Tests with the Watson Function

Hans W Borchers

Feb 20, 2016

John, I really admire your `nlsr::nlfb` implementation, the only minimizing solver that found the true minimum of the Watson function in 12 dimensions.

```
lsWatson <- function(x) {
    n <- length(x)
    t <- (1:29)/29
    f <- numeric(31)
    for (i in 1:29) {
        f[i] <- sum( (1:(n-1)) * x[2:n] * (t[i]^(0:(n-2))) ) -
                sum( x * (t[i]^(0:(n-1))) )^2 - 1
    }
    f[30] <- x[1]
    f[31] <- x[2] - x[1]^2 - 1
    f
}

fnWatson <- function(x) {
    f <- lsWatson(x)
    sum( f * f )
}

grWatson <- function(x) pracma::grad(fnWatson, x)
```

Applying `nlfb` to the least-squares version returns a value very near to the best minimum I do know of, `4.72238e-10`.

```
x12 <- rep(0, 12)
sol12 <- nlsr::nlfb(x12, lsWatson, trace=FALSE)
```

```
## no weights
```

```
sol12[c("coefficients", "ssquares")]
```

```
## $coefficients
##  [1] -4.046194e-07  1.000002e+00 -5.615990e-04  3.477603e-01 -1.561673e-01
##  [6]  1.049947e+00 -3.238507e+00  7.271595e+00 -1.025134e+01  9.058734e+00
## [11] -4.534898e+00  1.010839e+00
##
## $ssquares
## [1] 4.724056e-10
```

The closest I could come to this value was with my new adaptive Nelder-Mead in package *pracma*, version 2.0.2, on R-Forge.

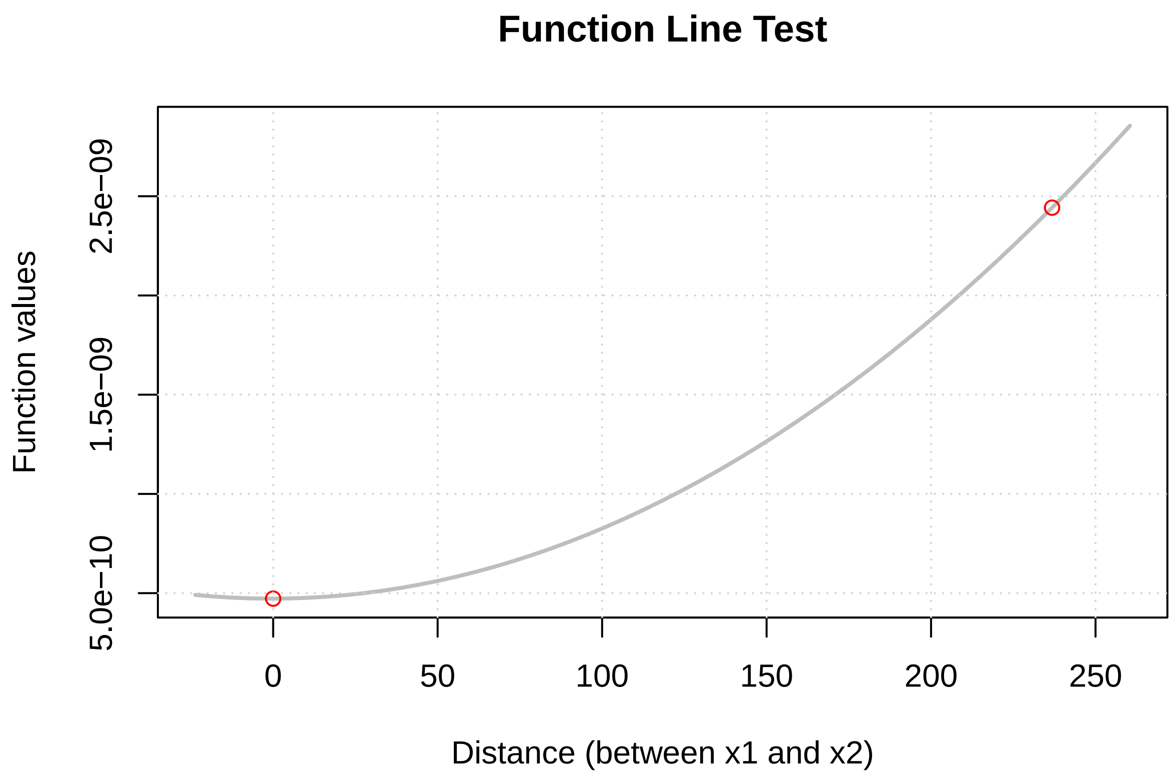
```
# library(pracma)  # v2.0.2 on R-Forge
(sol_anms <- pracma::anms(fnWatson, x12))
```

```
## $xmin
##  [1] -3.343786e-08  9.999944e-01  8.468975e-04  3.175546e-01  1.212238e-01
##  [6] -3.387601e-01  9.536750e-01 -7.041198e-01 -6.160913e-01  1.885323e+00
```

```
## [11] -1.532294e+00  4.700562e-01
##
## $fmin
## [1] 2.442405e-09
##
## $nfeval
## [1] 7202
```

We see that quite a different point has been found by `anms`. We can compare these by looking on the Watson function along a line connecting these two points.

```
adagio::flineviz(fnWatson, sol12$coefficients, sol_anms$xmin)
```



Interestingly, it never became so clear to me: though minimal values of found with different methods appear to be very close, the minimal points itself may be quite far away from each other.

I was not so lucky with applying your `opm` function.

When providing my own gradient of Watson, I get the following error:

```
sol <- optimrx::opm(x12, fnWatson, gr = grWatson, method = "ALL",
                    control = list(trace = 0))
```

```
Error in phev[npar] <= (-1) * kkt2tol * (1 + abs(fval)) :
  invalid comparison with complex values
```

On the other hand, when requesting a smaller tolerance I get:

```
sol <- optimrx::opm(x12, fnWatson, gr = "grcentral", method = "ALL",
                    control = list(kkttol = 1e-08, trace = 0))
```

```
Error in nlm(f = nlmfn, p = spar, iterlim = iterlim,
             print.level = print.level,  :
  probable coding error in analytic gradient
```

```
Gradient check details:
  max. relative difference in gradients=  1.9355

  analytic gradient: 0 0 -35.527 -71.054 -71.054 -71.054 -71.054
                     -71.054 -71.054 -71.054 -71.054 -71.054

  numerical gradient: 0 -60 -60 -61.034 -62.069 -63.115 -64.172
                      -65.241 -66.322 -67.413 -68.517 -69.631
Error in spg(par = spar, fn = efn, gr = egr,
             lower = slower, upper = supper,  :
  Analytic gradient does not seem correct! See comparison above.
  Fix it, remove it, or increase checkGrad.tol.
Successful convergence  Restarts for stagnation =0
Function evaluation limit exceeded -- may not converge.
```

Using the different possibilities for gradient strings, the following table is a compilation of results for the gradient variants "grcentral" and "grnd".

	grcentral	grnd
BFGS	5.9283e-08	9.7026e-08
CG	1.5004e-05	2.2420e-05
Nelder-Mead	7.0826e-06	7.0826e-06
L-BFGS-B	1.3255e-05	1.3389e-05
nlm	8.9885e+307	1.3521e-05
nlminb	1.5754e-07	2.6643e-09
lbfgsb3	1.3238e-05	1.3357e-05
Rcgmin	2.0707e-06	1.4500e-07
Rtnmin	1.0170e-06	2.5544e-07
Rvmmin	1.4440e-08	4.7224e-10
spg	8.9885e+307	2.8537e-05
ucminf	4.9441e-08	9.3362e-08
newuoa	1.6948e-07	1.6948e-07
bobyqa	2.0752e-07	2.0752e-07
nmkb	6.0361e-04	6.0361e-04
hjkb	6.4914e-06	1.3638e-07
hjn	5.3870e-06	5.3870e-06
lbfgs	1.8309e-07	1.5967e-07

Please take a closer look at this. For instance, `Rvmmin` with "grnd" finds a better minimum than `nlfb` above, but with "grcentral" the hit is not as good. Is "grnd" more accurate than "grcentral", or less? For its 'sister' method, `optim` with "BFGS", things happen vice versa.

I wonder what would happen if we plot all those minima points as a 'multidimensional scaling' figure (in two dimensions, of course), as all these points may be scattered around in space.