Force Distribution Analysis with Gromacs

Wolfram Stacklies

October 21, 2009

Contents

1 Patching and installation

Currently we only provide a `.tar` file with the modified code; a patch will follow soon. Installation should work as normal, see the Gromacs website[1] for details.

2 Force distribution Analysis

Force distribution analysis (FDA) is a method to detect stress propagation in proteins, reminiscent of finite element analysis used to engineer macroscopic structures. The method is based on molecular dynamics simulations during which we directly calculate forces between each atom pair in a system. The most recent version of FDA is now implemented in Gromacs-4.0.5 [1].

What the FDA code basically does is to write out forces F_{ij} between each atom pair i and j in a molecule. Forces include contributions of individual bonded (bond, angle, dihedral) and non-bonded (electrostatic and van der Waals) terms below the cutoff distance, which are stored separately for further analysis. The force between each atom pair is represented as the norm of the force vector and thus is a scalar. Attractive and repulsive forces are assigned opposite signs (negative for attractive, positive for repulsive). It is important to make yourself clear about the concept of pair wise forces. A pair wise force is the force an atom exerts on another atom. It is NOT the total force acting on a certain atom. By considering the direct force between each atom pair, the equilibrium force between these atoms can be different from zero, even for the theoretical case of a system without any motion. We hereby obtain the advantage to be able to observe signal propagation even through stiff materials, where, by definition, forces propagate without causing major atomic displacement. Atomic forces, i.e. the sum over the force vectors acting on an atom, instead average out to zero over time and are not of interest here. A real world example for such force propagation is Newton's cradle, where atomic forces would only see the first and the last beat moving. But when using pair wise forces one is able to see the whole force propagation pathway.

The careful reader may have noticed that we do not write out forces beyond the cutoff distance. This basically only affects Coulomb interactions, as the VdW potential quickly decays to zero. Due to the nature of the non-bonded potentials, pairwise forces are most significant for atom pairs in relative close proximity. At distances $> 1\,\mathrm{nm}$ the Coulomb potential becomes very flat, i.e. slight changes in atomic distances have little effect on

[1] www.gromacs.org

the force between them. For this reason we would anyway end up with a force-propagation network comprised of a series of short to medium ranged connections, and thus it is no problem to ignore forces beyond the cutoff distance. The long range interactions are important for a proteins overall stability, but they play only a minor role in propagating forces.

2.1 When (and why) should I use FDA

FDA is a valuable tool to investigate allostery, or more general, the propagation of mechanical signals within molecules. It is particularly useful to follow signal propagation in stiff allosteric proteins or crystalline materials [2, **?**]. In these structures, by definition, allosteric signals propagate without causing major atomic rearrangements, making it impossible to follow the signal with conventional methods. In other cases, such as e.g. molecular motors, mechanical tension caused by an allosteric event may build up and eventually result in a conformational change, e.g. a rotation. These changes are slow and generally not accessible in simulation time-scales. Even if we cannot see the actual transition, FDA still is able to tell you where tension builds up. Thereby it can identify crucial residues of a protein and help you to understand the underlying machinery.

Another case we used FDA for is the debugging of force field parameters, or to find errors in the initial system setup (oh my god, my system exploded again!!). Getting the forces acting between each atom pair allows to quickly identify and hopefully eliminate the origin of unrealistically high forces.

Pair wise forces are very sensitive to even minor conformational rearrangements. This normally is an advantage, but may become a problem if an allosteric event involves large conformational changes. In this case it might be appropriate to use other methods. For the same reason I strongly discourage you to compare forces obtained from different crystal structures. Even if both structures have very low RMSD, the changes between the structures are most likely stronger than the changes you observe upon a certain perturbation of your system.

2.2 Usage of the FDA code

It is theoretically possible to write out pair wise forces directly during your MD simulation, and in rare cases this might make sense. The caveat of that is that Gromacs uses assembly loops optimized for performance, and turning on the FDA code will force Gromacs to fallback to the slow C inner loops.

Another caveat is that FDA currently does not support domain decomposition, what will slow your simulation down further. Hence, the best solution is to perform a standard MD simulation and afterwards calculate pair wise forces during a rerun MD. **For now I strongly suggest to only use one CPU when doing FDA, I will not guarantee for any results done in parallel!**

Usage of the FDA code is straightforward. You will notice that `mdrun` now supports four new parameters, which are:

- -fi (`pforce.fi`) - The input parameters for FDA (explained in detail later on).
- -fo (`forceout.fo`) The parsed FDA parameters (output).
- -fm (`forcemat.fm`) The matrix containing pair wise forces, a binary file.
- -fn (`pforce.ndx`) An index file defining the atoms for which pair wise forces are written out.

After the simulation you will end up with a three dimensional matrix of size $N \times N \times S$ where N is the number of atoms and S the number of write steps. The matrix is stored in a binary sparse matrix representation, see Section 3 for more information.

But let's have a look at the input parameters given in the `.fi` file. Currently there are only four options, namely the output frequency (in simulation steps), the index group for which forces are written out, the option to write variances in forces instead of the forces themselves and, last but not least a flag that indicates if the (slower) pair-wise force inner loops should be used. This should only be enabled if you really intend to to FDA for waters, what makes sense in only few cases, see also Section 2.6. A typical input file would then look like:

```
output_freq     = 10
group           = Protein
variance        = no
water           = no
```

In this example we would average forces for Protein atoms over every 10 simulation steps and then write them out. Setting `output_freq` to -1 will average forces over the whole simulation time. If we would set variance to `yes` we would end up with a matrix containing only one frame storing the variances in forces over time.

The `.fm` and `.ndx` files contain the force matrix and the index; the index file needs to contain at least the group given in the `.fi` file. The exact format of the force matrix file is explained in Section 3 but is not of interest for the normal user.

2.3 Analysis of the FDA results

There are two steps in the analysis, the "digestion" of the raw forces and the interpretation of the data. First let me give you some guidelines for the first, the "digestion" step.

Looking at the force distribution pattern of a single configuration of your system may reveal some interesting features, but it is unlikely to reveal a force propagation pathway. When talking about "configurations" I hereby mean something like a protein in apo and holo form, or with and without applied external force. Propagation of forces becomes visible when comparing two states, let's say the apo and holo configuration of an enzyme. Regions that do not show change in inter atomic forces obviously do not respond to the external perturbation, whereas other regions may respond to a varying extend. Generally FDA requires exhaustive sampling of the conformational space, and I normally do a bunch of trajectories for a single configuration. E.g. I will do 10 trajectories for the holo and 10 trajectories for the apo configuration. Afterwards I will average forces over all these trajectories. To ease this job, the FDA code comes with a tool called `g_fdatools` that allows to read, edit and compare your `.fm` files. I will shortly summarize what the tool can do, and refer to the documentation coming with it for a more detailed description. `g_fdatools` allows to process force matrices, but it does not provide methods for the analysis. It allows to average forces over time, sum up `.fm` files, calculate variances or standard errors and to average forces over residue pairs. It is also possible to only include certain interaction types in the output matrix.

The option to sum up a set of `.fm` files is very useful in cases when you have a vast number of force matrix files. Such cases can occur if you need to exhaustively sample a certain configuration, in order to obtain converged averages for your forces.

Calculation of standard errors between average forces for individual trajectories gives you an estimate of how well you sampled the conformational space. Getting a high standard error indicates that sampling at this specific position is not particularly good, and that changes observed for this position should be seen with caution. Please note that this option works on **average** forces, i.e. it will only read the first frame of each of the given force matrices.

You might prefer to look at forces between pairs of residues instead of

forces between pairs of atoms. Hereto it is possible to sum up forces between the atoms for each pair of residues. By giving an index file it is also possible to restrict the summation to certain atom types, e.g. backbone or side chain only. Summation can optionally be done on vectorized forces, in this case the corresponding trajectory must be supplied to calculate force vectors.

Finally, `g_fdatools` allows to concatenate several `.fm` files in case your simulation crashed.

Practical examples

Here is a short walkthrough the capabilities of `g_fdatools`. The required files come with the tutorial found on our website.
First, let's sum up a bunch of trajectories:

```
g_fdatools -fm waa.1.fm waa.2.fm waa.3.fm waa.4.fm \
    -sum waa.all.fm
```

The resulting matrix, `waa.all.fm` is of equal dimension, namely $1379 \times 1379 \times 11$. Note that the values have already been averaged, in this case divided by 4.

Next, we calculate averages / variances over the whole simulation time

```
g_fdatools -fm waa.1.fm -avg waa.1.avg.fm
g_fdatools -fm waa.1.fm -var waa.1.var.fm
```

Or do both in one step:

```
g_fdatools -fm waa.2.fm -avg waa.2.avg.fm -var waa.2.var.fm
```

Calculate averages for waa.1.fm to waa.4.fm yourself, you will need them in the next steps. The averages are just time averages over the whole trajectory, the variances are a measure for the fluctuations in force (the flexibility) of a certain interaction during time.

It is possible to calculate averages over certain intervals (here, we join every 2 frames)

```
g_fdatools -fm waa.1.fm -avg waa.1.avg.fm -stride 2
```

The output, `waa.1.avg.fm`, will now be of dimension $1379 \times 1379 \times 6$

We will now use the time averages calculated before to get standard errors between individual trajectories.

```
g_fdatools -fm waa.1.avg.fm waa.2.avg.fm waa.3.avg.fm \
    waa.4.avg.fm -stderr waa.stderr.fm
```

We will later on use this information to normalize our data.

Last but not least let's calculate residue averages (juppiduppiduuu).
First by simply summing up forces between residue pairs, ignoring the ori-

entation of the atomic force vectors. The output trajectory has the same number of frames as the input force trajectory.

```
g_fdatools -fm waa_pca.fm -s waa.tpr -n waa.ndx -res res_simple.fm
```

Second, using exact force vectors which are derived from a trajectory:

```
g_fdatools -fm waa_pca.fm -s waa.tpr -n waa.ndx -f waa_pca.xtc
    -res res_vector.fm
```

In this case, if the input trajectory has N frames, the output trajectory will have $5 \times N$ frames. This is because the output trajectory will contain the norm of the force vectors as well as the x, y, z components and distances between the center of masses of the residues. Having individual contributions is very useful when doing PCA.

It is also possible to include only certain interactions (e.g. bonded or non-bonded). The next example will only use Coulomb and Vdw interactions:

```
g_fdatools -fm waa_pca.fm -s waa.tpr -n waa.ndx -f waa_pca.xtc
    -res res_nb.fm -interaction 5+6
```

2.4 Analysis with the FDAtools package in R

For further analysis, we currently provide an R[4] package that allows for direct import and analysis of pair-wise force data into in R. In case you don't want to use R, there is another tool called `g_fdaconv` that allows to convert binary `.fm` files into an ASCII format. The `FDAtools` package comes with documentation and code examples for every function, this guide will thus give a more general overview over the functionality.

`FDAtools` provides a set of methods for importing and analyzing `.fm` files. The most simple one is `readFM`, just reading all data into a three dimensional matrix, of dimension $N \times N \times F$, where N is the number of atoms and F the number of frames. This function is memory consuming, but the output format is easy to handle. I normally use this method when working with average forces, i.e. with matrices having only one single frame, or when using residue averaged forces what typically results in a small N. If your are just interested in part of your force-matrix, e.g. a set of hydrogen bonds, use `readFMpart`. This function lets you specify a set of atoms forces will be read for. Another function, named `readFMpca` allows for memory efficient reading of residue averaged force trajectories and will be explained in more detail later on.

Various functions to visualize force distribution by mapping pair-wise forces onto pdb structures are provided. I found two ways of visualization particularly helpful. The first is to simply color the secondary structure after the overall change single atoms / residues see upon a perturbation.

The second way is to draw a network like representation of edges connecting atom pairs showing high change in pair-wise forces. Such networks are helpful to identify e.g. a force bearing scaffold, or to visualize correlated changes in forces. Simple networks connecting atoms / residues showing changes in pair-wise force, ΔF, can be created using the `atomGraph` and `resGraph` methods. Single connections in such a network, e.g. hydrogen bonds between two β-sheets can be drawn with `pdbEdges`. All of these functions assign individual b-factors to each edge, according to the change in force.

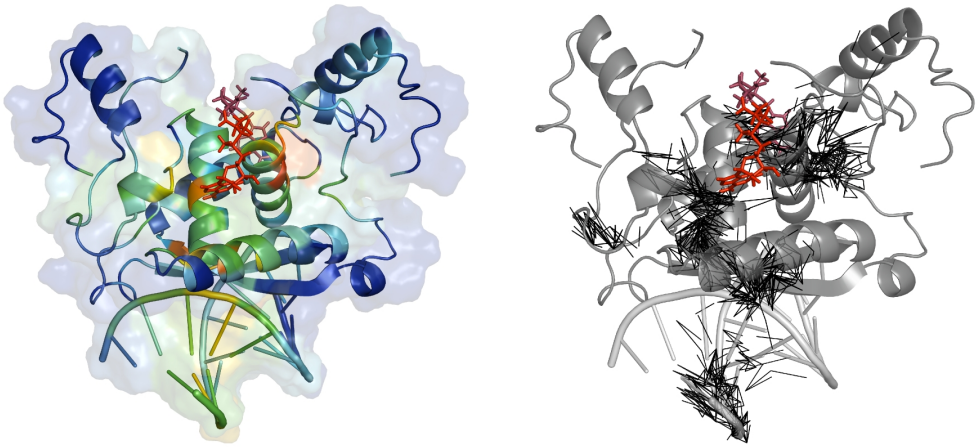

Figure 1: Changes in pair-wise forces color coded onto a protein structure (left) and in network representation (right). The network was created using `atomGraph`. All figures were created using Pymol.

A bit more effort is required to created networks showing correlated changes in pair-wise forces, e.g. caused by harmonic fluctuations or by an evil user pulling a helpless protein in regular intervals. As calculating a covariance matrix on atom wise forces is generally not feasible simply due to the vast number of atoms, this functionality is limited to residue averaged forces. To find correlations in residue wise forces, we first need to process our data in an adequate way. Hereto, the first step is to calculate a force matrix containing as much frames as possible, typically the number of frames in your Gromacs `.xtc / .trr` trajectory. Then one calculates residue averaged forces, using `g_fdatools` also supplying your coordinate trajectory. The command should look something like:

```
g_fdatools -s mol.tpr -f trj.xtc -fm force.fm -res resavg.fm
```

By specifying the `-interaction` flag it is also possible to restrict averaging to certain potentials, e.g. Coulomb or Vdw only, see Section 2.3 for an example. Now we should have ended up with a (much smaller) residue average force matrix containing the norm of the force, as well as x, y, z components

and distances of the residue center of masses. Next I recommend to delete the original (atom wise) force matrix, it most likely occupies some senseless GB on the hard drive.
We are done with pre-processing now. In order to identify correlated changes, we will use principal component analysis (PCA), as implemented by the `prcomp` function in R. The first step is to read in pair-wise forces using the `readFMpca` function. This function expects forces in x, y, z component as can be calculated form a force and a coordinate trajectory by `g_fdatools` (using the `-res` option). It is advisable to perform PCA for x, y, z separately and to calculate cumulative eigenvectors afterwards. It is possible to tell `readFMpca` what one wants to read, options are one of the x, y, z components or the norm of the force. Distances between COMs of the residues can also be selected, e.g. for force-distance plots. The output matrix produced by `readFMpca` can directly be used as input for `prcomp`. As a convenience function, `fpca` automatically performs PCA on the x, y, z component of a given force trajectory, by internally calling `readFMpca` several times.

Having run PCA, now is a good time to check the eigenvalue structure. If there is one (or only few) dominant eigenvalues most of the observed covariance is captured within the first few eigenvectors. Only in this case projecting / visualizing these eigenvectors makes really sense. In case you decide visualizing is a good thing to do, a network of residue pairs showing correlated changes in forces can be drawn using the `pcaGraph` function. This function will draw an edge between each residue pair for which the given eigenvector is greater than the cutoff. The cutoff should be chosen to mostly include significant values (I agree that it is somewhat tricky to define "significant", and it will highly depend on your system).

2.5 Noise and normalization

Depending on the level of noise in your data normalization will be helpful. The noise level mainly depends on the flexibility of your system and on the simulation time you have available for conformational sampling. I found normalization by the standard error of the mean to improve data quality in most cases. In [2] we compared the change in force between a system in two states A (F_{ij}^{A}) and B (F_{ij}^{B}) for each atom pair i, j. We then normalized by the standard error of the mean ϵ to account for differences caused by insufficient sampling, i.e. slow side chain fluctuations that cannot equilibrate in simulation time. The normalized change in force, $\Delta f_{i,j}$ was then defined as:

$$\Delta f_{ij} = \frac{F_{ij}^{\mathrm{A}} - F_{ij}^{\mathrm{B}}}{\epsilon_{ij}^{\mathrm{A}} + \epsilon_{ij}^{\mathrm{B}}} \tag{1}$$

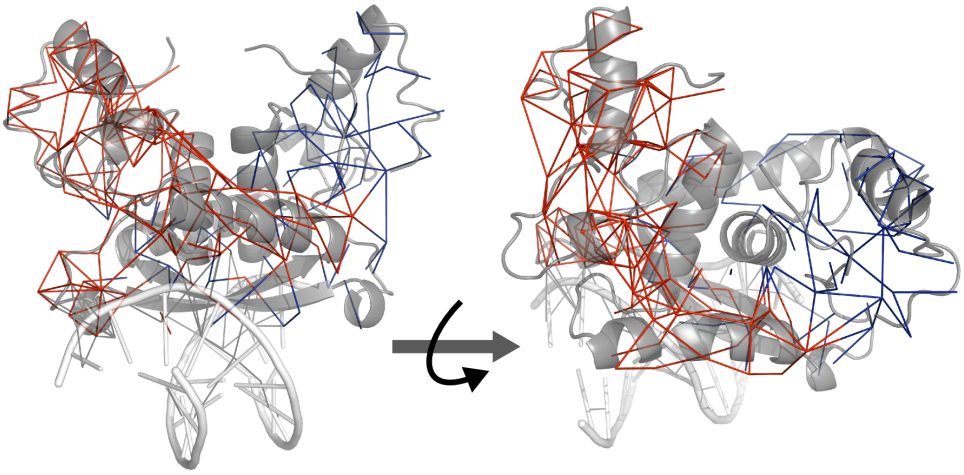

Figure 2: A network showing correlated changes in pair-wise forces as revealed by PCA. The network was created using `pcaGraph`, PCA was done for x, y, z components of residue averaged forces.

Hereby, the overall distribution pattern remained largely unchanged, but unrealistically high changes in force for some solvent exposed residues were removed.

2.6 Limitations

Due to the fast movement of water molecules FDA can only applied to immobilized water molecules, e.g. such buried inside proteins or interfaces. As calculation of pair-wise forces for waters is time-consuming you will have to enable FDA for water by setting the
`water = yes`
flag in your `.fm` file.

As discussed in Section 2 long range electrostatic interactions (PME) are not taken into account. A workaround is to do a rerun MD with an extremely large cutoff. But be aware that the size of your force matrix file will increase quadratically with the number of atoms. Don't tell me you have not been warned...

3 Implementation details

At runtime, forces are stored in a matrix of dimension $N \times X$, where N is the number of atoms and X is the maximal number of interaction partners for

a single atom. The number of interaction partners may change during the simulation, and if required additional memory will be dynamically allocated. Typically X will be in the range of 300-600. You can estimate the amount of memory needed (in MB) by the formula $N \cdot X \cdot 18/1048576$. Note that the matrix is not symmetric! For non-bonded interactions Vdw and Coulomb forces are stored separately. The right triangle (`row` > `column`) is used for Vdw interactions, the left triangle (`column` > `row`) is used for Coulomb interactions, Figure 3. To save space, bonded interactions are stored only once for each pair as well.

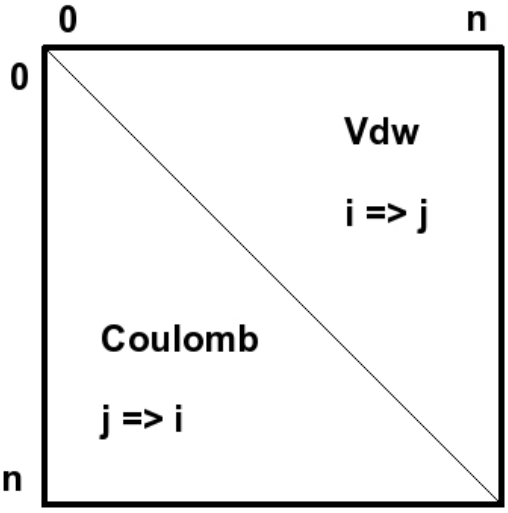

Figure 3: Space for Vdw and Coulomb interactions in the force-matrix.

`g_fdatools` internally uses a slightly different sparse matrix implementation. The version described above is relatively fast in access, but not optimal regarding memory consumption. As under some conditions `g_fdatools` needs to keep a bunch of matrix frames in memory, I decided to implement the sparse matrix as a list. This is easier to handle and consumes less memory, but is also considerably slower.

3.1 Approximations for multi body forces

Angle and dihedral terms are multibody forces defined between atoms i, j, k (angles) and i, j, k, l (dihedrals). Internally Gromacs calculates the force vectors $\vec{i}, \vec{j}, \vec{k}$ and $\vec{l}$ acting on the atoms. As we cannot directly map such multibody forces to pairwise interactions, we represent angle bending as $\vec{k} - \vec{i}$ and dihedral bending as $\vec{l} - \vec{i}$, giving as a result approximately the force acting between all atoms in direction i, k / i, l. This is illustrated in Figure 4.

Both approximations do not provide physically correct forces, but they are sufficient to detect rearrangements under strain. Proper and improper dihe-

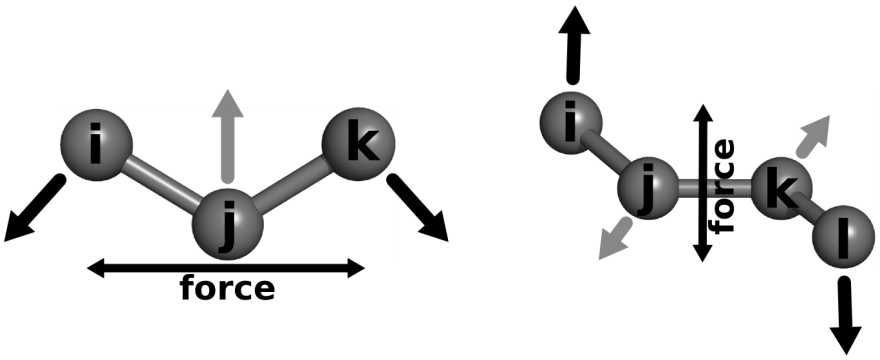

Figure 4: Approximations used for angle and dihedral terms.

dral angles are ignored as it is not really possible to map these forces into a pair wise representation.

3.2 The force matrix file format (version 1.2)

The `.fm` file contains a 13-line ASCII header followed by a number of binary data frames, each frame represents one write step. The following entries in the header are allowed, entries need to be of the form `label=value`:

- version (char*) - `.fm` file version
- groupname (char*) - index group for which forces were written out
- writefreq (int) - output frequence (in simulation steps)
- nsteps (int) - number of frames in the file. This is an estimate, don't trust this value!
- sysanr (int) - number of atoms in the whole system, including water and ions
- fmdim (int) - dimension of the force matrix, what equals to the number of atoms for which forces were written out
- intsize (int) - the `sizeof(int)` for the hardware the file was written on
- realsize (int) - the `sizeof(real)` for the hardware the file was written on

Comments in the header follow the Gromacs standard and must contain a ";" as first character.

Each of the following binary blocks contains an array of forces and the according index and interaction arrays. The index array stores the index in the force matrix (according to a symmetric atoms × atoms matrix) and the interaction array the interaction type (bond, angle, dihedral, VdW or Coulomb). Table 1 shows the structure of a binary block.

datatype	count	description
int	1	Current write step
int	1	Number of entries in the force-matrix (entries)
int*	fmdim	Atom ID's of the atoms in the force matrix
int*	entries	The index array
real*	entries	The forces
char*	entries	The interaction type between each two atoms. This can be bond | angle | dihedral | Vdw | Coulomb.
int	1	Each block is closed by writing an integer with value NEW_ENTRY

Table 1: Structure of a binary block in the force matrix file.

3.3 ASCII representation of the force-matrix

The ASCII representation as created by `g_fdaconv` is in principle identical to the binary structure, and Table 1 should allow to intuitively understand the format. The header is unchanged; each ASCII block in the table starts with a `<begin_block>` and ends with a `<end_block>` tag. Thus no `NEW_ENTRY` flag is needed.

3.4 Where do I find the FDA code?

Most of the FDA code is found in three files in the `gmxlib` directory. Functions for handling pairwise forces at runtime are defined in `./include/fmutils.h`. The I/O functionality is combined in `./include/fmio.h` and functions related to sparse matrix handling are defined in `./include/fmsparse.h` and `./include/types/forcemat.h`.

Writing out pair wise forces requires modifications of the Gromacs inner loops. To get best performance Gromacs inner loops are implemented as assembly files and were optimized for various systems. Alternatively one

can use relatively slow inner loops implemented in C. Only the inner loops written in C have been modified to output pair wise forces. To still allow for efficient computation, I decided to provide a separate set of non-bonded FDA kernels. These kernels are slight extensions of the standard kernels and are found in `./src/gmxlib/nonbonded/nb_kernel_pf/`. The routine calling the non-bonded kernels, `./src/gmxlib/nonbonded/nonbonded.c` was modified such that FDA the (slow) FDA kernels are only used for protein-protein interactions. For protein-water and water-water interactions the default (normally assembly) kernels are used. For bonded interactions the functions in `./src/gmxlib/bondfree.c` were modified and now contain an additional call to a function that stores the pair wise forces. For a complete list of modified files see the Appendix.

References

[1] Hess B, Kutzner C, Van der Spoel D, Lindahl E (2008) GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation* 4(3):435-447

[2] Stacklies W, Vega, M C, Wilmanns M, Gräter F (2009) Mechanical network in titin immunoglobulin from force distribution analysis. *Plos Computational Biology* 5(3):e1000306

[3] Xiao S, Stacklies W, Cetinkaya M, Markert B, Gräter F (2009) Mechanical response of silk crystalline units from force distribution analysis. *Biophysical Journal*, 96(10):3997-4005

[4] R Development Core Team (2008) R: A Language and Environment for Statistical Computing *R Foundation for Statistical Computing*

A Files modified for the FDA implementation

This is a list of all files that were modified (or added) in order to implement FDA into Gromacs-4.0.5.

```
=> ./
./configure.ac

=> ./include
./include/Makefile.am
./include/bondf.h
./include/typedefs.h
add: ./include/fmio.h
add: ./include/fmsparse.h
add: ./include/fmutils.h

=> ./include/types
./include/types/Makefile.am
./include/types/filenm.h
./include/types/ifunc.h
./include/types/forcerec.h
add: ./include/types/forcemat.h

=> ./src/gmxlib
./src/gmxlib/Makefile.am
./src/gmxlib/bondfree.c
./src/gmxlib/dihres.c
./src/gmxlib/disre.c
./src/gmxlib/filenm.c
./src/gmxlib/orires.c
add: ./src/gmxlib/fmio.c
add: ./src/gmxlib/fmsparse.c
add: ./src/gmxlib/fmutils.c

=> ./src/gmxlib/nonbonded/
./src/gmxlib/nonbonded/Makefile.am
./src/gmxlib/nonbonded/nb_kerneltype.h
./src/gmxlib/nonbonded/nonbonded.c
add directory: ./src/gmxlib/nonbonded/nb_kernel_pf
```

```
=> ./src/mdlib/
No files modified here :)

=> ./src/kernel/
./src/kernel/md.c
./src/kernel/mdrun.c

=> ./src/tools/
./src/tools/Makefile.am
./src/tools/gmx_disre.c
add: ./src/tools/g_fdatools.c
add: ./src/tools/g_forcemat.c
add: ./src/tools/g_forcemat.h
add: ./src/tools/g_fdaconv.c
```