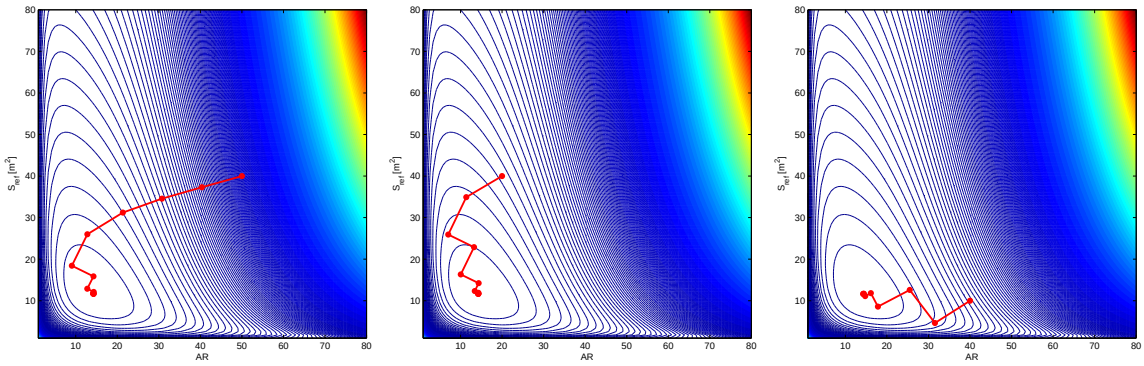
Report 3: So Many Dimensions

Rob Rau

February 26, 2014

What a Drag Part 2

Let's get straight to the point. The minimum drag value is $258.367N$ with an aspect ratio of 14.2816 and a reference planform area of $11.6682m^2$. Now for the specifics. There's not much to say about the steepest descent optimizer. It behaved as expected, exhibiting the zig-zag pattern as it converged on the solution. The pattern would depend on the start point of course. Start points that were farther out would sometimes head in a straight line as the gradient of successive points would be roughly in the same direction. This effect can be seen in fig. 1.



(a) Drag minimization with start point of $AR = 50$, $S_{ref} = 40$

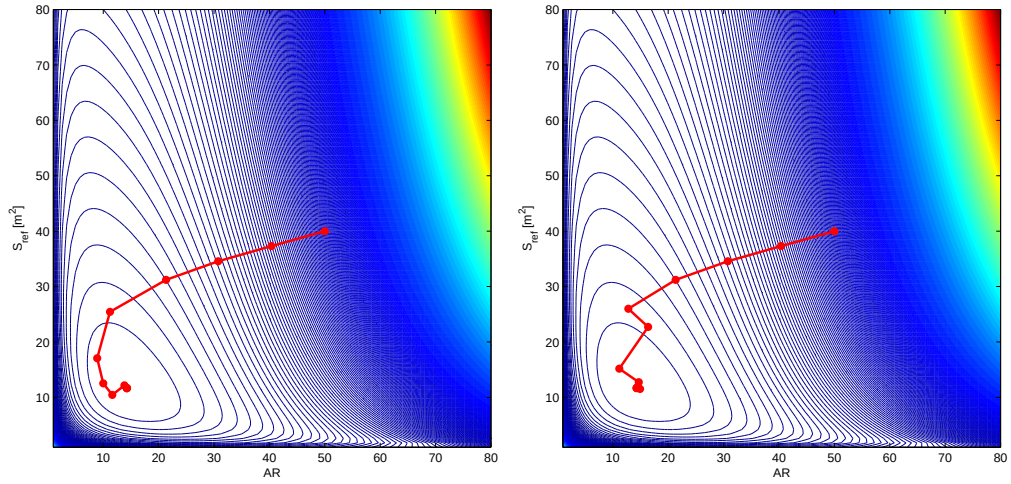
(b) Drag minimization with start point of $AR = 20$, $S_{ref} = 40$

(c) Drag minimization with start point of $AR = 40$, $S_{ref} = 10$

Figure 1: Comparison of start points for steepest descent method.

The conjugate gradient method I found to be a bit more interesting. I first coded up the conjugate gradient method using the Fletcher-Reeves update method. After some testing I wasn't particularly happy with this. It can be seen in table 1 through table 3 that the Fletcher-Reeves method wasn't performing much better than the steepest descent. I then decided to try the Polak-Ribière variant. I found that this method seemed to work better and more consistently that Fletcher-Reeves. I also did attempt the Dai-Yuan variant but could not get it to work at all. The paths of both methods can be seen in fig. 2

For the quasi-newton method, I chose to use BFGS. While this method seemed like the most complex of the three (four) I coded, it actually turned out to be the easiest to debug. As



(a) Conjugate gradient using Fletcher-Reeves update with start point of $AR = 50$, $S_{ref} = 40$

(b) Conjugate gradient using Polak-Ribière update with start point of $AR = 50$, $S_{ref} = 40$

Figure 2: Comparison of start points for steepest descent method.

expected BFGS converged much quicker than the other methods discussed. As will be seen later, this method also performed the best with a large number of variables. The path this method took can be seen in fig. 3.

Table 1: Drag minimization with start point of $AR = 50$, $S_{ref} = 40$

Optimizer	Major Iterations	Minor Iterations
Steepest Descent	13	273
Conjugate Gradient (Fletcher-Reeves)	14	292
Conjugate Gradient (Polak-Ribière)	12	282
BFGS Quasi-Newton	9	201

Table 2: Drag minimization with start point of $AR = 20$, $S_{ref} = 40$

Optimizer	Major Iterations	Minor Iterations
Steepest Descent	11	229
Conjugate Gradient (Fletcher-Reeves)	11	234
Conjugate Gradient (Polak-Ribière)	8	173
BFGS Quasi-Newton	7	158

Table 3: Drag minimization with start point of $AR = 40$, $S_{ref} = 10$

Optimizer	Major Iterations	Minor Iterations
Steepest Descent	11	230
Conjugate Gradient (Fletcher-Reeves)	9	191
Conjugate Gradient (Polak-Ribière)	10	212
BFGS Quasi-Newton	6	135

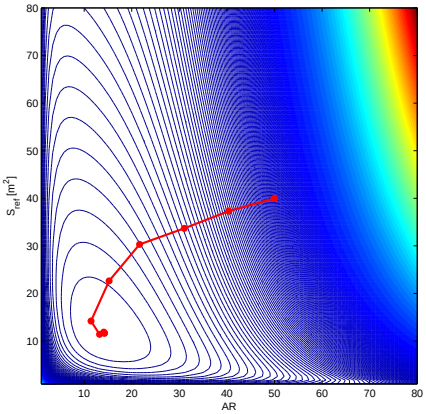


Figure 3: BFGS drag minimization path with start point of $AR = 50$, $S_{ref} = 40$

Convergence

The convergence of the various methods in the context of drag minimization can be seen in fig. 4. The interesting thing here is the convergence of the Polak-Ribière method. All the other methods exhibit expected behavior, I currently have no explanation for this. To add to the mystery, the Polak-Ribière method seemed to perform better than the others.

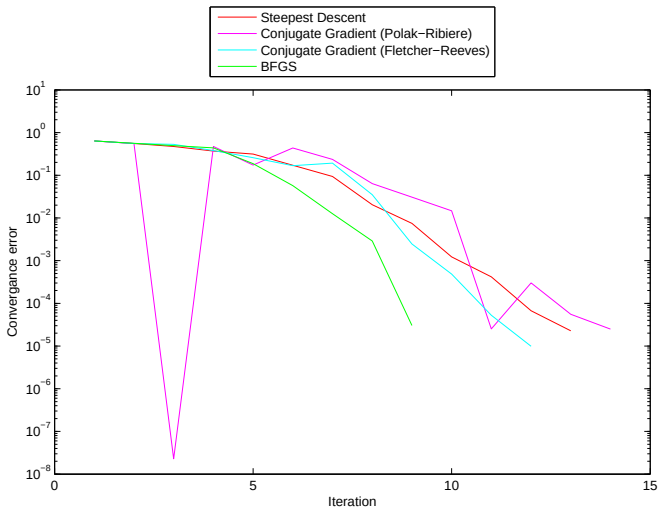


Figure 4: Convergence of various methods with start point of $AR = 50$, $S_{ref} = 40$

This may be an artifact of the specific problem. When looking at the convergence of the Rosenbrock function, the Polak-Ribière method behaves as expected. This can be seen in fig. 5.

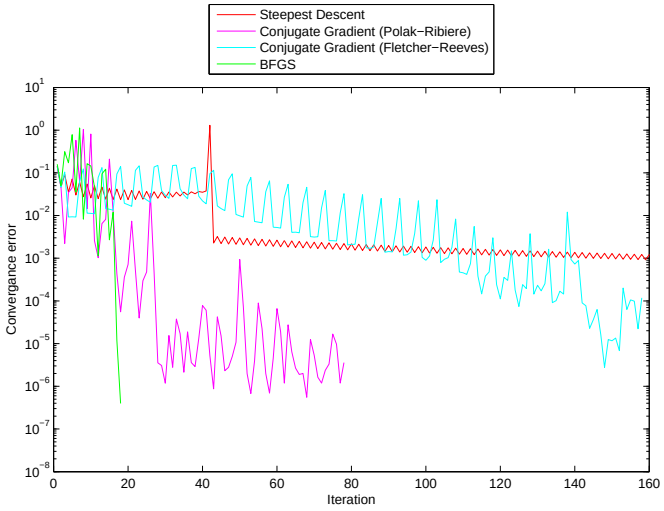


Figure 5: Convergence of various methods of the 2D Rosenbrock function

Effects of gradient precision

To test the effects of gradient precision, I decided to to switch my derivative method to finite differences as opposed to the complex step. I set the step size to something large ($1.0 \times 10^{-}2$) and let it go. On the drag minimization problem this didn't seem to have a huge effect. The results can be seen in table 4.

Table 4: Drag minimization using bad derivatives with start point of $AR = 50$, $S_{ref} = 40$

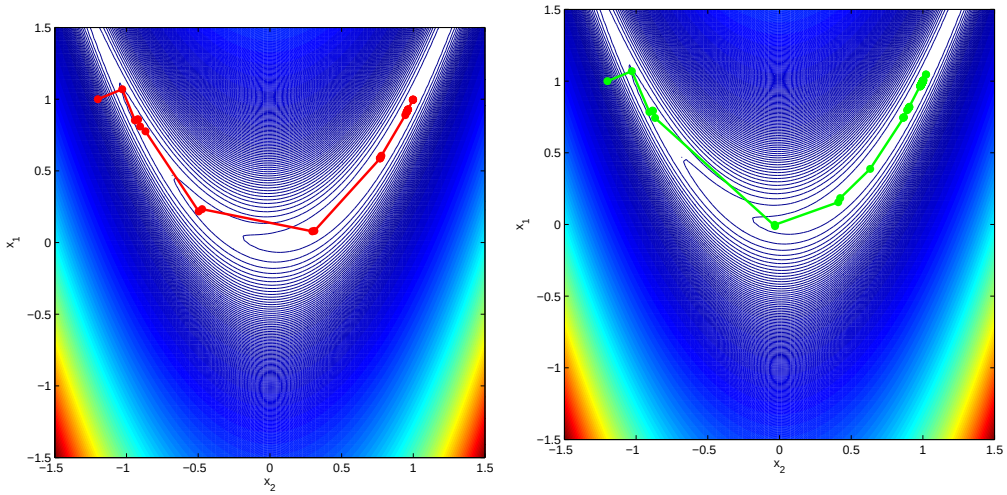
Optimizer	Major Iterations	Minor Iterations
Steepest Descent	13	273
Conjugate Gradient (Fletcher-Reeves)	12	260
Conjugate Gradient (Polak-Ribière)	12	260
BFGS Quasi-Newton	9	200

To make things strange, the Fletcher-Reeves conjugate gradient method actually ended up performing better.

As with the convergence, I found this effect to be more prominent and interesting when done to the Rosenbrock function.

As fig. 6 shows, the path taken by the optimizer is significantly different when using poor derivatives. It also took 3 orders of magnitude longer to converge!

The precision of the derivatives also has an effect on the solution it finds. The optimal point of the Rosenbrock function is at (1.0, 1.0) with a value of 0. Using the complex step, BFGS would converge to a solution of (1.0, 0.99999999) with a function value of 4.3913×10^{-12}. Using bad derivatives it converged to (0.981471, 0.963291) with a function value of 0.000343311. This is no good, we loose significant precision of our answer with this. This again is more prominent in the Rosenbrock function that it is with the drag function.



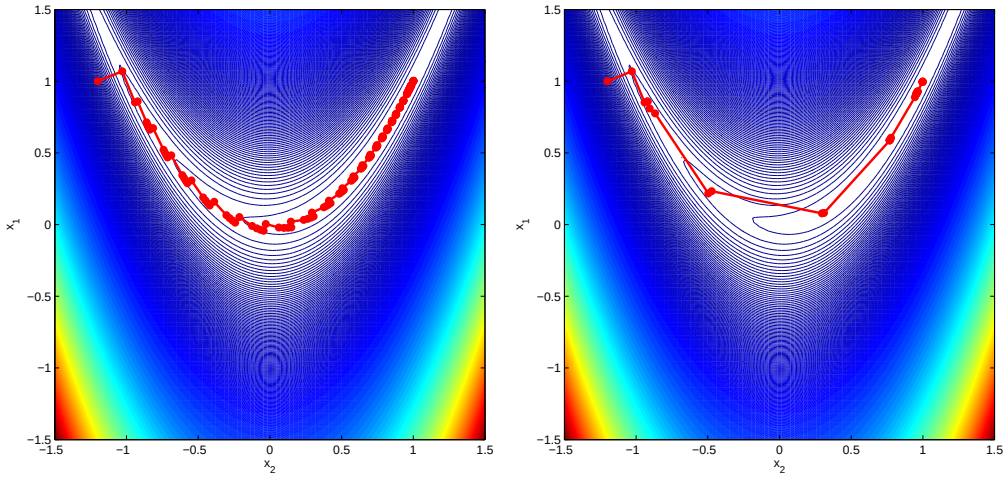
(a) Conjugate gradient using complex step derivatives. 53 major iterations

(b) Conjugate gradient using finite difference derivatives. 15197 major iterations

Figure 6: Comparison of different derivative methods.

Rosenbrockly and the Effects of Dimensionality

Working on this problem is when I realized I wasn't happy with the Fletcher-Reeves conjugate gradient method and decided to try the Polak-Ribière method for updating β. As can be seen in fig. 7 the Fletcher-Reeves conjugate gradient method seemed to take many small steps as it approaches the minimum. I played around with the parameters to try and improve it, but to no avail.

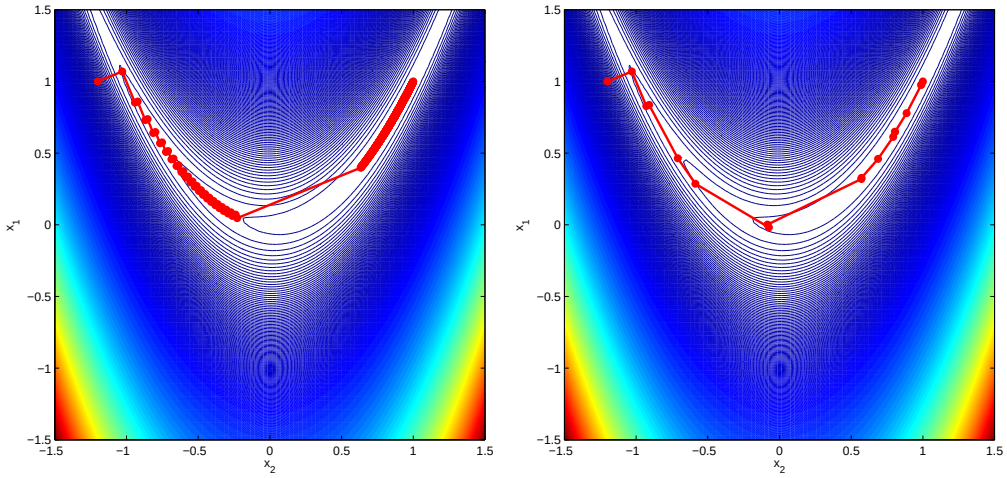


(a) Conjugate gradient using Fletcher-Reeves update.

(b) Conjugate gradient using Polak-Ribière update.

Figure 7: Comparison of different conjugate gradient methods.

On the other hand, the Polak-Ribière method take a far more direct path to the minimum. Both these methods use the exact same line search with the same parameters. The only difference is how β is updated. The other methods behaved as expected as seen in fig. 8.



(a) Steepest descent method on 2 dimensional Rosenbrock.

(b) BFGS method on 2 dimensional Rosenbrock.

Figure 8: Steepest descent and BFGS on 2 dimensional Rosenbrock function.

Upping the Anti

So how does increasing the number of dimensions effect the number of iterations needed to find the minimum? This didn't turn out quite as I expected it to. The results can be seen in fig. 9.

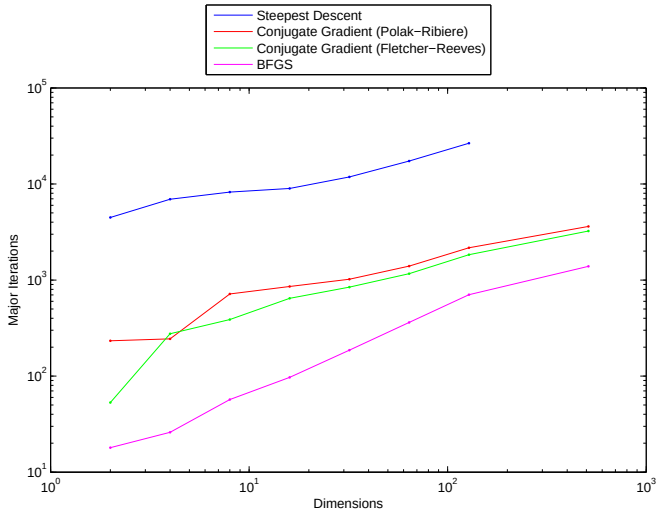


Figure 9: Number of major iterations required as number of dimensions increases.

These results were more or less the opposite of what I expected. The steepest decent method, while still requires a large amount of iterations to converge, it seems to scale much better than the other methods. If this trend is consistent to higher dimensions, the steepest descent might actually become the more efficient method. Unfortunately, due to time constraints, I was not able to test this to the extent that I wanted to. BFGS, while great for a low number of dimensions, does not scale as well as even the conjugate gradient methods.

This isn't the entire picture however. This analysis would be incomplete without looking at the effect on both the minor iterations and the total computation time. Interestingly fig. 10 shows that the number of required minor iterations scales almost exactly as the major iterations.

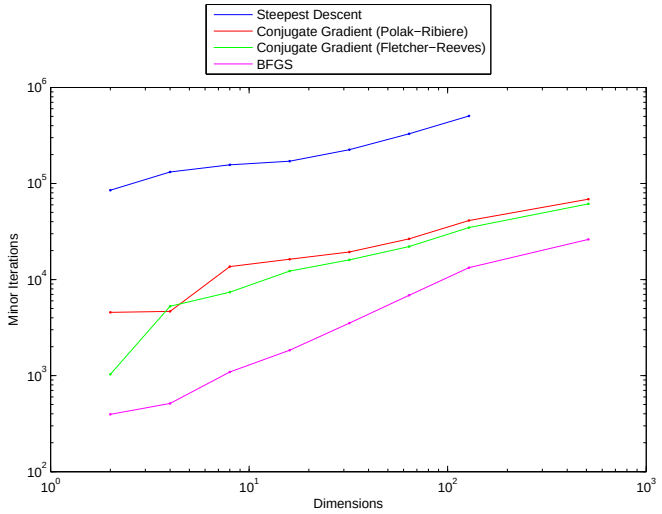


Figure 10: Number of minor iterations required as number of dimensions increases.

But if we look at fig. 11, we see that in terms of total computation time, all methods scale very similarly.

Unfortunately I did not have time to run the steepest descent for 512 dimensions.

This will only run to 8 dimensions in the turned in version.

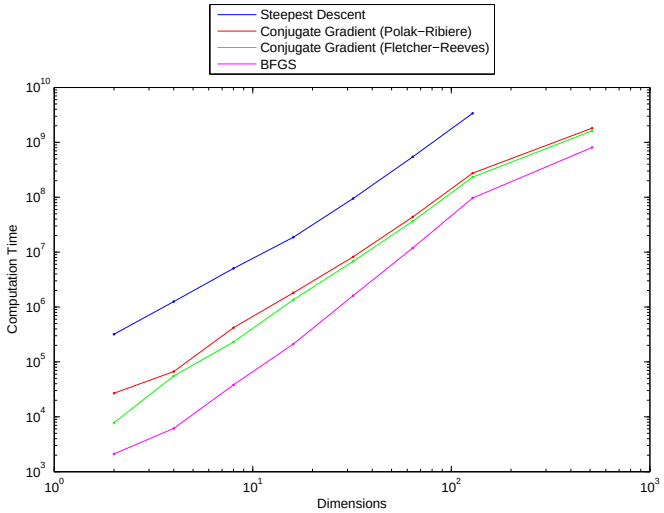


Figure 11: Total computation time as number of dimensions increases.

Performance

The BFGS quasi-newton method was far more effective than any of the other methods studied. As shown by the convergence, it performes very well, converging faster than anything else. Even when looking at the total computation time, BFGS scales just as well as any of the other methods.

The conjugate gradient method ended up garnering most of time though. Comparing both the Fletcher-Reeves method and Polak-Ribière it's clear that the choice of how you pick β has a significant impact on the effectiveness of the method.

With both the conjugate gradient method and the BFGS method, there was a lot of tuning that can be done in regards to how frequently it resets either β or the hessian update. This number seems to have a relatively large impact on the performance of the method.

The steepest descent method is, of course, the slowest method. Its performance does however seem to depend largely on the problem. In well conditioned problem the steepest descent method could perform very well, but as we know, that is rarely the case.