFFSQP is thus designed

[1]Copyright © 1997 by Mingyan D. Liu and André L. Tits. All Rights Reserved

to automate the procedure. The user only has to provide a valid input program set in FORTRAN77, as required by FFSQP, with little alteration, and invoke ADIFFSQP. ADIFFSQP takes the input program, generates all the files required by ADIFOR2.0, invokes it, incorporates the ADIFOR-generated subroutines into the original input program, makes essential modifications and outputs a new FORTRAN77 program that can be used directly with FFSQP.

2 ADIFFSQP Architecture

Before we start off to discuss the interrelationship and working procedures of FFSQP, ADIFOR and ADIFFSQP, it is necessary to define some terms which will be frequently mentioned in the following context.

independent variable: By independent variables we mean the components of `x`, the third argument in the argument list of the objective and constraint evaluation subroutines called by FFSQP [1].

dependent variable: The fourth argument, e.g., `gj` or `fj`, in the argument list of the objective and constraint evaluation subroutines called by FFSQP [1].

function evaluation subroutine: A subroutine evaluating the function whose derivative is sought. Sometimes, more than one subroutine may be needed to evaluate a single function. In such cases, there must be a top-level subroutine, as well as one or more lower-level subroutines directly or indirectly called by the top-level subroutine. We call these subroutines a *function evaluation subroutine set.*

derivative evaluation subroutine: A subroutine evaluating the derivative of a corresponding function. This may also require more than one subroutine, in which case there is also a top-level subroutine calling lower level subroutines. These subroutines is called a *derivative evaluation subroutine set.*

Obviously, independent and dependent variables must be defined in both function and derivative evaluation subroutines, at least in the top-level subroutines.

The structure of ADIFFSQP is determined by both ADIFOR and FFSQP. We will discuss how they work in these two subsections.

2.1 ADIFOR 2.0

Figure 1, which we borrowed from [2], is the block diagram illustrating the process of ADIFOR. ADIntrinsic and SparsLinC are two libraries that accompany the ADIFOR package. More detailed descriptions can also be found in [2].

To summarize, a user working with ADIFOR2.0 generally needs to take the following steps:

(1) Provide a function evaluation subroutine (set).

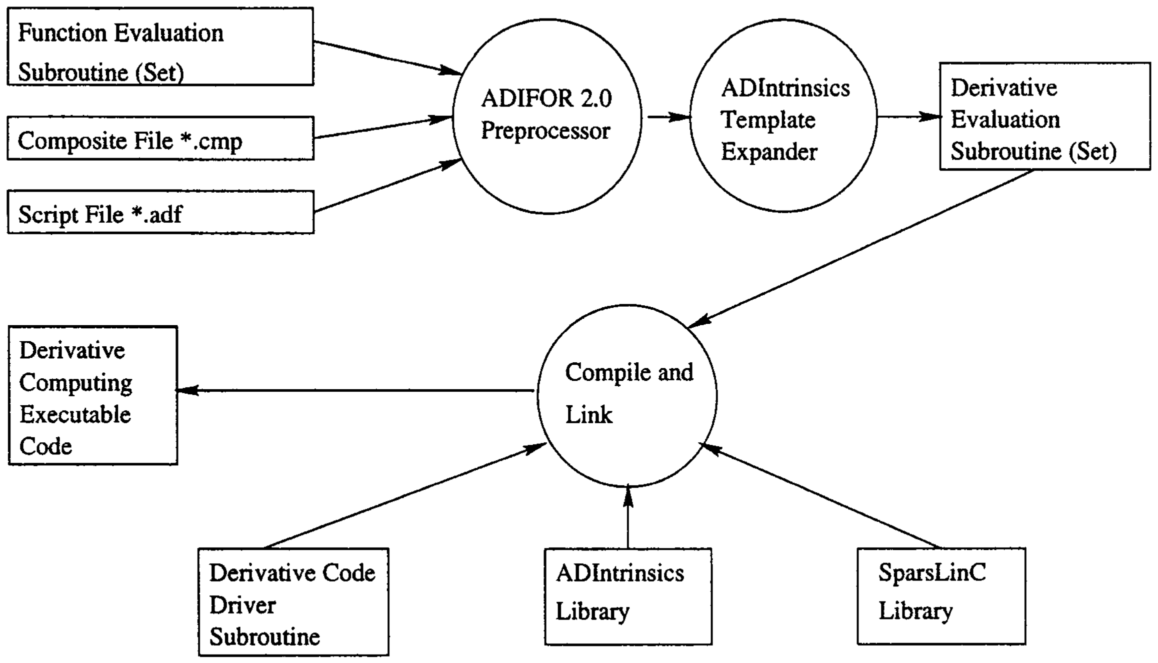
Function Evaluation Subroutine (Set)
Composite File *.cmp
Script File *.adf
ADIFOR 2.0 Preprocessor
ADIntrinsics Template Expander
Derivative Evaluation Subroutine (Set)
Derivative Computing Executable Code
Compile and Link
Derivative Code Driver Subroutine
ADIntrinsics Library
SparsLinC Library

Figure 1. Block Diagram of the ADIFOR2.0 Process

(2) Provide a dummy main program calling the top-level function evaluation subroutine;

(3) Create a "composite file" listing the names of the files containing the dummy main program, top-level subroutine, and all the lower-level subroutines directly or indirectly called by the top-level subroutine, i.e., all subroutines in the set;

(4) Create a "script file" listing the names of the composite file and of the top-level subroutine as well as the names of the dependent and independent variables and the number of independent variables;

(5) Execute ADIFOR2.0. The output is a derivative evaluation subroutine (set) corresponding to the function evaluation subroutine set. Each output file has the same name as the corresponding input file but with prefix g_. The name of each derivative evaluation subroutine is the name of the corresponding function evaluation subroutine, prefixed with g_. Subroutines that do not involve any independent or dependent variables are included unprocessed in the derivative evaluation subroutine set.

(6) Provide a derivative code "driver subroutine" invoking the top-level g_ subroutine;

(7) Compile and link the driver program, the ADIFOR-generated g_ subroutines, SparsLinC library and ADintrinsic library to build the desired derivative computing executable.

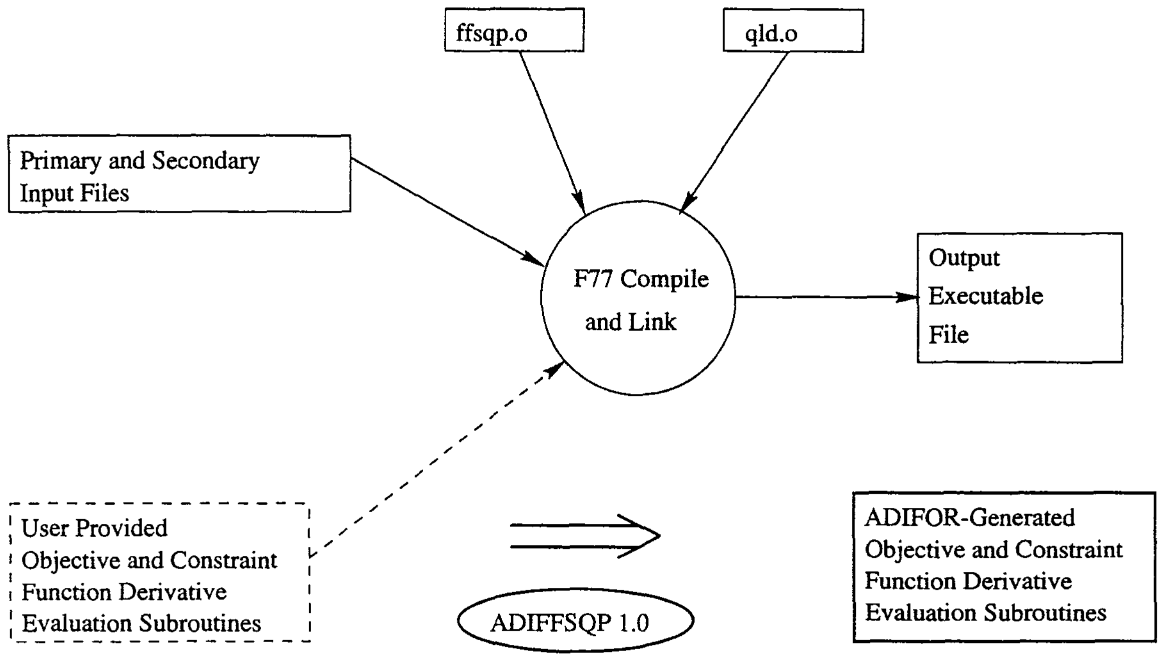
ffsqp.o
qld.o
Primary and Secondary Input Files
F77 Compile and Link
Output Executable File
User Provided Objective and Constraint Function Derivative Evaluation Subroutines
ADIFFSQP 1.0
ADIFOR-Generated Objective and Constraint Function Derivative Evaluation Subroutines

Figure 2. Block Diagram of FFSQP Process and Design Objective of ADIFFSQP

Note that ADIFOR only processes one function evaluation problem each time it is invoked, i.e., it allows only one top-level subroutine. This restriction will have implications on ADIFFSQP, as discussed in Section 2.3 below.

2.2 **FFSQP**

To use FFSQP, the user must provide one or more files, referred to below as *input files*, each of which could be in source or object form, which, together with ffsqp.o and qld.o, must form a complete, linkable program. Obviously, in a meaningful program, there is a file that includes a call to FFSQP. We will assume that only one file contains such a call. This file will be referred to as *primary input file.* For the set of input files, or *input set*, to be linkable with ffsqp.o and qld.o, it must include a subroutine evaluating the objective function(s) and another one evaluating the constraint function(s). These top-level subroutines may in turn invoke lower-level subroutines. The user may provide his/her own subroutines computing the derivatives of the objective and constraint functions, or request that FFSQP execute finite differencing by using reserved subroutine names within the FFSQP function call. If the input set consists of more than one file, the additional file(s) will be referred to as *secondary input file(s).*

An FFSQP executable code is generated as illustrated in Figure 2.

The block in dotted lines indicates an optional component. If it is not provided,

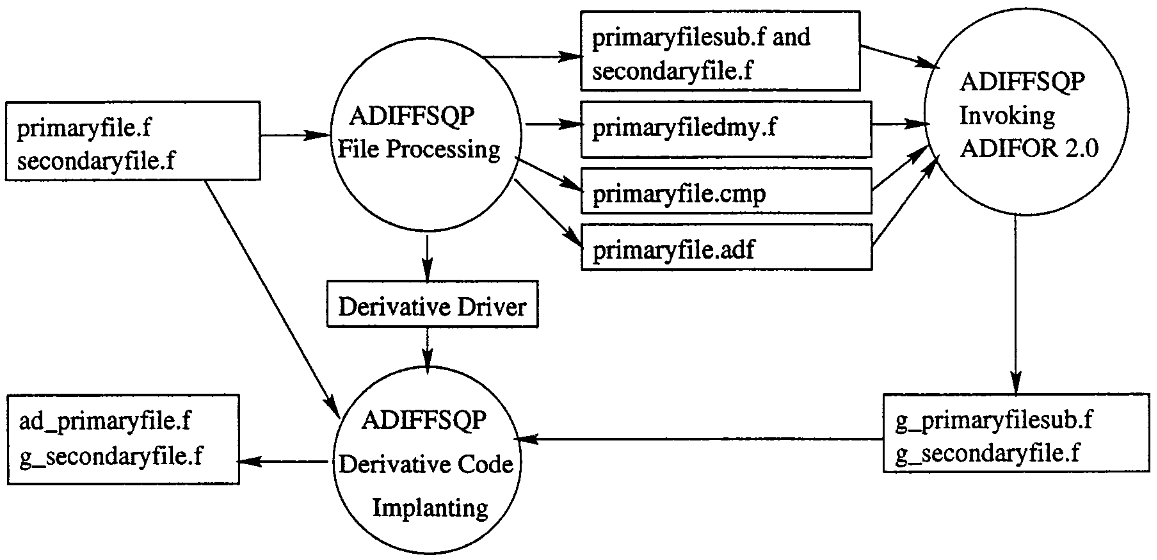

Figure 3. Block Diagram of The ADIFFSQP Process

FFSQP executes the intrinsic finite differencing. The design objective of ADIFFSQP is to replace/provide this block with ADIFOR-generated derivative evaluation subroutines, as indicated by the double-arrow in the diagram.

For more details on FFSQP, see

`http://www.isr.umd.edu/Labs/CACSE/FSQP/fsqp.html`

2.3 ADIFFSQP

The above discussion leads to the architecture of ADIFFSQP which is illustrated in Figure 3.

Given a valid input set for FFSQP, the set of source files in this input set constitutes a valid input set for ADIFFSQP provided the following three conditions are satisfied:

1. the source file set contains the primary input file as well as all the subroutines whose arguments include independent or dependent variables, or whose common blocks include independent or dependent variables, except possibly the objective and constraint derivative evaluation subroutines.

2. the primary input file contains a call to FFSQP and also includes a `parameter` statement assigning a constant value to `nparam` [1], the number of independent variables. The primary input file must also include top-level function evaluation subroutines.

3. if present, the user provided top-level derivative evaluation subroutines appear in the primary input file[2].

ADIFFSQP proceeds as follows:

[2]These subroutines will be discarded by ADIFFSQP. To avoid mistakes, the user would be well advised to remove his/her own derivative evaluation subroutines before using ADIFFSQP.

(1) Read the input file(s). Suppose the name of the primary input file is `primaryfile.f`, and that of the secondary input file is `secondaryfile.f`;

(2) Examine the arguments in the FFSQP function call to determine the names of the top-level objective function evaluation subroutine, the top-level constraint function evaluation subroutine, as well as their derivative evaluation subroutines if they exist;

(3) Generate names to be assigned to the driver subroutines (See Step (9) below). If derivative evaluation subroutines are provided, simply use those names.

(4) Write out to a separate file, `primaryfilesub.f`, all subroutines found in `primaryfile.f`, except for the two user provided top-level derivative evaluation subroutines if they exist. The names of this file and of all secondary input files will be passed to ADIFOR2.0 in the composite file (Step 6 below);

(5) Create a dummy main program `primaryfiledmy.f` calling a dummy top subroutine `primaryfiletop`, which calls both the top-level objective and constraint evaluation subroutines. The purpose of this dummy top subroutine is to avoid invoking ADIFOR2.0 twice. As pointed out in 2.1, ADIFOR2.0 only allows one top-level subroutine at a time. By using this dummy top subroutine, we are actually making ADIFOR2.0 process two functions, both of objective and of constraint, at the same time ;

(6) Create the corresponding composite file `primaryfile.cmp` and script file `primaryfile.adf`;

(7) Execute the command `Adifor2.0` to generate the derivative evaluation subroutine set correponding to the dummy top subroutine and the objective and constraint function evaluation subroutines, including both top-level ones and lower-level ones. These Adifor-generated subroutines have prefix g_, and are contained in file `g_primaryfilesub.f` and `g_secondaryfile.f`, as described in 2.1.

(8) Move all contents from the primary input file to an output file `ad_primaryfile.f`, which is generated by ADIFFSQP, except for the derivative subroutines if they exist

(9) Write two driver subroutines to the output file, invoking the ADIFOR-generated top-level objective and constraint function evaluation subroutines. The names of the driver subroutines are those assigned at Step (3) above;

(10) Move to `ad_primaryfile.f` the ADIFOR-generated subroutines from file `g_primaryfilesub.f`;

(11) Delete all temporary files;

Thus a new FORTRAN77 program is completed, containing the automatic differentiation code. It can be compiled with FFSQP to solve the optimization problem immediately.

2.4 ADIFFSQP Linked with FFSQP and ADIFOR

The whole process and interrelationship between FFSQP, ADIFOR and ADIFFSQP is illustrated in Figure 4.

By convention, when executing `adiffsqp`, the first input file name should be the primary input file. The names of all secondary input files should also be given in the

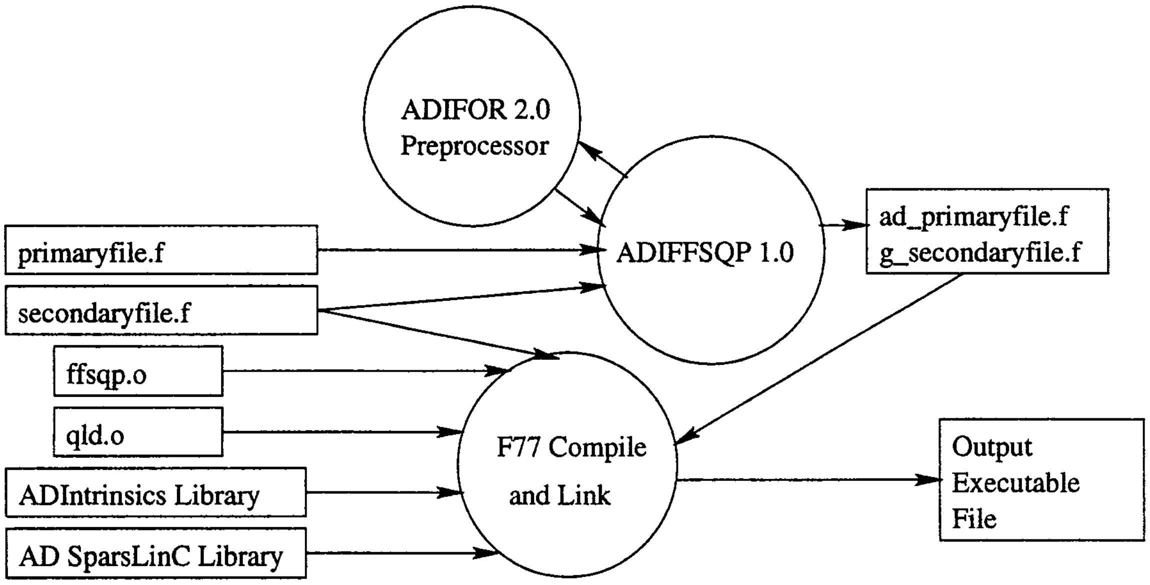

Figure 4. Block Diagram of ADIFFSQP Linking with FFSQP

input.

The limit on the number of input files is 100.

3 Installing ADIFFSQP

The ADIFFSQP distribution includes `adiffsqp.c`, `adiffsqp.h` and this user manual.

The user should obtain a copy of the ADIFOR2.0 system, which consists of the ADIFOR2.0 preprocessor, the ADIntrinsics template expander and library, and the SparsLinC library. The ADIFOR2.0 software can be retrieved by visiting:

`http://www.mcs.anl.gov/adifor`, or `http://www.cs.rice.edu/~adifor`.

These pages describe how to request access to ADIFOR2.0 and how to download the software.

The details on how to install ADIFOR2.0 can be found in [2]. This includes unpacking the files:

```
gunzip adifor2.0.tar.gz
tar xf adifor2.0.tar
gunzip adifor2.0.lib.tar.gz
tar xf adifor2.0.lib.tar
```

ADIFOR2.0 can run on a Sparc, an IBM RS 6000 or an SGI workstation and it provides the necessary libraries for each of these machines. A "C" compiler is required to compile the SparcLinC library. Since ADIFFSQP is written in C, it can be run in all environment where ADIFOR2.0 is supported.

An example of setting up certain environmental variables is as follows:

```
setenv AD_HOME /user/local/ADIFOR2.0
setenv AD_LIB  /user/local/ADIFOR2.0.lib
setenv PATH    $AD_HOME/bin:$PATH
setenv MANPATH $AD_HOME/man:$MANPATH
setenv AD_ARCH sun4
setenv AD_OS   SunOS-4.x
```

The environment variable AD_HOME is set to be the path to the directory ADIFOR2.0, AD_LIB to be the path to the directory ADIFOR2.0.lib, the variable AD_ARCH to "sun4" [2].

Then the user should compile adiffsqp.c to an object file using a C compiler, e.g.,

```
cc -c adiffsqp.c
```

ADIFFSQP is now installed and ready to accept user input.

4 Using ADIFFSQP

First, prepare an input file(s) as discussed in Section 3. An example with a single file is shown in Section 5 below: sampl1.f.

Second, execute the command to process the input file:

```
adiffsqp <primaryfile.f> [secondaryfile1.f] [secondaryfile2.f]...
```

`secondaryfile1.f` and `secondaryfile2.f` may contain several subroutines, including the top-level subroutine. ADIFFSQP will process the file(s), invoke ADIFOR2.0 and generate the output file, e.g., `ad_primaryfile.f`. ADIFOR-generated files are `g_secondaryfile1.f` and `g_secondaryfile2.f`. If `primaryfile.f` contains function evaluation subroutines, then `ad_primaryfile.f` will contain corresponding ADIFOR-generated derivative evaluation subroutines as described in Section 2.3. If it dose not, then all ADIFOR-generated subroutines will be contained in `g_secondaryfile.f`(s).

Third, `ad_primaryfile.f` can then be compiled along with all the objective files required by FFSQP and ADIFOR2.0. If there are secondary input files, e.g., `secondaryfile1`, and ADIFOR2.0 has generated corresponding g_files, e.g., `g_secondaryfile1`, for it, these should also be included:

```
    f77 ad_primaryfile.f ffsqpd.o qld.o                         \
        [secondaryfile1.f] [secondaryfile2.f] ...               \
        [g_secondaryfile1.f] [g_secondaryfile2.f] ...           \
        [others.f]                                              \
        $AD_LIB/lib/ReqADIntrinsics-$AD_ARCH.o                  \
        $AD_LIB/lib/libADIntrinsics-$AD_ARCH.a -o samplad
```

here others.f refers to those files needed in running the optimization program, but do not involve in function evaluation or derivative evaluation and thus are not provided to ADIFFSQP.

It is highly recommended that the user create a Makefile for this purpose.

5 Example

The following is a suitable main program sampl1.f (identical to that contained in the FFSQP distribution):

```
c
c     program description
c
      program sampl1
c
      integer iwsize,nwsize,nparam,nf,nineq,neq
      parameter (iwsize=29, nwsize=219)
      parameter (nparam=3, nf=1)
      parameter (nineq=1, neq=1)
      integer iw(iwsize)
      double precision x(nparam),bl(nparam),bu(nparam),
     *        f(nf+1),g(nineq+neq+1),w(nwsize)
      external obj32,cntr32,grob32,grcn32
c
      integer mode,iprint,miter,nineqn,neqn,inform
      double precision bigbnd,eps,epseqn,udelta
c
      mode=100
      iprint=1
      miter=500
      bigbnd=1.d+10
      eps=1.d-08
      epseqn=0.d0
      udelta=0.d0
```

```
c
c     nparam=3
c     nf=1
      nineqn=1
      neqn=0
c     nineq=1
c     neq=1
c
      bl(1)=0.d0
      bl(2)=0.d0
      bl(3)=0.d0
      bu(1)=bigbnd
      bu(2)=bigbnd
      bu(3)=bigbnd
c
c     give the initial value of x
c
      x(1)=0.1d0
      x(2)=0.7d0
      x(3)=o.2d0
c
      call FFSQP(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,
     *           miter,inform,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,
     *           iw,iwsize,w,nwsize,obj32,cntr32,grob32,grcn32)
      end
c
c
      subroutine obj32(nparam,j,x,fj)
      integer nparam,j
      double precision x(nparam),fj
c
      fj=(x(1)+3.d0*x(2)+x(3))**2+4.d0*(x(1)-x(2))**2
      return
      end
c
      subroutine grob32(nparam,j,x,gradfj,dummy)
      integer nparam,j
      double precision x(nparam),gradfj(nparam),dummy,fa,fb
      external dummy
c
      fa=2.d0*(x(1)+3.d0*x(2)+x(3))
      fb=8.d0*(x(1)-x(2))
      gradfj(1)=fa+fb
      gradfj(2)=fa*3.d0-fb
      gradfj(3)=fa
      return
      end
c
      subroutine cntr32(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),gj
      external dummy
c
      go to (10,20),j
 10   gj=x(1)**3-6.0d0*x(2)-4.0d0*x(3)+3.d0
      return
 20   gj=1.0d0-x(1)-x(2)-x(3)
      return
      end
c
      subroutine grcn32(nparam,j,x,gradgj,dummy)
      integer nparam,j
      double precision x(nparam),gradgj(nparam),dummy
c
      go to (10,20),j
 10   gradgj(1)=3.d0*x(1)**2
      gradgj(2)=-6.d0
      gradgj(3)=-4.d0
      return
 20   gradgj(1)=-1.d0
      gradgj(2)=-1.d0
      gradgj(3)=-1.d0
      return
      end
```

The ADIFFSQP-generated output program is as follows:

```
      program sampl1
      integer iwsize,nwsize,nparam,nf,nineq,neq
      parameter (iwsize=29, nwsize=219)
      parameter (nparam=3, nf=1)
      parameter (nineq=1, neq=1)
```

```
 integer iw(iwsize)
 double precision x(nparam),bl(nparam),bu(nparam),f(nf+1),g(nineq+n
*eq+1),w(nwsize)
 external obj32,cntr32,grob32,grcn32
 integer mode,iprint,miter,nineqn,neqn,inform
 double precision bigbnd,eps,epseqn,udelta
 external grob32,g_obj32,grcn32,g_cntr32
 mode=100
 iprint=1
 miter=500
 bigbnd=1.d+10
 eps=1.d-08
 epseqn=0.d0
 udelta=0.d0
 nineqn=1
 neqn=0
 bl(1)=0.d0
 bl(2)=0.d0
 bl(3)=0.d0
 bu(1)=bigbnd
 bu(2)=bigbnd
 bu(3)=bigbnd
 x(1)=0.1d0
 x(2)=0.7d0
 x(3)=o.2d0
 call FFSQP(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,miter,infor
*m,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,iw,iwsize,w,nwsize,obj32,cn
*tr32,grob32,grcn32)
 end
 subroutine grob32(nparam,j,x,gradj,dummy)
 integer nparam,j
 double precision dummy,x(nparam),gradj(3),
*      g_x(3,3),gj
 external dummy
 g_x(3,3)=1.0
 call g_obj32(nparam,nparam,j,x,g_x,nparam,gj,
*        gradj,nparam)
 return
 end
 subroutine obj32(nparam,j,x,fj)
 integer nparam,j
```

```
      double precision x(nparam),fj
      fj=(x(1)+3.d0*x(2)+x(3))**2+4.d0*(x(1)-x(2))**2
      return
      end
      subroutine grcn32(nparam,j,x,gradj,dummy)
      integer nparam,j
      double precision dummy,x(nparam),gradj(3),
     *      g_x(3,3),gj
      external dummy
      g_x(1,1)=1.0
      g_x(2,2)=1.0
      g_x(3,3)=1.0
      call g_cntr32(nparam,nparam,j,x,g_x,nparam,gj,
     *        gradj,nparam)
      return
      end
      subroutine cntr32(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),gj
      go to (10,20),j
 10   gj=x(1)**3-6.0d0*x(2)-4.0d0*x(3)+3.d0
      return
 20   gj=1.0d0-x(1)-x(2)-x(3)
      return
      end
      subroutine g_obj32(g_p_, nparam, j, x, g_x, ldg_x, fj, g_fj, ldg_f
     *j)
        integer nparam, j
        double precision x(nparam), fj
        integer g_pmax_
        parameter (g_pmax_ = 3)
        integer g_i_, g_p_, ldg_fj, ldg_x
        double precision d2_p, d1_p, d6_b, d5_b, d7_v, d9_v, d7_b, g_fj(
     *ldg_fj), g_x(ldg_x, nparam)
        integer g_ehfid
        data g_ehfid /0/
        call ehsfid(g_ehfid, 'obj32','g_obj32.f')
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        d7_v = (x(1) + 3.d0 * x(2) + x(3)) * (x(1) + 3.d0 * x(2) +
x(3))
        d2_p = 2.0d0 * (x(1) + 3.d0 * x(2) + x(3))
        d9_v = (x(1) - x(2)) * (x(1) - x(2))
        d1_p = 2.0d0 * (x(1) - x(2))
        d5_b = 4.d0 * d1_p
        d6_b = d5_b + d2_p
        d7_b = -d5_b + d2_p * 3.d0
        do g_i_ = 1, g_p_
          g_fj(g_i_) = d2_p * g_x(g_i_, 3) + d7_b * g_x(g_i_, 2) + d6_b
     ** g_x(g_i_, 1)
        enddo
        fj = d7_v + 4.d0 * d9_v
        return
      end
      subroutine g_cntr32(g_p_, nparam, j, x, g_x, ldg_x, gj, g_gj, ldg_
     *gj)
        integer nparam, j
        double precision x(nparam), gj
        integer g_pmax_
        parameter (g_pmax_ = 3)
        integer g_i_, g_p_, ldg_gj, ldg_x
        double precision d1_p, d2_v, g_gj(ldg_gj), g_x(ldg_x, nparam)
        integer g_ehfid
        data g_ehfid /0/
        call ehsfid(g_ehfid, 'cntr32','g_cntr32.f')
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        goto (10, 20), j
 10     d2_v = x(1) ** ( 3 - 2)
        d2_v =  d2_v * x(1)
        d1_p =  3 *  d2_v
        d2_v =  d2_v * x(1)
        do g_i_ = 1, g_p_
          g_gj(g_i_) = -4.0d0 * g_x(g_i_, 3) + (-6.0d0) * g_x(g_i_, 2) +
     * d1_p * g_x(g_i_, 1)
        enddo
        gj = d2_v - 6.0d0 * x(2) - 4.0d0 * x(3) + 3.d0
        return
```

```
20     do g_i_ = 1, g_p_
         g_gj(g_i_) = -g_x(g_i_, 3) + (-g_x(g_i_, 2)) + (-g_x(g_i_, 1))
       enddo
       gj = 1.0d0 - x(1) - x(2) - x(3)
       return
     end
```

References

[1] J.L.Zhou &amp; A.L.Tits, *User's Guide for FSQP Version3.0c: 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, College Park, MD 20742m 1992.

[2] Christian Bischof, Alan Carle, Peyvand Khademi, Andrew Mauer, and Paul Hovland, *ADIFOR2.0 User's Guide*, Mathematics and Computer Science Division Technical Memorandum No.192 and Center for Research on Parallel Computation Technical Report CRPC-95516-s, August 1995.

[3] Andreas Griewank, *On Automatic Differentiation*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60493, U.S.A.