GROMACS - Bug #2641

Possible l-bfgs improvements

09/11/2018 03:52 PM - David van der Spoel

Status:	Closed		
Priority:	Normal		
Assignee:	David van der Spoel		
Category:	mdrun		
Target version:	2019.2		
Affected version - extra info:	everything after as well	**Difficulty:**	hard
Affected version:	5.1		

Description

Minimization of even small molecules to convergence is problematic in GROMACS. I am finding that energy minimization not always succeeds, even for very simple molecules such as 2-octanol or pentoxypentane. This leads to problems when you want to do a normal mode analysis based on that structure.

It would be useful to have a minimizer that **does** go all the way to zero force (steepest descents doesn't do it either).

To reproduce the problem, unpack the archive, then run

gmx_d grompp
gmx_d mdrun

It will finish with:

"Energy minimization has stopped, but the forces have not converged to the
requested precision Fmax < 0.001"

Related issues:

Related to GROMACS - Bug #1593: step-size scaling in lbfgs doesn't work as in...	**Closed**	**09/07/2014**

Associated revisions

Revision 23c57b58 - 04/05/2019 10:15 AM - Морозов Дмитрий

Fixes bug in l-bfgs.

l-bfgs should now work excatly as it was in 4.6.5,
up-to some numerical precision in forces and energies.

Fixes #2641

Change-Id: I11a85a4241cc933fef94e2095dd8c3cbbb28b01b

History

#1 - 09/12/2018 02:41 PM - Erik Lindahl

- Status changed from New to Rejected

I don't necessarily think this is an error, and in any case I'm pretty sure it's not in the minimizers.

First, for reference, although it sounds simple, it's not possible to just have a minimizer "go all the way to zero force" when using floating-point arithmetics. To understand this, it's easiest to think about the one-dimensional line minimzer step. Close to the minimum, the function we are minimizing will have a shape like delta-y = k * delta-x^2.

Since we need to evaluate the numerical derivative to decide how to move, we will be limited by machine precision (EPS) when evaluating delta-y close to the minimum - which in turn means we cannot get closer to the exact minimum than delta-x = sqrt(EPS). Since the derivative (=force) is also proportional to delta-x, that too will have a relative error proportional to the square root of machine precision.

However, the reason for not reaching very low energies in this case might is likely that you are using a plain shift of the function - which means there will be a discontinuity in the derivative at the cut-off.

If I change it to either use very long cut-offs (tested with group as well as verlet) or using switched interactions (only available with group for now), I am able to reach a norm of the force that is 6e-4. This seems to be quite close to a bond constant multiplied by sqrt(EPS).

So: Don't use shifted interactions for good energy minimization!

#2 - 09/12/2018 02:50 PM - Berk Hess

- Status changed from Rejected to Feedback wanted

Gerrit just mailed me that someone in his group has some improvements for bfgs. Lets hear what he has to say.

As I said in my reply on gmx-developers, I have heard from several people that one or more of the minimizers used in GROMACS work better in other software.

#3 - 09/12/2018 03:16 PM - Erik Lindahl

Just remember that we don't use a minimizer in isolation.

The Gromacs minimizers seem to work just A-OK when used with conservative potentials where both the potential and force are continuous. I too would of course be interested in any changes that also make them better when this is NOT the case, but if they don't use that in the other code it's not a very useful comparison.

I forgot to mention: With the shifted interaction the norm stops at ~3e-1, while it goes to 4e-4 with smooth interactions, so here the big difference is definitely due to the cutoff setup.

#4 - 09/12/2018 07:31 PM - David van der Spoel

- Status changed from Feedback wanted to Rejected

Well that's embarrassing! Thanks for debugging my mdp file. I will close this one then.

#5 - 09/12/2018 07:31 PM - David van der Spoel

- Status changed from Rejected to Closed

#6 - 09/12/2018 10:59 PM - Mark Abraham

- Subject changed from Reliable accurate minimization to Possible l-bfgs improvements

- Status changed from Closed to Feedback wanted

#7 - 09/30/2018 08:47 PM - Erik Lindahl

I'll let this be open another 2-3 weeks in case there is any feedback, but if it stays silent I'll close it ;-)

#8 - 03/25/2019 11:27 AM - Dmitry Morozov

- File wat-2-tip3p.zip added

Hi,

I've identified the problem with current L-BFGS implementation.
Seems like it has been broken at some point completely.

In the attachments there is a simple case of system with just two TIP3P Water molecules. No constraints or something like that.
If you will do the optimization with L-BFGS, then you could not find a minimum like at all with Gromacs 2019. CG still works fine.

I've tested that input on Gromacs 2018, 2019 and old one 4.6.5
In Gromacs 4.6.5 L-BFGS works correctly. In 2018 and 2019 it does not work, so probably something happened in the process of transition from C to C++.

I trying to debug it right now, using comparison with the older 4.6.5 results. While problem is not resolved, I suggest to turn L-BFGS Off in release version.

#9 - 03/25/2019 11:42 AM - Berk Hess

- Category set to mdrun

- Target version set to 2019.2

Great! With such a simple testcase we should be able to fix this for 2019.2.

#10 - 03/25/2019 01:02 PM - David van der Spoel

I have reproduced the issue. The minimization works fine in 4.6 but from 5.1 it is broken. Have not tested 5.0 but will run git bisect now.

#11 - 03/25/2019 01:06 PM - Christian Blau

- Tracker changed from Feature to Bug

- Affected version - extra info set to everything after as well

- Affected version set to 5.1

#12 - 03/25/2019 04:53 PM - David van der Spoel

Git bisect points to I54f238cedc78aadb0dad080ea3b28f001dce8d94

#13 - 03/25/2019 05:55 PM - David van der Spoel

I fixed it. Do we want it in 2018 as well?

#14 - 03/25/2019 06:00 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2641.
Uploader: David van der Spoel (spoel@xray.bmc.uu.se)
Change-Id: gromacs~release-2018~I6e58aeed09606819fd24078407b7c05639a1af07
Gerrit URL: https://gerrit.gromacs.org/9354

#15 - 03/25/2019 09:51 PM - Mark Abraham

David van der Spoel wrote:

> Git bisect points to I54f238cedc78aadb0dad080ea3b28f001dce8d94

Which is 4a6c1455bb9b0a899a1fcd4845aaa8a24cfed0a4 at https://gerrit.gromacs.org/#/c/4010

Erik and I collaborated on that change, so I'm sure he will have further thoughts on this issue.

Clearly we need to add test cases or to stop pretending we support these things.

#16 - 03/25/2019 09:54 PM - Mark Abraham

- Related to Bug #1593: step-size scaling in lbfgs doesn't work as intended added

#17 - 03/25/2019 10:20 PM - David van der Spoel

In fact there is a test in master now, which presumable will fail.

#18 - 03/26/2019 09:54 AM - Dmitry Morozov

Why there are such a strange things like:
stepsize = 1.0/fnorm;
in the Gromacs l-bfgs code?

I mean stepsize is just a scaling factor of the real step, which we want to take.
stepsize = 1.0 means we are trying to make a full displacement s[] as a step.
We are searching for a suitable stepsize between 0.0 and 1.0 in the procedure of line search.
For scaling purposes there is already controllable parameter: inputrec->em_stepsize.
I hardly could imagine then what is the purpose of stepsize = 1.0/fnorm in that case.
So I suggest to remove this from everywhere just reset it to stepsize = 1.0

#19 - 03/26/2019 10:21 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2641.
Uploader: David van der Spoel (spoel@xray.bmc.uu.se)
Change-Id: gromacs~release-2019~I6919c790b2728ffd7f31725d929baadb7d752be9
Gerrit URL: https://gerrit.gromacs.org/9358

#20 - 03/26/2019 10:53 AM - Dmitry Morozov

Hi I've found another bug in the minimize.cpp: line 2166
It is now:
if (std::fabs(sb->epot - Epot0) < GMX_REAL_EPS || nminstep >= 20)

It should be
if (std::fabs(sb->epot - Epot0) < GMX_REAL_EPS)

Because even if we have performed all 20 line search steps and did not found exact minimum along the line, doesn't mean that we have no energy improvements with respect to the starting point!
Tested it against CG optimization on the same 2 Waters input.

#21 - 03/26/2019 11:50 AM - David van der Spoel

Thanks for pointing that out. Interestingly, this bug does not show up in single precision but it does in double precision. This is likely due to the minimizer converging to sufficient precision in single faster.

#22 - 03/26/2019 12:18 PM - Dmitry Morozov

Yes exactly, I've also noticed that it affects only double precision, because line search never reaches 20 steps in single precision (on that test system of course). But anyway this is a nasty bug.

Actually, line-search procedure in L-BFGS and, potentially, in CG could also be improved a lot by applying Wolfe conditions: https://en.wikipedia.org/wiki/Wolfe_conditions

I'll try to prepare an updated version of L-BFGS minimizer in a month or so.

#23 - 03/26/2019 12:37 PM - Erik Lindahl

Gentlemen,

Let's take a step back and work on one issue (or not) at a time and properly diagnose exactly what it is, what the code says, whether that disagrees with the algorithm, and if so what the change should be. I would not take my own original implementation as a reference, and if we want to revert some of the later changes we should understand why they were wrong instead of just kicking the can and adding something that works for one particular system.

It was a LONG time ago that I looked at this code, but here are some thoughts from a quick look at the code:

1. If you search the do_lbgfs() routine in minimize.cpp, you'll see that inputrec->em_stepsize is only used as a check for the upper limit for any displacement. It is never used to scale any other step size.

2. s[] is the search direction, which initially is just the force gradient. The comments for this appears to have been better prior to the C++ change, but note that this is a **non-normalized** direction, so we most definitely want to scale it initially at least (which we still do, since David did not change it in the other places).

3. Now, if I have to guess (reading your own old C code is really difficult when we're used to well-documented C++...) I suspect it might be related to the fact that the search direction in later steps is not simply the force gradient, but we tried to be "smart" and avoid the cost of properly normalizing the vectors.

I think a proper solution might be to simply normalize all direction vectors each time we calculate them. I can offer to work through the code if someone can test the different code versions with ~10 different (small and large!) test systems and check if all of them worked fine in version 4.6!

As for the comment about line 2166, this statement is inside the conditional starting on line 2080, which should mean the case you describe cannot happen. Second, even if it were to happen, the only thing the conditional would do is to reset the search direction memory (which is anyway recommended to do every 20-100 steps), so in worst case it would only make the convergence slightly slower - it should never be able to cause a failed minimization.

#24 - 03/26/2019 12:54 PM - David van der Spoel

Great, this would of course be much better.

I am however not convinced by your statement in gerrit that l-bfgs works for many systems. If it does not work for two water molecules it is broken even if you may have one or more "counter examples".

And in my tests the removal of the conditional is needed to make sure that the water dimer converges in DP. Without it the minimization just starts over from scratch many times.

So with that comes the question, implement a hack now or disable it?

#25 - 03/26/2019 01:16 PM - Dmitry Morozov

OK, I've looked through the whole code again, so

1) We have direction of search given by the array s[] which is also defined as pointer to dx[point][] array (line 1938)

2) We want to find suitable step (step_taken variable) along that direction which will minimize energy. That step_taken variable is used as a scaling factor of dx[point][] array to make an actual step (lines 2227-2231)

3) step_taken variable is chosen between three values denoted in the code as variables a, b and c (lines 2195-2215). Implying that they all are scaling factors of array s[] (dx[point][])

4) At the begining of line search (line 2005) variabe a = 0.0. And c is setted up to c = a + stepsize (line 2013). Hereby if we set stepsize to be 1.0 means that c will be equal to c = 0.0 + 1.0 = 1.0 meaning that we trying full Quasi-NR step along s[] direction

5) Also in the cycle lines 2010-2032 we are ensure that our c*s[] will not produce displacement in any of coordinates large than inputrec->em_stepsize, this is done by scaling stepsize (line 2029) and recalculating c (line 2013)

I did not found any other purpose of variable stepsize in the code rather than (5), where it is already scaled down using inputrec->em_stepsize as a

criteria.

P.S. I'm strongly disagree with the Erik's point (2). s[] (dx[point][]) is not just a search direction. It is rather Quasi-NR step given by the approximation of NR equation s = -(H^-1)*g and updated each step using L-BFGS scheme (lines 2233-2321)

#26 - 03/26/2019 04:18 PM - Erik Lindahl

Whether you call something "Quasi-NR step" or search direction is really just semantics. In terms of the **line minimization**, that's a process that happens along a line specified by a vector, and what happens outside that process is completely irrelevant for the line minimizer (then it's a separate matter that we historically added a few more tweaks to the line minimizer in L-BFGS because the method is capable of finding very good minima).

Everything in the discussion above has focused on the line minimizer, so if we for now assume the problem lies there, we can forget everything about the outer L-BGFS algorithm and focus on the line minimizer. To the line minimizer, the ONLY relevant thing from the outer loop is to be provided with **search direction** - but whether that is provided from steepest descent, CG, BFGS, or "divine inspiration" is irrelevant :-)

To select that line we need a **direction**. In every sane sense of mathematics that means we should be a normalized vector, but because we were doing stupid optimizations 2 decades ago we didn't do that properly - mea culpa. In the very first step the solution of instead normalizing the stepsize works fine, but something might go wrong for the latter iterations.

There is nothing more special with a stepsize of 1.0 than 12342.234624362, so that is not an argument - in particular not if we are doing monkey business by combining it with line minimization along something that is likely not properly normalized. It's likely just luck/bad lack when some systems work but others fail - which is the reason I don't just want to hack it back to the old value, because I think that one is wrong too.

#27 - 03/26/2019 05:01 PM - Dmitry Morozov

OK, I agree that this is just semantics.

Lets then normalize s[] with ||s|| = snorm.
Lets also define smax as a maximum stepsize which will satisfy the following condition: smax*s[i] <= inputrec->em_stepsize for all (i) coordinates of the system.
Now, we could initialize our lower boundary of line search to a = 0.0 and upper boundary to c = min(snorm, smax). And now we could do line search for minima on the [a;c] interval in the direction of s[].

But, in reality that renormalization will not cause any differences in actual algorithm, because we always could reverse it back at any point by multiplying back s[] with snorm and dividing a, b and c by snorm. And the outcome of line search procedure would be the same.

#28 - 03/26/2019 11:20 PM - Erik Lindahl

Oh, the normalization will also enter for things like calculating the scalar gradient, and while the code does account for the non-normalized vector, I'm less convinced than you about the 20-year-younger Erik's ability to be 100% consistent there :-)

Actually, the range and step size shouldn't matter at all. A line minimizer should be insensitive to the initial step size - which it has to be, because in general we have no idea about the shape of the function. So, at least in theory it should be perfectly fine to either multiply or divide by any norm we want (as long as it's not NaN). The only thing that should happen with a stupid choice is that we would need a bunch of extra steps when the stepsize is scaled up/down (just slowing down convergence).

For some reason that is not the case, which is obviously an indication there's an "undocumented feature" somewhere - and that's what we need to understand. I don't want to just set it to a different arbitrary step size that worked in the past for some system - because to me that's a strong indication we haven't identified the bug, but rater randomly try to paint it over.

Now, what makes the line minimization implementation a little bit more complicated here than for the other methods is that L-BFGS was the first minimizer where we spend efforts on actually getting VERY close to the theoretical machine precision. However, as I've commented in other threads that per se can be very challenging for large systems due to the quadratic form limiting the resolution in the line minimizer to sqrt(EPS), so when we're getting close to the minimum we try to apply a few heuristics (such as continuing just a bit further as long as the gradient is negative, even if the potential is just-so-slightly higher). Some of those things could lead to artefacts depending on the system size.

Actually, given that we are completely ANAL about all numerical precision stuff nowadays I'm 100% certain there are a bunch of things we should change (such as Berk's comment), but those are tasks for a rewritten module, not a fix in the release branch.

To fix it in 2019.2, we need to understand exactly where the line minimizer goes wrong and why.

#29 - 03/27/2019 09:16 AM - Dmitry Morozov

Yes, That's why I've spent last evening to check other implementations of L-BFGS in the following library:
https://github.com/chokkan/liblbfgs
and also original Nocedal implementation of it
http://users.iems.northwestern.edu/~nocedal/lbfgs.html

In both cases they indeed scale linesearch stepsize to 1.0/gnorm for the first iteration, which is steepest descent in the direction of gradient. However on the following iterations they always initialize linesearch stepsize with the 1.0 . And I couldn't find any normalization procedure for gradient or search direction. In both cases they use non-normalized vectors.

P.S. Also they both use by default a different, and much more complicated, linesearch algorithm called More-Thuente linesearch. It uses quadratic or cubic (depending of the points relative energies and gradients) polynomial interpolation to minimize function along the line. Current implementation in Gromacs uses simple linesearch with linear interpolation to 0 line gradient.

#30 - 03/29/2019 12:37 PM - Dmitry Morozov

- File minimize.cpp added

I've made L-BFGS in master branch to behave like an old 4.6.5 (up to some numerical precision)
minimize.cpp with changes attached to the message, for some reason I could not submit directly to Gerrit.
So maybe David could do that?
I can also create pull request on Github.

#31 - 03/29/2019 01:37 PM - David van der Spoel

Thanks, but there are way too many unrelated changes in this file to be acceptable. We need to find just the two or three lines that have to fixed.

#32 - 03/29/2019 02:29 PM - Dmitry Morozov

- File minimize_new.cpp added

Ahh, OK seems like I've used a wrong branch.
Here is your minimize.cpp file from the patch on Gerrit.
I've just added my fix to it.

#33 - 04/03/2019 02:38 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2641.
Uploader: Dmitry Morozov (aracsmd@gmail.com)
Change-Id: gromacs~master~I11a85a4241cc933fef94e2095dd8c3cbbb28b01b
Gerrit URL: https://gerrit.gromacs.org/9393

#34 - 04/05/2019 10:30 AM - Anonymous

- Status changed from Feedback wanted to Resolved

Applied in changeset 23c57b58a9a6ad7d12aee776090139282a5eece9.

#35 - 04/05/2019 04:27 PM - Mark Abraham

- Status changed from Resolved to Closed

Files

2-octanol.zip	280 KB	09/11/2018	David van der Spoel
wat-2-tip3p.zip	4.48 KB	03/25/2019	Dmitry Morozov
minimize.cpp	101 KB	03/29/2019	Dmitry Morozov
minimize_new.cpp	100 KB	03/29/2019	Dmitry Morozov