IFFCO

Implicit Filtering for Constrained Optimization

USERS' GUIDE

by

Paul Gilmore

Contents

Chapter 1

Introduction to IFFCO

IFFCO (Implicit Filtering for Constrained Optimization) is an algorithm for optimizing functions with multiple minima. IFFCO is designed to solve problems subject to simple box constraints. The mathematical description of these problems is:

$$\min_{x \in Q} \hat{f} : R^n \rightarrow R \ \text{ where } \ Q = \{ x \in R^n \mid l^i \leq x^i \leq u^i \ i = 1, \ldots, n \} \ , \quad (1.1)$$

where l^i and u^i are the lower and upper bounds respectively on the ith variable. The set Q, defined by the constraints on the variables is called the hyper-box.

IFFCO is designed to minimize functions of the form:

$$\hat{f}(x) = f(x) + \phi(x) \ . \quad (1.2)$$

$f(x)$ is a smooth function with a simple form. For example, $f(x)$ could be a convex quadratic. $\phi(x)$ contains the low-amplitude high-frequency terms of $\hat{f}(x)$. In this context, low amplitude means

$$\max_{x,y \in Q} |\phi(x) - \phi(y)| \ll \max_{x,y \in Q} |f(x) - f(y)| \ . \quad (1.3)$$

$\phi(x)$ need not be continuous. IFFCO is particularly effective on problems where the amplitude of $\phi(x)$ decays near local minima of $f(x)$. It is not necessary to be able to calculate $f(x)$ and $\phi(x)$. It is only necessary that $\hat{f}(x)$ behaves like a function that satisfies equation (1.2). An example of the type of function that IFFCO is designed to minimize is shown in figure 1.1.

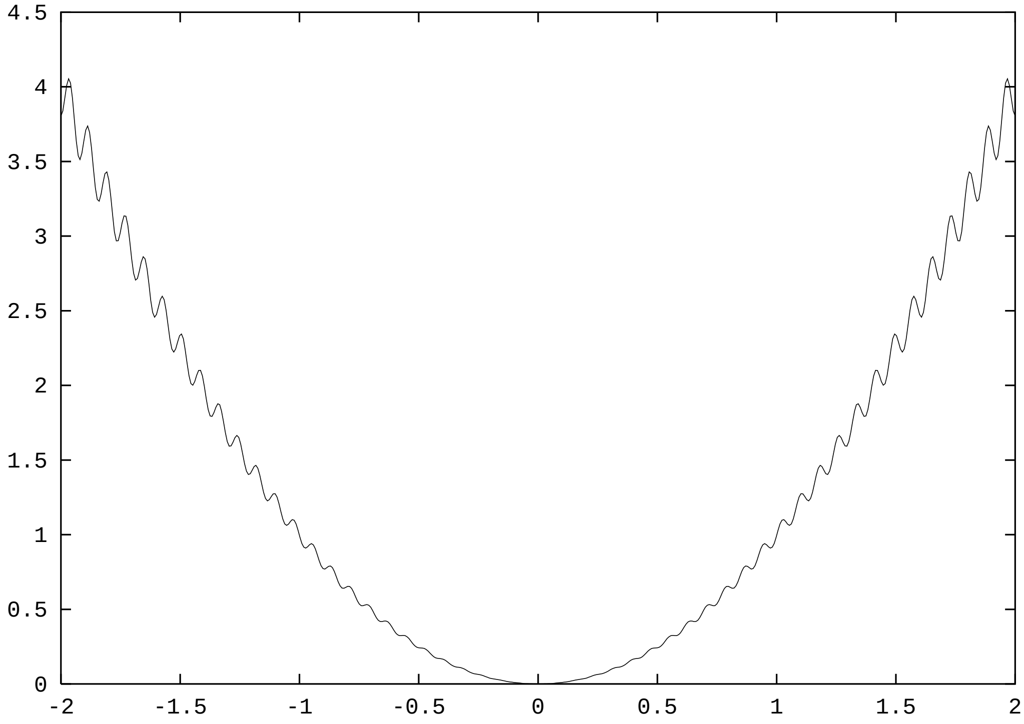

Figure 1.1: A function that IFFCO can minimize.

Applications in which functions of the form (1.2) arise are described in [4], [5], [6], and [7]. The algorithm used in IFFCO is analyzed in [3].

To obtain IFFCO, send e-mail to paul@latour.math.ncsu.edu.

IFFCO is a variation on the gradient projection method described in [1], that uses a sequence of finite difference steps (scales) to approximate the gradient. A brief outline of the algorithm used in IFFCO is given below. A more detailed description of the algorithm is given in Section 3.2.

Algorithm 1.1 *Given an initial point $x \in Q$, an initial scale h, and a lower bound for the scales used to calculate the gradient, $minh$:*

1. *Calculate the finite difference gradient $d(x)$ using the scale h.*

2. *If x satisfies a termination criteria (described in Section 3.2) depending on h:*

   (a) *Set $h = \frac{h}{2}$.*

   (b) *If $h < minh$ terminate. Otherwise go to 1.*

3. *Solve $S\delta x = -d(x)$, where S is an approximation to the Hessian. The calculation of S is described in Section 3.2.*

4. *Set $x = P(x + \alpha\delta x)$, where $P(y)$ indicates the projection of y onto Q. The step length parameter α is determined using a line search algorithm described in Section 3.2.*

5. *Go to 1.*

A variation of Algorithm 1.1 is to restart Algorithm 1.1 using the last point obtained in the iterative process as the next initial point until a point is obtained that satisfies the termination criteria at every scale. A point that satisfies the termination criteria at every scale is called a minimum at all scales.

Chapter 2

Using IFFCO

This chapter provides information about using IFFCO, including details on calling IFFCO and selecting appropriate values for parameters. The troubleshooting section describes common problems and advises the user on how to solve them. To use IFFCO the user must have access to two LINPACK subroutines: dchdc and dposl, and to the BLAS subroutines and functions: daxpy, ddot, and, dswap. The user must also provide a subroutine for evaluating the function the user wishes to minimize. This subroutine must be in the format described in Section 2.1.

2.1 Calling IFFCO

This section describes the calling sequence for IFFCO and the arguments in the calling sequence. User provided subroutines are also discussed.

- **Calling sequence**

  call iffco(fcn,x,u,l,fscale,rminh,rmaxh,n,maxit,restart,writ,tol,f,ncuts)

- **Arguments**
  The arguments are listed in the order they appear in the calling sequence.

  - **On Entry**

fcn – is the argument containing the name of the user-supplied subroutine that returns values for the function to be minimized. *fcn* must be declared external in the calling program.

x – is a user-supplied double-precision vector of length n. x is the initial point in the iterative process. x must be within the hyper-box defined by the constraints.

u – is a user-supplied double-precision vector of length n. u is the vector containing the upper bounds for the n independent variables. The hyper-box defined by the constraints on the variables is mapped to the unit cube in IFFCO. IFFCO performs all calculations on points within the unit cube. The final solution is mapped back to the original hyper-box before being returned to the user.

l – is a user-supplied double-precision vector of length n. l is the vector containing the lower bounds for the n independent variables.

fscale – is a user-supplied double-precision constant used to scale the function. *fscale* should be an approximation to the maximum size in absolute value the function to be minimized obtains in the hyper-box Q. The default value for *fscale* is 1.d0.

rminh – is a user-supplied double-precision constant. *rminh* is a lower bound for the last and smallest finite difference step (scale) that IFFCO uses to calculate the finite difference gradient. When a scale is less than *rminh,* the algorithm either terminates or restarts. However, an optimization run is always done with the scale *rmaxh* regardless of the value of *rminh.* *rminh* must satisfy:

$$0 < rminh \, . \tag{2.1}$$

If inequality 2.1 is not satisfied, *rminh* is set to 1.d-4. See Section 2.2 for help in determining *rminh.*

rmaxh – is a user-supplied double-precision constant. *rmaxh* is the first and largest scale the algorithm uses to calculate the finite difference gradient. *rmaxh* must satisfy:

$$0 < rmaxh \leq 0.5 \, . \tag{2.2}$$

The upper bound in inequality 2.2 is necessary to insure that the finite difference steps taken using the larger scales do not violate the constraints on the variables. The lower bound in inequality 2.2 insures that the finite difference gradient is well defined. If inequality 2.2 is not satisfied, *rmaxh* is set to 5.d-1. See Section 2.2 for help in determining *rmaxh.*

n – is an integer that represents the dimension of the problem. If $n > 100$ then the parameter maxor in the variable list at the beginning of IFFCO must be set larger than n. maxor is a parameter used to dimension the work arrays used in IFFCO.

maxit – is a user-supplied integer. *maxit* specifies the maximum number of iterations the algorithm is allowed to take at a given scale. The default value for *maxit* is 200.

restart – is a user-supplied integer. If *restart* is set to 1, the algorithm restarts after the smallest scale has been used. The restart uses the last point generated by the algorithm as the new initial point and *rmaxh* as the first scale in the sequence of finite difference steps. Restarts are done to help insure that the final point in the iterative process is a minimum at all scales. In practice restarts are not usually needed. Furthermore restarts are expensive in terms of function evaluations. Thus the user should select restarts (by setting $restart = 1$) only if the answers obtained by IFFCO are not as good as the user expects.

writ – is a user-supplied integer. *writ* specifies the verbosity of the algorithm:

writ = 1 causes the algorithm to write error messages and the values of user-supplied parameters to standard output.

writ = 2 causes the algorithm to produce the same information as for $writ = 1$ plus a history of the iterative process.

tol – is a user-supplied double-precision constant used for determining convergence of the algorithm at a given point for a given scale. Determining appropriate values for *tol* is problem-dependent. See Section 2.2 for help determining *tol*. The default value for *tol* is 1.

ncuts – is a user-supplied integer. *ncuts* specifies the maximum number of cutbacks in the line search the algorithm may take. The default value for *ncuts* is 10.

– **On Return**

x – is the final point obtained in the optimization process. x should be a good approximation to the global minimum for the function within the hyper-box.

f – is the value of the function at x.

- **User-Supplied Subroutines**

  – The function evaluation subroutine. The name of this subroutine is determined by the user and must be declared in an external statement in the user's calling program. The subroutine should have the form:

```
subroutine subroutine_name(x,n,f)
integer n
real*8 x(n),f
.
.
.
return
end
```

  – LINPACK Subroutines. The user must have access to two LINPACK subroutines: dchdc and dposl, and to the BLAS subroutines and functions: daxpy, ddot, and, dswap.

2.2 Selecting Values for Arguments

This section advises the user how to choose values for several parameters used in IFFCO.

fscale – is used to scale the function being optimized by IFFCO. *fscale* should be an estimate of the maximum in absolute value that the function $\hat{f}(x)$ achieves on Q.

Unfortunately, $max_{x \in Q}\left(|\hat{f}(x)|\right)$ is not usually known. Hence, users may have to experiment with values of *fscale* to determine a value appropriate for their problem. If *fscale* is too large, the function appears flattened, and IFFCO may not be able to obtain a solution acceptable to the user. Choosing *fscale* too large often manifests itself by IFFCO obtaining answers that are unacceptable to the user while reporting convergence at many scales without taking a step at these scales. If *fscale* is too small, the gradients calculated may become too large. Choosing *fscale* too small often manifests itself by the line search being unable to find a suitable new point for many point scale pairs.

The default value for *fscale* is 1.

rminh – is a lower bound for the minimum scale used. If *rminh* is too large, the algorithm may not obtain an answer acceptable to the user. If *rminh* is too small, some of the scales used may be so small that gradients calculated using them are dominated by changes in $\phi(x)$.

If the amplitude of $\phi(x)$ is approximately constant then, optimally, *rminh* should be a low estimate of $O(\max_{x \in Q}(|\phi(x)/fscale|^{1/3})$. If $\phi(x)$ decays near minima of $f(x)$, *rminh* may be smaller. For instance, if it is known that $\hat{f}(x)$ is smooth near minima of $f(x)$, *rminh* may be on the order of the cube root of machine precision. Unfortunately, the behavior of $\phi(x)$ is usually not well known, so users may have to experiment with *rminh* to find a suitable value for their problems.

As mentioned earlier, poor answers may indicate that *rminh* is too large. Choosing *rminh* too small often manifests itself by the line search being unable to find a suitable new point for some of the smaller scales.

Determining a suitable value for *rminh* is a complicated problem involving the curvature of $f(x)$, the amplitude of $\phi(x)$, the amount of accuracy needed by the user, and the cost per function evaluation. Therefore users may have to experiment.

The default value for *rminh* is 10^{-4}.

rmaxh – *rmaxh* is the first and largest scale in the sequence of scales used to calculate the gradient. If the user chooses $rmaxh \leq 0$ or $rmaxh > 0.5$, *rmaxh* is set to 0.5. If $rmaxh \leq rminh$, IFFCO performs a one scale optimization run using *rmaxh*.

The user should choose a large value for *rmaxh* if the initial point may be far from a minimum of $f(x)$, or if the amplitude of $\phi(x)$ is large. Usually, *rmaxh* should be a high estimate of $O(\max_{x \in Q} |\phi(x)/fscale|^{1/3})$. However if the user is confident that $\phi(x)$ is small or the initial point is close to an acceptable answer small values of *rmaxh* may be chosen.

If the algorithm is unable to reduce the function while using *rmaxh* and other large scales then *rmaxh* may be too small. If the user has chosen a small value of *rmaxh* and IFFCO is returning answers unacceptable to the user then increasing *rmaxh* might help.

The default value for *rmaxh* is 0.5.

maxit – is the maximum number of iterates IFFCO will take at a given scale. The size of *maxit* needed depends on the problem and the number of scales used.

The default value of *maxit* is 200.

restart – is an integer that tells IFFCO whether to do restarts. If the user sets $restart = 1$, IFFCO will perform restarts; otherwise no restarts are done. Restarts should be done only if IFFCO is returning answers that are not as good as the user expects. Restarts are done to help ensure that the final answer obtained is a minimum at all scales.

The default is not to do restarts.

tol – *tol* is a constant used in determining convergence at a given scale. Convergence at a point x is determined using,

$$\|x - P(x - d(x))\| \leq tol \times h \ .$$

Where $P(x - d(x))$ is the projection of the steepest descent step onto the hyper-box Q. At this point no heuristic exists for determining *tol*. In many computations, using the default value for *tol* ($tol = 1$) has worked well. However, this is not the case for all problems.

If *tol* is too large, the algorithm will report convergence at many scales without finding a new point at these scales, and return an answer unacceptable to the user. If this phenomena is observed, reducing *tol* may help. If *tol* is too small, the algorithm may not reduce scales fast enough. In this case the line search will be unable to find acceptable new points for many scales, thus costing the user many unnecessary function evaluations. If this phenomena is observed, increasing tol may help.

The default value for tol is 1.

2.3 Troubleshooting

This section discusses common problems and gives advice on fixing them. Set $writ = 2$ to obtain a history of the iterative process in the output from IFFCO. The first column in the output, labeled with K, gives the number of iterates at the current scale. The second column, labeled with $|X|$ gives the two-norm of the current point within the unit cube. The third column, labeled with F, is the function value at the current iterate divided by $fscale$. The fourth column, labeled with $|GF|$, gives the two-norm of the finite difference gradient, calculated using the current scale, at the current point. The fifth column, labeled with H gives the current scale being used. The sixth column, labeled $CUTS$ gives information on the line search. In this column IFFCO either returns the number of cutbacks used in the line search or one of the following two messages:

1. Convergence. This message indicates that the current point is a minimum for the current scale.
2. Line search failed. This message indicates that the line search was unable to find an acceptable new point.

2.3.1 Failure in the Line Search

This section discusses what to do if the line search is not able to find a new point at many scales. Failure in the line search is indicated by the warning "Line search failed" in the output from IFFCO.

- If IFFCO is returning answers that are not as good as the user expects and IFFCO is reporting failure in the line search at many scales then $fscale$ may be too small. Here is an example of output from IFFCO when $fscale$ is too small for the problem:

```
   K     | X |         F          | GF |        H         CUTS

   0  0.1000D+01  0.8495D+04  0.1134D+05  0.5000D+00   10
   1  0.2357D-01  0.8443D+04  0.1133D+05  0.5000D+00  Convergence
   0  0.2357D-01  0.8443D+04  0.1700D+05  0.2500D+00  Line search failed
   0  0.2357D-01  0.8443D+04  0.1983D+05  0.1250D+00  Line search failed
   0  0.2357D-01  0.8443D+04  0.2125D+05  0.6250D-01  Line search failed
Number of function evaluations:   58
```

Using larger values for $fscale$ usually solves this problem. Users may have to experiment to find a good value for $fscale$.

- If IFFCO is returning answers acceptable to the user but IFFCO is reporting failure in the line search at many scales, then tol may be too small. Here is an example of output from IFFCO when tol is too small for the problem:

```
   K     | X |         F          | GF |        H         CUTS

   0  0.1000D+01  0.8495D+01  0.1134D+02  0.5000D+00    1
   1  0.3571D+00  0.8009D+01  0.1134D+02  0.5000D+00    0
   2  0.7308D+00  0.7882D+00  0.5182D+01  0.5000D+00  Line search failed
   0  0.7308D+00  0.7882D+00  0.6157D+01  0.2500D+00    5
   1  0.6685D+00  0.1758D+00  0.5112D+00  0.2500D+00    1
   2  0.5145D+00  -.3437D-02  0.7545D-01  0.2500D+00    1
   3  0.4967D+00  -.9604D-02  0.2065D-01  0.2500D+00  Line search failed
   0  0.4967D+00  -.9604D-02  0.2067D-01  0.1250D+00  Line search failed
   0  0.4967D+00  -.9604D-02  0.2067D-01  0.6250D-01  Line search failed
Number of function evaluations:   58
```

- If IFFCO is returning answers acceptable to the user but IFFCO is reporting failure in the line search for many of the smaller scales, then $rminh$ may be too small. Here is an example of output from IFFCO when $rminh$ is too small for the problem:

```
   K     | X |         F          | GF |        H         CUTS

   0  0.1000D+01  0.8508D+01  0.1134D+02  0.5000D+00    1
   1  0.3477D+00  0.8008D+01  0.1133D+02  0.5000D+00  Convergence
   0  0.3477D+00  0.8008D+01  0.1697D+02  0.2500D+00    4
   1  0.7884D+00  0.7990D+01  0.1697D+02  0.2500D+00    0
   2  0.5091D+00  -.9306D-02  0.5221D-01  0.2500D+00  Convergence
   0  0.5091D+00  -.9306D-02  0.5615D-01  0.1250D+00  Convergence
```

```
  0  0.5091D+00  -.9306D-02  0.5754D-01  0.6250D-01  Convergence
  0  0.5091D+00  -.9306D-02  0.5792D-01  0.3125D-01  Convergence
  0  0.5091D+00  -.9306D-02  0.5802D-01  0.1562D-01    1
  1  0.5011D+00  -.9794D-02  0.2134D-01  0.1563D-01  Convergence
  0  0.5011D+00  -.9794D-02  0.8965D-02  0.7813D-02  Convergence
  0  0.5011D+00  -.9794D-02  0.3496D+00  0.3906D-02    3
  1  0.5027D+00  -.9869D-02  0.2015D+00  0.3906D-02    2
  2  0.5027D+00  -.9939D-02  0.2809D-02  0.3906D-02  Convergence
  0  0.5027D+00  -.9939D-02  0.4012D-01  0.1953D-02  Line search failed
  0  0.5027D+00  -.9939D-02  0.5027D-01  0.9766D-03  Line search failed
  0  0.5027D+00  -.9939D-02  0.1020D+00  0.4883D-03    5
  1  0.5027D+00  -.9939D-02  0.6640D-01  0.4883D-03    2
  2  0.5028D+00  -.9939D-02  0.6526D-01  0.4883D-03  Line search failed
  0  0.5028D+00  -.9939D-02  0.7184D-01  0.2441D-03  Line search failed
  0  0.5028D+00  -.9939D-02  0.7356D-01  0.1221D-03    5
  1  0.5028D+00  -.9940D-02  0.6012D-03  0.1221D-03    2
  2  0.5028D+00  -.9940D-02  0.5869D-03  0.1221D-03  Line search failed
  0  0.5028D+00  -.9940D-02  0.4747D-03  0.6104D-04  Line search failed
  0  0.5028D+00  -.9940D-02  0.4466D-03  0.3052D-04  Line search failed
  0  0.5028D+00  -.9940D-02  0.4395D-03  0.1526D-04  Line search failed
Number of function evaluations:   179
```

2.3.2 Poor Answers

This section discusses what to do if IFFCO is obtaining answers that the user feels are unacceptable but the message "Line search failed" is not reported in the output from IFFCO. To determine if poor answers are being obtained the user must have some idea as to what constitutes a poor answer in their problem.

- If IFFCO reports convergence at many scales but poor answers are being obtained then *tol* may be too big and/or *fscale* may be too big. An example of this behavior follows:

```
   K     | X |         F          | GF |         H          CUTS

  0  0.1000D+01  0.8508D+00  0.1134D+01  0.5000D+00  Convergence
  0  0.1000D+01  0.8508D+00  0.1701D+01  0.2500D+00  Convergence
  0  0.1000D+01  0.8508D+00  0.1984D+01  0.1250D+00  Convergence
  0  0.1000D+01  0.8508D+00  0.2125D+01  0.6250D-01  Convergence
Number of function evaluations:   12
```

- If IFFCO obtains poor answers and $rmaxh$ is set to a value less than 0.5, then the iterates may have become trapped in a local minima. Using a larger value of $rmaxh$ may help.

- If IFFCO returns poor answers and $rminh$ has been set to a value larger than 10^{-4}, the answer returned by IFFCO may not have the accuracy required by the user. Using the default value for $rminh$ (10^{-4}) or another small value may help. The size of $rminh$ to use depends on the problem, the accuracy needed by the user, and the precision of the hardware being used.

Chapter 3

Algorithm Description

3.1 Overview

IFFCO is a projected quasi-Newton algorithm that uses a decreasing sequence of finite difference steps (scales) to approximate the gradient. It uses an approximation to the Hessian and a line search algorithm that gives the code global capabilities. IFFCO is based on an algorithm presented in [1].

As mentioned in Chapter 1 the function to be minimized is of the form:

$$\hat{f}(x) = f(x) + \phi(x)$$

$f(x)$ is smooth function with only a few minima within the hyper-box defined by the constraints. $\phi(x)$ contains the low-amplitude high-frequency terms of $\hat{f}(x)$. $\phi(x)$ creates local minima in $\hat{f}(x)$. These local minima will trap most gradient based algorithms. However by using a sequence of finite difference steps (scales) to approximate the gradient these local minima can be avoided and the global minima can be found up to the level of noise caused by $\phi(x)$. The first elements in the sequence of scales are fairly large, typically half the length of the hyper-box. Thus, the approximate gradient $d(x)$ gives global information about the problem and allows the sequence of quasi-Newton steps to avoid the local minima. As the sequence continues, the algorithm becomes more and more of a local algorithm having the fast local convergence properties for which quasi-Newton methods are known.

Because analytic second derivatives may not be available and because the cost of function evaluations may be too high to use a difference Hessian, the SR1 (Symmetric Rank one) approximation to the Hessian is used. This approximation is given by the following formula:

$$S^c = S^- + \frac{r^- {r^-}^T}{{r^-}^T s^-}$$

where S^- is the previous SR1 approximation to the Hessian,

$$s^- = x^c - x^- \ ,$$

$$r^- = y^- - S^- s^- \ ,$$

$$y^- = d(x^c) - d(x^-) \ ,$$

and x^- is the iterate previous to x^c the current iterate. To accommodate the constraints a reduced approximate Hessian is used. The reduced approximate Hessian is obtained by setting the off diagonal elements in the ith row and column to zero if the constraints on the ith variable are active.

To keep the sequence of quasi-Newton steps within the hyper-box, each step is projected onto the hyper-box using the following formula:

$$P(x)_i = \begin{cases} u_i, & \text{if } x_i > u_i \\ x_i, & \text{if } l_i \le x_i \le u_i \\ l_i, & \text{if } x_i < l_i \end{cases} \ .$$

3.2 Algorithm Outline

1. Calling sequence

   call iffco(fcn,x,u,l,fscale,rminh,rmaxh,n,maxit,restart,writ,tol,f,ncuts)

   See Section 2.1 for a description of the arguments.

2. Preprocessing

   (a) Scale the initial point for use in IFFCO using

   $$x = \frac{x}{u-l} - \frac{l}{u-l} \ .$$

   (b) Check for errors in the user-supplied parameters.

3. Set $h = rmaxh$.
4. Set $k = 0$.
5. Calculate the function value $f(x)$ at the current iterate.
6. Calculate the finite difference gradient $d(x)$ at the current point using the current scale.
   (a) Calculate the ith element of the gradient using the centered difference formula, unless a finite difference step violates a constraint. If a constraint is violated, use either the forward or backward difference formula, depending on which difference step violates a constraint.
   (b) Set the ith component of the gradient to zero if the ith constraint is active and the ith component of the gradient gives a direction out of the hyper-box.
7. Test for convergence.
   (a) Calculate the projection of the steepest descent step $P(x - d(x))$ onto the hyper-box.
   (b) If $\|x - P(x - d(x))\| \leq tol \times h$, go to 14.
8. Calculate the negative quasi-Newton step p.
   (a) If this is the first step taken at the current scale, then $p = d(x)$.
   (b) Calculate the reduced SR1 approximate Hessian S.
   (c) If S is not positive definite, then $p = d(x)$. Otherwise, solve $Sp = d(x)$.
   (d) If $\|p\| < h$, set $p = \frac{h}{\|p\|}$.
9. Perform the line search.
   (a) Set $j = 1$.
   (b) Calculate the step size parameter α.
      i. If $j = 1$, set $\alpha = 1$.

ii. If $P(x - \hat{\alpha}p)$ activates a previously unactivated constraint, set $\alpha = 0.5\hat{\alpha}$. $\hat{\alpha}$ is the previous step size parameter.

iii. If $\hat{\alpha}$ was the first step size parameter such that $P(x - \hat{\alpha}p)$ did not activate a previously unactivated constraint, calculate α using a quadratic model of the function. See [2] for details on the formulation of the quadratic model.

iv. Otherwise, use a cubic model of the function to calculate α. See [2] for details on the formulation of the cubic model.

(c) If $\|\alpha p\| < 0.01h$, return reporting failure in the line search.

(d) Calculate $P(x - \alpha p)$ and $f(P(x - \alpha p))$.

(e) If $f(P(x - \alpha p)) \leq f(x) - 10^{-4}\alpha d(x)^T p$,

i. Set $x = P(x - \alpha p)$.

ii. Return reporting success in the line search.

(f) Set $j = j + 1$.

(g) If $j > ncuts$, return reporting failure in the line search.

10. If the line search failed, go to 14.

11. If the line search succeeded, set $k = k + 1$.

12. Calculate the gradient at the new point using the current scale. Use the same procedure as in 6. Go to 7.

13. If $k > maxit$, go to 14.

14. Set $h = \frac{1}{2}h$.

15. If $h \geq rminh$, go to 4.

16. If $h < rminh$ and no restarts are to be performed, go to 19.

17. If $h < rminh$ and no steps have been taken at any scale, go to 19.

18. Go to 3.

19. Unscale the final iterate and the function value at the final iterate and return to the calling program.

Bibliography

[1] D. B. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Trans. Autom. Control*, (2):174–184, 1976.

[2] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Nonlinear Equations and Unconstrained Optimization.* Prentice-Hall, Englewood Cliffs, N.J, 1983.

[3] P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. Submitted for publication in SIAM J. Optimization.

[4] D. E. Stoneking, G. L. Bilbro, R. J. Trew, P. Gilmore, and C. T. Kelley. Yield optimization using a GaAs process simulator coupled to a physical device model. In *Proceedings IEEE/Cornell Conference on Advanced Concepts in High Speed Semiconductor Devices and Circuits*, pages 374–383, 1991.

[5] D.E. Stoneking, G.L. Bilbro, R.J. Trew, P. Gilmore, and C. T. Kelley. Yield optimization using a GaAs process simulator coupled to a physical device model. *IEEE Transactions on Microwave Theory and Techniques*, 40(7):1353–1363, 1992.

[6] T. A. Winslow, R. J. Trew, P. Gilmore, and C. T. Kelley. Doping profiles for optimum class B performance of GaAs MESFET amplifiers. In *Proceedings IEEE/Cornell Conference on Advanced Concepts in High Speed Semiconductor Devices and Circuits*, pages 188–197, 1991.

[7] T. A. Winslow, R. J. Trew, P. Gilmore, and C. T. Kelley. Simulated performance optimization of GaAs MESFET amplifiers. In *Proceedings*

IEEE/Cornell Conference on Advanced Concepts in High Speed Semiconductor Devices and Circuits, pages 393–402, 1991.