GROMACS - Bug #172

incorrect handling of angle bending leads to errors for obtuse angles and crashes for angles of 180

10/17/2007 12:59 AM - David Mobley

Status:	Closed		
Priority:	Normal		
Assignee:	David van der Spoel		
Category:	mdrun		
Target version:	3.3.1		
Affected version - extra info:		**Difficulty:**	uncategorized
Affected version:			

Description

Earlier, I wrote to the users list about how I ran a large set of small molecules using GAFF parameters and all of my small molecules with triple bonds (nitriles, pentyne, propyne, etc) were crashing. This was not a parameter problem, and not a timestep problem, and, perplexingly, only seems to occur for these molecules when they are in vacuum or water, but not in a binding site.

As it turns out, we (John Chodera, Chris Fennell and I) have now traced the problem back to problems with the angles, and in particular, how GROMACS handles angle bending for nearly linear molecules. These molecules, since they have triple bonds, are basically linear, of course.

Anyway, as it turns out, all of these molecules have one or several angles with preferred angles of nearly pi (or 180, whichever units you prefer). As it turns out, the angle bending code apparently can't handle angles of exactly 180 degrees, which causes the problem. This also explains why these molecules are OK when run in a binding site: The binding site bends the angle so it isn't 180.

In gmxlib/bondfree.c, the relevant algorithm (in pseudocode) is something like this (where
x_i, x_j, and x_k are atom vectors):

t_1 = x_i - x_j
t_2 = x_k - x_j
cos_theta = dot(t_1, t_2) / (|t_1| |t_2|)
theta = inverse_cosine(cos_theta)
V = (1/2) K (theta - theta_0)^2

Now, if theta_0 is anywhere near pi, this obviously isn't going to work if inverse_cosine is defined on the domain [-pi,+pi). Hence, the problem we're seeing.

It's also worth noting that the above routine leads to **incorrect forces** for obtuse angles, also.

The solution is simple, as John Chodera points out to me: shift this inverse_cosine domain to cover a range of [-pi,+pi) around theta_0. Then only in the (extremely rare) case that a molecule is so strained that it is pi away from its preferred value would there be instabilities.

I can submit a "test molecule" if it would be helpful, but since this is just an algorithmic thing I think the problem should be clear even without a test case.

History

#1 - 10/17/2007 08:23 AM - David van der Spoel

If the solution is simple then please provide code or at least equations or preferably both.

#2 - 10/17/2007 08:59 AM - John D.

As far as my meagre brain can grasp, the angle energy terms and gradient are computed in gmxlib/bondfree.c, in a function called 'angles'. The current torsion angle is computed by a call to bond_angle(), which computes the cos of the angle between the ij and ik bond vectors, and then calls 'acos' to compute the inverse cosine, which returns the angle theta in the range [0,pi]. The energy and gradient is then computed by a call to 'harmonic', which simply computes the linearly interpolated harmonic contributes by computing the deviation from the reference angle (let's call it theta0).

Keeping either of these angle terms in David's .top file would cause David Mobley's vacuum calculations to segfault:

[angles]

```
;  ai    aj    ak funct  theta   cth
2     1     4     1  1.7838e+02  3.7489e+02 ;
1     2     3     1  1.7799e+02  4.7196e+02 ;
```

Note that both reference angles are nearly 180 degrees. They run fine in AMBER, apparently.

The problem is more subtle than I first appreciated. At first, I thought it was simply that the range of the result of acos(cos_theta) would need to be shifted, but once theta opens up to pi, it is impossible to tell that it is continuing to open up larger than pi, and it looks like the angle is shrinking again.

AMBER seems to ensure that cos_theta remains on the domain [-0.999,+0.999] before feeding it to acos(cos_theta) by clipping. I imagine that numerical errors in the computation of cos_theta in gromacs might be causing values that exceed the defined domain to be fed to acos(), and that could be causing the observed problems, but I haven't experimented to figure this out.

I apologize for my earlier remarks that this bug could be fixed by just shifting the range of the output of acos() -- it looks to be more subtle, and hopefully simply a matter of ensuring the input to acos() remains in this domain.

It should also be noted that the derivative is formally discontinuous at theta = pi here, but I don';t think there's any way to fix that except to move to force terms that are harmonic in cos(theta) rather than theta, but that's a forcefield problem rather than a numerical stability of the implementation problem.

I'll upload David Mobley's example in a subsequent mail.

Cheers,

- John

#3 - 10/17/2007 09:08 AM - John D.

Created an attachment (id=250)
propyne in vacuum test case from David Mobley.

Added propyne in vacuum test case from David Mobley. Unpack and run ./run0.sh to run, which should crash during dynamics phase. The first and last members of the [angles] section seem to be responsible for the crash.

#4 - 10/17/2007 12:01 PM - David van der Spoel

> AMBER seems to ensure that cos_theta remains on the domain [-0.999,+0.999]
> before feeding it to acos(cos_theta) by clipping. I imagine that numerical
> errors in the computation of cos_theta in gromacs might be causing values that
> exceed the defined domain to be fed to acos(), and that could be causing the
> observed problems, but I haven't experimented to figure this out.

So the real problem is what to do when cos_theta falls outside the domain! Suggestions? What does AMBER do in these cases? Set the force to zero? In that case it probably becomes discontinuous.

It's not that we haven't thought of this before, we have, but for proteins it is not a problem.

#5 - 10/17/2007 04:40 PM - David Mobley

I've been doing a little more testing, and, perhaps not surprisingly, this all works basically fine in double precision, presumably because the extra precision means that cos_theta never hits exactly +1 or -1. Of course, that's not really a solution if you want to be able to use single precision.

Maybe clipping as AMBER does would be a good solution.

#6 - 10/17/2007 07:25 PM - John D.

Hi guys,

Could you simply use double-precision accumulators for the computation of cos_theta and acos(cos_theta)? If David Mobley is right and cos_theta never hits exactly +1 or -1, then this should fix the problem.

Clipping, as in AMBER, doesn't seem like the optimal solution, though it at least avoids the problem of feeding acos() an argument outside of its defined domain.

Cheers,

- John

#7 - 10/17/2007 07:28 PM - Erik Lindahl

But will double precision really solve it, rather than just making it a couple of orders of magnitude more improbable?

Cheers,

Erik

#8 - 10/17/2007 07:38 PM - David Mobley

Erik, I worried about that too. What I can say at this point is it seems more than a couple orders of magnitude better in double precision, in that before it was crashing reliably in single precision within a few hundred ps and so far I've run ~30 5 ns simulations in double precision with no crashes.

My intuition is that you are correct and this is not really a fix, but would simply make the problem a lot more rare. Maybe rare enough, though (it seems to be making it rare enough for my purposes).

As John points out, clipping is also less than ideal, though.

My suggestion is to either:
(a) Go with some sort of clipping because it's what AMBER does and at least prevents the problem
(b) Go to double precision here and put a note in the documentation in case it ever comes up again, in which case then add some (very slight) clipping in double precision.

#9 - 10/20/2007 09:56 AM - David van der Spoel

I have added a patch in routine bond_angle like this:

```
costh=cos_angle(r_ij,r_kj);         / 25         */
  if (*costh == -1.0)
    *costh += GMX_REAL_EPS;
  else if (*costh == 1.0)
    *costh -= GMX_REAL_EPS;
```

it seems to solve the problem. If you'd looked in include/vec.h you'd see that we already had tests in cos_angle for costh > 1 or costh < -1, however it is the problem that angles get exactly 180 or 0 that causes these problems.

Shall I commit this to the 3.3 cvs branch?

By the way David, you are really good at setting up "troublesome" systems: I also had to patch grompp to be able to get your topology grompp-ed, since there is a different of constraint generation in 3.3.1 and 3.3.2. Oh well, all the better.

#10 - 10/20/2007 10:13 AM - John D.

Thanks for the quick diagnosis and fix, David!

You're right -- I had missed the rectification conditions to ensure that cos_theta was ensured of being in the domain [-1,+1].

Cheers,

- John

#11 - 10/22/2007 06:28 PM - David Mobley

David, committing this to CVS sounds like a good idea -- it sounds like your solution should do it.

As far as setting up troublesome systems, I'm always happy to oblige. Can I get paid twice my usual rate for finding two problems with one Bugzilla? :)

David

#12 - 10/24/2007 12:13 PM - David van der Spoel

Fixed in CVS for 3.3 and 4.0. The fix is applied to all force calculations (angles dihedrals etc.)

Files

propyne.tgz	3.32 KB	10/17/2007	John D.