CG_DESCENT Version 1.1 User's Guide *

William W. Hager† Hongchao Zhang‡

December 10, 2004

*This material is based upon work supported by the National Science Foundation under Grant No. CCR-0203270.

†hager@math.ufl.edu, http://www.math.ufl.edu/∼hager, PO Box 118105, Department of Mathematics, University of Florida, Gainesville, FL 32611-8105. Phone (352) 392-0281. Fax (352) 392-8357.

‡hzhang@math.ufl.edu, http://www.math.ufl.edu/∼hzhang, PO Box 118105, Department of Mathematics, University of Florida, Gainesville, FL 32611-8105.

1 Introduction

This document provides a guide for using the Fortran 77 code CG_DESCENT, an implementation of the conjugate gradient algorithm for which the search directions are always descent directions. The code along with the papers [1, 2] developing the algorithm and comparing its convergence properties to that of other algorithms for unconstrained optimization are posted at the following web site:

http://www.math.ufl.edu/∼hager/papers/CG

In this manual, we explain the design of the software and how to use it.

The conjugate gradient method is an approach for solving an unconstrained optimization problem of the following form:

$$\min \{f(\mathbf{x}) : \mathbf{x} \in \Re^n\},$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is continuously differentiable. The iterates $\mathbf{x}_k$, $k \geq 0$, in conjugate gradient methods satisfy the recurrence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k,$$

where the stepsize α_k is positive, and the directions $\mathbf{d}_k$ are generated by the rule:

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k, \quad \mathbf{d}_0 = -\mathbf{g}_0.$$

In CG_DESCENT, we make the following special choice for the parameter β_k:

$$\begin{aligned} \beta_k &= \max\{B_k, \eta_k\}, \quad \text{where} \\ \eta_k &= \frac{-1}{\|\mathbf{d}_k\| \min\{\eta, \|\mathbf{g}_k\|\}}, \\ B_k &= \frac{1}{\mathbf{d}_k^{\mathsf{T}} \mathbf{y}_k} \left(\mathbf{y}_k - 2\mathbf{d}_k \frac{\|\mathbf{y}_k\|^2}{\mathbf{d}_k^{\mathsf{T}} \mathbf{y}_k} \right)^{\mathsf{T}} \mathbf{g}_{k+1}. \end{aligned}$$

Here $\eta > 0$ is a user specified constant.

The stepsize α_k is computed by a line search routine that exploits a combination of secant and bisection steps for fast convergence. The line search is terminated when the Wolfe conditions [3, 4] are satisfied. Defining $\phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{d}_k)$, these conditions are:

$$\delta \phi'(0) \geq \frac{\phi(\alpha_k) - \phi(0)}{\alpha_k} \quad \text{and} \quad \phi'(\alpha_k) \geq \sigma \phi'(0), \tag{1}$$

where $0 < \delta \leq \sigma < 1$.

In [1] we observe that the first condition in (1) is difficult to implement numerically since the subtraction $\phi(\alpha_k) - \phi(0)$ is relatively inaccurate near a local minimum. To cope with this numerical inaccuracy, we introduce the approximate Wolfe conditions in [1] and [2]:

$$(2\delta - 1)\phi'(0) \geq \phi'(\alpha_k) \geq \sigma\phi'(0), \tag{2}$$

where $0 < \delta < 1/2$ and $\delta \leq \sigma < 1$. The first inequality in (2) is an approximation to the first inequality in (1). In a neighborhood of a local minimum, this approximation can often be evaluated more accurately than the original condition. The approximate Wolfe conditions are employed only when

$$\phi(\alpha_k) \leq \phi(0) + \epsilon_k, \tag{3}$$

where ϵ_k is an estimate for the error in the function value at iteration k. We incorporate the following possible expressions for the error in the function value:

$$\epsilon_k = \epsilon C_k \quad \text{or} \quad \epsilon_k = \epsilon, \tag{4}$$

where ϵ is a small, user specified parameter, and C_k is generated by the following recurrence:

$$\left.\begin{array}{ll} Q_k = 1 + Q_{k-1}\Delta, & Q_{-1} = 0, \\ C_k = C_{k-1} + (|f(\mathbf{x}_k)| - C_{k-1})/Q_k, & C_{-1} = 0. \end{array}\right\} \tag{5}$$

Here $\Delta \in [0, 1]$ is a parameter used in the averaging of the previous absolute function values. As Δ approaches 0, we give more weight to the most recent function values. Since there is no theory to guarantee convergence when using the approximate Wolfe conditions, one of the code's parameters allows the user to employ only the standard Wolfe conditions. But by default, the code uses the approximate Wolfe conditions when (3) holds since we observe greater accuracy and efficiency when these conditions are utilized. Alternatively, by setting the parameter AWolfe to false, the code initially computes points satisfying the usual Wolfe conditions until the following inequality is satisfied:

$$|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| \leq \omega C_k. \tag{6}$$

Thereafter, the code switches to the approximate Wolfe conditions.

2 Running the code

CG_DESCENT requires a parameter file CG.PARM, which should be placed in the same directory where the code is run, and subroutines to evaluate the function $f(\mathbf{x})$ and the gradient $\nabla f(\mathbf{x})$. The arguments of the subroutine are the following:

1. GRAD_TOL (double) – specifies the desired accuracy in the solution. If STOPRULE in CG.PARM is true, then the code terminates when

$$\|\nabla f(\mathbf{x})_k\|_\infty \leq \max\{\text{GRAD\_TOL}, \text{STOPFAC} * \|\nabla f(\mathbf{x}_0)\|\}, \tag{7}$$

   where $\|\cdot\|_\infty$ denotes the sup-norm (maximum absolute component of the vector). If STOPRULE is false, then the code terminates when

$$\|\nabla f(\mathbf{x})_k\|_\infty \leq \text{GRAD\_TOL}(1 + |f(\mathbf{x}_k)|). \tag{8}$$

   The code also terminates when

$$-\alpha_k \phi'(0) \leq \text{FEPS}|f(\mathbf{x}_{k+1})|, \tag{9}$$

   where the default value of FEPS in CG.PARM is 0.d0.

2. X (double) – array of length n containing the starting guess on input and computed minimizer on output.

3. N (int) – problem dimension.

4. CG_VALUE (external) – name of the routine to evaluate the cost function $f(\mathbf{x})$. CG_VALUE (F, X, N) puts the value of the cost function in the double precision variable F, where X is a double precision array containing the vector $\mathbf{x}$.

5. CG_GRAD (external) – name of the routine to evaluate the gradient $\nabla f(\mathbf{x})$. CG_GRAD (G, X, N) puts the gradient of the cost function in the double precision array G, where X is a double precision array containing the vector $\mathbf{x}$.

6. STATUS (int) – the value indicates how the code terminates. As explained below, a nonzero value for status indicates abnormal termination.

7. GNORM (double) – if STEP in CG.PARM is .true., then GNORM contains a guess for the line search minimizer at $k = 0$; in other words, GNORM is the user's approximation to a value of $\alpha > 0$ that minimizes $f(\mathbf{x}_0 - \alpha \mathbf{g}_0)$. If STEP is .false., then GNORM is ignored at startup, and the code generates its own starting guess. On termination, GNORM contains $\|\nabla f(\mathbf{x}_k)\|_\infty$.

8. F (double) – value of $f(\mathbf{x}_k)$ at the final iteration.

9. ITER (int) – number of iterations that were performed.

10. NFUNC (int) – number of times the function was evaluated.

11. NGRAD (int) – number of times the gradient was evaluated.

12. D (double) – work array of length n containing the search direction.

13. G (double) – work array of length n containing the gradient.

14. XTEMP (double) – work array of length n containing $\mathbf{x}_k + \alpha \mathbf{d}_k$.

15. GTEMP (double) – work array of length n containing $\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)$.

The values of status and their meaning are list below:

0 – The convergence tolerance specified by GRAD_TOL was satisfied.

1 – Terminated with $-\alpha_k \phi'(0) \leq$ FEPS$|f(\mathbf{x}_{k+1})|$.

2 – The maximum number of iterations exceeded the limit MAXIT specified through CG.PARM.

3 – The slope $\phi'(\alpha)$ is always negative for a sequence of values of α becoming very large.

4 – The number of secant iterations during the line search exceeds the value of NSECANT (default 50) given in CG.PARM.

5 – The current search direction is not a direction of descent. According to the theory in [1, 2], the search direction should be a direction of descent for f.

6 – The line search has failed in the initialization part of the line search.

7 – The line search has failed in the bisection step.

8 – The line search has failed in the interval update routine.

We illustrate the use of CG_DESCENT with the problem:

$$\min \sum_{i=1}^{n} e^{x_i} - x_i\sqrt{i},$$

and the starting guess $x_i = 1$ for each i. The following code shows how to set up the problem and invoke the subroutine:

```
      INTEGER M
      PARAMETER (M = 100000)
      DOUBLE PRECISION X (M), D (M), G (M), XTEMP (M), GTEMP (M),
     &     GNORM, F
      INTEGER I, N, STATUS, ITER, NFUNC, NGRAD
      EXTERNAL MYVALUE, MYGRAD
      N = 100
      DO I = 1, N
         X (I) = 1.D0
      ENDDO
      CALL CG_DESCENT (1.D-8, X, N, MYVALUE, MYGRAD, STATUS,
     &     GNORM, F, ITER, NFUNC, NGRAD, D, G, XTEMP, GTEMP)
      END

      SUBROUTINE MYVALUE (F, X, N)
      DOUBLE PRECISION X (1), F, T
      F = 0.D0
      DO I = 1, N
         T = I
         T = DSQRT (T)
         F = F + DEXP (X (I)) - T*X (I)
      ENDDO
      RETURN
      END

      SUBROUTINE MYGRAD (G, X, N)
      DOUBLE PRECISION G (1), X (1), T
```

```
do i = 1, n
      t = i
      t = dsqrt (t)
      g (i) = dexp (x (i)) - t
enddo
return
end
```

The following output is generated when the code is run:

```
Termination status: 0
Convergence tolerance for gradient satisfied
absolute largest component of gradient: 0.7200D-08
function value: -653.07867273306
cg iterations: 31
function evaluations: 54
gradient evaluations: 43
```

The algorithm parameters are specified in the file cg.parm, which the code reads at the start of execution. Hence, this file should be placed in the directory where the code is run. A list of the parameters and their default values appears in Table 1. We now give an overview of these parameters:

- The maximum number maxit of iterations allowed by the code is maxit_fac*n, where n is the problem dimension. By default, maxit is 500*n. We also impose limits in the line search. The maximum number of secant steps is nsecant and the maximum number of expansions when we try to find an initial bracketing interval in the line search is nexpand.

- The code automatically computes an initial step for the very first conjugate gradient iteration. This automated guess can be crude. If the user wishes to provide the starting guess for a minimizer of $f(\mathbf{x}_0 - \alpha \mathbf{g}_0)$ over $\alpha > 0$, then set the parameter Step to .true., and in this case, the value of the gnorm argument of cg_descent should be the initial guess.

- If AWolfe is true, then the codes terminates the line search whenever either the ordinary Wolfe conditions (1) or the approximate Wolfe conditions (2) are satisfied along with (3). If AWolfe is false, then

Value	Parameter	Value	Parameter
.1d0	δ	1.0d0	RESTART_FAC
.9d0	σ	500.d0	MAXIT_FAC
1.d−6	ϵ	0.d0	FEPS
.5d0	θ	.7d0	QDECAY
.66d0	γ	50	NEXPAND
5.0d0	ρ	50	NSECANT
.01d0	η	.true.	PERTRULE
.01d0	ψ_0	.true.	QUADSTEP
.1d0	ψ_1	.false	PRINTLEVEL
2.d0	ψ_2	.true.	PRINTFINAL
1.d−12	QUADCUTOFF	.true.	STOPRULE
.0d0	STOPFAC	.true.	AWOLFE
1.d−3	AWOLFEFAC	.false.	STEP
		.false.	DEBUG

Table 1: Parameters in file CG.PARM and their default values.

the code initially computes points satisfying the usual Wolfe conditions until the the inequality (6) is satisfied. Thereafter, AWOLFE is set to true. By default, the code tests the approximate Wolfe conditions when (3) holds. The parameter ω in (6) is the same as the parameter AWOLFEFAC in the parameter file. To completely by-pass the approximate Wolfe conditions, the value of AWOLFE is set to false and AWOLFEFAC is set to 0.

- The parameter ϵ is used in (3) to obtain an estimate ϵ_k for the error in the function value. This estimate for the error governs when we use the approximate Wolfe conditions, and it enters into the update rules in the line search. The parameter Δ in (5) is the same as the parameter QDECAY in the parameter file.

- If PERTRULE is true, then we take $\epsilon_k = \epsilon C_k$ in (4). Otherwise, we take $\epsilon_k = \epsilon$

- If DEBUG is true, then in each iteration, we check whether $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + 10^{-10} C_k$, where C_k is generated in (5). When this inequality is violated, execution stops.

- By default, execution is terminated when (7) holds. By setting the parameter StopRule to false, execution terminates when (8) holds. The code also terminates when (9) holds. By default feps is 0.d0, and the condition (9) has no effect. The user may wish to terminate execution when the change in function value becomes negligible, in which case feps should be set to a small positive value, typically much smaller than the machine epsilon.

- By default, the code prints the results of the run, excluding the value of $\mathbf{x}$. To by-pass this printout, set PrintFinal to false. By default, the code delays all printing until the end of the run. To obtain detailed information concerning the line search and the convergence, set PrintLevel to true.

- As explained in [2], conjugate gradient methods preserve their n-step convergence property when the line search involves a "quadratic step." By default, the code attempts to make such a quadratic step. To deactivate this step, set QuadStep to .false. . The quadratic step is only attempted when the relative change in the function value for consecutive iterations is larger than QuadCutOff. If the relative change is tiny, then the quadratic step can be inaccurate, and it is skipped.

- The number nrestart of conjugate gradient iterations before performing the restart $\mathbf{d}_k = -\mathbf{g}_k$ is restart_fac*n. By default, restart_fac is 1 and nrestart is n.

- In computing an initial bracketing interval in the line search, we evaluate $\phi(\alpha)$ for a series of α's, each new value of α is ρ times its predecessor. The default value for ρ is 5. In some cases, however, it could be necessary to decrease ρ, while preserving the relation $\rho > 1$.

- The parameters δ, σ, θ, γ, and η are connected with the line search, termination conditions, and the formula for β_k. These parameters could be fine tuned to improve performance in some applications. The following inequalities should be maintained: $0 < \delta < .5$, $\delta \leq \sigma < 1$, $\eta > 0$, $\epsilon \geq 0$, $0 < \theta < 1$, and $0 < \gamma < 1$.

- The parameters ψ_0, ψ_1 and ψ_2 are all used in generating the initial stepsize in the line search, as explained in [2].

3 Trouble shooting

We now discuss the error messages and their possible cause. If the argument GRAD_TOL of CG_DESCENT is so small that the tolerance cannot be achieved (due to rounding errors in the evaluation of the function and its gradient), then the code can terminate in several abnormal ways. For example, the iterations could continue until the iteration limit MAXIT is reached; also, numerical errors in the line search might lead to termination. In the example given above, when we change the argument 1.d−8 to 1.d−20, we generate the following output:

```
Termination status: 4
Line search fails, too many secant steps
- your tolerance (grad_tol = 0.1000D−19) is too strict
absolute largest component of gradient: 0.1776D−14
function value: −653.07867273306
cg iterations: 217
function evaluations: 532
gradient evaluations: 707
```

Observe that the gradient is relatively small, however, we did not reach the requested tolerance 1.d−20.

The parameter FEPS in CG.PARM provides another mechanism to terminate execution when the code has essentially attained the highest possible accuracy. If we set GRAD_TOL to 1.d−20 and we change the value of FEPS in CG.PARM to 1.d−25, then the the following output is generated:

```
Termination status: 1
Terminating since change in function value ≤ feps*|f|
absolute largest component of gradient: 0.1910D−13
function value: −653.07867273306
cg iterations: 52
function evaluations: 75
gradient evaluations: 85
```

The default value for FEPS is 0.d0, in which case this termination condition has no effect.

The parameter ϵ in CG.PARM is used to obtain an estimate ϵ_k in (3) for the error in the function value. This estimate is used in the approximate

Figure 1: Rounding errors near a local minimum leading to an artificial hump in the numerical f

Wolfe conditions and in the update rules for a bracket interval in the line search. If ϵ_k is too small, then an error can arise in the line search near a local minimizer; numerically, we can have $\phi(\alpha) > \phi(0) + \epsilon_k$, while with exact arithmetic, $\phi(\alpha) < \phi(0)$. Hence, the code thinks the function looks like the graph depicted in Figure 1, when the actual function is monotone decreasing on the interval $[0, \alpha]$. In other words, the hump seen in Figure 1 may be due to rounding errors. This discrepancy between numerical function and true function leads to an error in the line search. For example, setting the parameter ϵ in CG.PARM to 0.d0 yields the following output:

TERMINATION STATUS: 8
LINE SEARCH FAILS
POSSIBLE CAUSES OF THIS ERROR MESSAGE:
- YOUR TOLERANCE (GRAD_TOL = 0.1000D−07) IS TOO STRICT
- YOUR GRADIENT ROUTINE HAS AN ERROR
- THE PARAMETER EPSILON IN CG.PART IS TOO SMALL
ABSOLUTE LARGEST COMPONENT OF GRADIENT: 0.8624D−07
FUNCTION VALUE: −653.07867273306
CG ITERATIONS: 29

FUNCTION EVALUATIONS: 103
GRADIENT EVALUATIONS: 90

The default value 10^{-6} of ϵ in CG.PARM is usually large enough to prevent this type of failure, except in cases where the function vanishes at the computed local minimum. When the function vanishes, the first form of ϵ_k in (3) approaches zero as the iterations convergence, while the actual error typically does not approach zero. In this case, where the function vanishes at the local minimum, you may need to use the second form for the error, which is activated by setting to .true. the parameter ERULE in CG.PARM. This problem connected with the estimation of the error in function value arises only when a high accuracy solution is computed. In the example just given, where we set $\epsilon = 0.d0$, we still computed a solution for which the absolute largest component of the gradient is less than 10^{-7} and the computed cost is correct to 14 significant digits.

If the code to evaluate the gradient of the cost function has an error, then the line search can fail. For our previous example, changing the minus sign to a plus sign in the code to evaluate the gradient yields:

TERMINATION STATUS: 6
LINE SEARCH FAILS
POSSIBLE CAUSES OF THIS ERROR MESSAGE:
- YOUR TOLERANCE (GRAD_TOL = 0.1000D−07) IS TOO STRICT
- YOUR GRADIENT ROUTINE HAS AN ERROR
- THE PARAMETER EPSILON IN CG.PART IS TOO SMALL
ABSOLUTE LARGEST COMPONENT OF GRADIENT: 0.1272D+02
FUNCTION VALUE: −399.63436462248
CG ITERATIONS: 1
FUNCTION EVALUATIONS: 53
GRADIENT EVALUATIONS: 52

One way to ensure that the gradient routine is correct is to use the ADIFOR software to automatically transform the routine for evaluating the cost function into a routine for evaluating the gradient. Alternatively, if your gradient routine is hand-coded, you can use finite difference approximations to check whether the code is correct. That is,

$$(\nabla f(\mathbf{x}))_i = \frac{f(\mathbf{x} + s\mathbf{e}_i) - f(\mathbf{x})}{s} + O(s), \tag{10}$$

where $\mathbf{e}_i$ is the i-th column of the identity matrix. By taking a sequence of s's approaching zero, the finite difference approximation should first approach the true gradient component, then diverge due to numerical errors connected with the evaluation of the numerator of (10). In the following code, we check the first component of the gradient in our model problem:

```
      PARAMETER (M = 100000)
      DOUBLE PRECISION X (M), G (M), F, NEWF,
     &     T, REL, DELTA, APPROX
      INTEGER I, N
      EXTERNAL MYVALUE, MYGRAD
      N = 100
      DO I = 1, N
         X (I) = 1.D0
      ENDDO
      CALL MYVALUE (F, X, N)
      CALL MYGRAD (G, X, N)
      DELTA = 1.E-1
      T = X (1)
      DO I = 1, 12
         X (1) = T + DELTA
         CALL MYVALUE (NEWF, X, N)
         APPROX = (NEWF - F)/DELTA
         REL = DABS ((APPROX - G (1)) / G (1))
         WRITE (6, *) DELTA, REL, APPROX, G(1)
         DELTA = DELTA/10
      ENDDO
      X (1) = T
      END
```

The output generated by this code appears in Table 2. Observe that for s between 10^{-1} and 10^{-7}, the relative error in the finite difference approximation decreases as it approaches the value of g (1), while for smaller s, the error increases. On the other hand, for the erroneous gradient code, obtained by replacing the minus sign in the gradient code by a plus sign, we obtain the results given in Table 3. Of course, you should check all components of the gradient, not just the first component.

S	Relative Error	Approximation	G (1)
0.100E+00	0.818E−01	0.18588419549E+01	0.17182818285E+01
0.100E−01	0.794E−02	0.17319186558E+01	0.17182818285E+01
0.100E−02	0.791E−03	0.17196414225E+01	0.17182818285E+01
0.100E−03	0.791E−04	0.17184177472E+01	0.17182818285E+01
0.100E−04	0.791E−05	0.17182954196E+01	0.17182818285E+01
0.100E−05	0.807E−06	0.17182832153E+01	0.17182818285E+01
0.100E−06	0.212E−06	0.17182821921E+01	0.17182818285E+01
0.100E−07	0.220E−05	0.17182856027E+01	0.17182818285E+01
0.100E−08	0.220E−04	0.17183197087E+01	0.17182818285E+01
0.100E−09	0.551E−04	0.17183765522E+01	0.17182818285E+01
0.100E−10	0.237E−02	0.17223555915E+01	0.17182818285E+01
0.100E−11	0.255E−01	0.17621459847E+01	0.17182818285E+01

Table 2: Output generated by correct gradient routine

S	Relative Error	Approximation	G (1)
0.100E+00	0.500E+00	0.18588419549E+01	0.37182818285E+01
0.100E−01	0.534E+00	0.17319186558E+01	0.37182818285E+01
0.100E−02	0.538E+00	0.17196414225E+01	0.37182818285E+01
0.100E−03	0.538E+00	0.17184177472E+01	0.37182818285E+01
0.100E−04	0.538E+00	0.17182954196E+01	0.37182818285E+01
0.100E−05	0.538E+00	0.17182832153E+01	0.37182818285E+01
0.100E−06	0.538E+00	0.17182821921E+01	0.37182818285E+01
0.100E−07	0.538E+00	0.17182856027E+01	0.37182818285E+01
0.100E−08	0.538E+00	0.17183197087E+01	0.37182818285E+01
0.100E−09	0.538E+00	0.17183765522E+01	0.37182818285E+01
0.100E−10	0.537E+00	0.17223555915E+01	0.37182818285E+01
0.100E−11	0.526E+00	0.17621459847E+01	0.37182818285E+01

Table 3: Output generated by erroneous gradient routine

References

[1] W. W. Hager and H. Zhang, *A New Conjugate Gradient Method with Guaranteed Descent and an Efficient Line Search*, Department of Mathematics, University of Florida, November 17, 2003 (revised July 10, 2004).

[2] W. W. Hager and H. Zhang, *CG_DESCENT, a Conjugate Gradient Method with Guaranteed Descent*, Department of Mathematics, University of Florida, January 12, 2004 (revised December 10, 2004).

[3] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.

[4] P. Wolfe, *Convergence conditions for ascent methods II: some corrections*, SIAM Rev., 13 (1971), pp. 185–188.