NEMATIC ORDERING, ORDERING DEPENDENT ANISOTROPIC ELASTICITY, AND MORPHOLOGY OF FLUID MEMBRANES

`Developer:` **Ramakrishnan Natesan** (www.seas.upenn.edu/~ramn)

`In Collaboration with:`

1. *Prof. John H. Ipsen* (MEMPHYS)
2. *Prof. P. B. Sunil Kumar* (IIT Madras)
3. *Dr. Sreeja Kutti Kandy* (IITM & University of Pennsylvania)
4. *Dr. Allan G. Hansen* (MEMPHYS)
5. *Prof. Ravi Radhakrishnan* (University of Pennsylvania)

`Bug reports and queries may be directed to:` *nrkssa@gmail.com*

BIBLIOGRAPHY

This code may be used to reproduce most of the results reported in the following articles

1. N. Ramakrishnan, P. B. Sunil. Kumar, John. H. Ipsen, *Phys. Rev. E 81, 41922 (2010)*
2. N. Ramakrishnan, P. B. Sunil. Kumar, John. H. Ipsen, *Macromolecular Theory and Simulations 20, 446–450 (2011)*
3. N. Ramakrishnan, John. H. Ipsen, P. B. Sunil. Kumar, P. B. Sunil. Kumar, *Soft Matter 8, 3058–3061 (2012)*
4. N. Ramakrishnan, P. B. Sunil. Kumar, John. H. Ipsen, *Biophysical Journal 104, 1–11 (2013)*
5. N. Ramakrishnan, P. B. Sunil. Kumar, Ravi Radhakrishnan, *Physics Reports 543(1), 1–60, (2014)*
6. N. Ramakrishnan, Ravi Radhakrishnan, *Advances in Planar Lipid Bilayers and Liposomes, (2015)*

OVERVIEW

Here, I give a brief overview of the physics underlying this code. It aims to evolve the morphology of an elastic surface, of a specified topology, in its corresponding configurational space. **In this version, the code only handles surfaces without boundaries.**

SYSTEM

There are three two components in the model:

1. **A membrane surface** whose configurations is governed by the Canham-Helfrich Hamiltonian

$$H_{el} = \frac{\kappa}{2} \int (2H - C_0)^2 \, dS$$

   In the implementation here, we have set $C_0 = 0$.
2. **A unit in-plane nematic field** $\hat{m}$, whose lateral organization and texture are governed by Lebwhol-Lasher interactions.

$$H_{NN} = \frac{\epsilon_{LL}}{2} \int (\nabla\theta)^2 \, dS$$

This is an one constant approximation of the Frank's Free energy. *It should be remembered that the angle between the neighboring nematic field vectors in the calculations in the codes is computed using a parallel transport technique.*

3. **An anisotropic elastic term that couples the orientation of the nematic field to the curvature of the membrane**

$$H_{NC} = \frac{\kappa_{\parallel}}{2} \int \left(\hat{m} K \hat{m} - C_0^{\parallel}\right)^2 dS + \frac{\kappa_{\perp}}{2} \int \left(\hat{m}^{\perp} K \hat{m}^{\perp} - C_0^{\perp}\right)^2 dS$$

   Terms 1 and 2 corresponding to anisotropic stiffness and curvature along directions parallel and perpendicular to $\hat{m}$.

Total energy of the system is given by:

$$H_{tot} = H_{el} + H_{NN} + H_{NC}$$

MODEL

The two dimensional surface is discretized into a triangulated surface with N vertices, T triangles and L links. These are related to the surface topology through the relation $\chi = N + T - L$, where χ is the Euler characteristic.

The morphology of the surface is evolved through a set of four Monte Carlo moves, which are aimed to change each degree of freedom independent of the other. The moves are as follows:

- **Vertex move:** A randomly chosen vertex is displaced by an arbitrary displacement. This move is designed to simulate thermal fluctuations in a membrane surface.
- **Link flip:** A randomly chosen link, which shares two triangles, is reconnected to the two previously unconnected vertices. This move is to simulate the diffusion of lipids in the membrane.
- **Rotation of the in-plane nematic field:** The nematic vector $\hat{m}$ at a randomly chosen vertex is rotated in the tangent plane of the vertex to a new orientation $\hat{m}^{'}$. This move aims to simulate the thermal noise in the orientation of the in-plane field (*breathing modes of a protein*).
- **Kawasaki exchange of the in-plane nematic field:** This move is relevant for partly decorated nematic membranes. Here, the nematic field $\hat{m}$ at a randomly chosen vertex is exchanged with that at one of the randomly chosen neighboring vertices.

All the above are accepted using the Metropolis criterion:

$$P_{acc} = \min(1, \exp(-\Delta H_{tot}))$$

where ΔH_{tot} is the change in the total energy after a move. *You need to follow a similar approach if you want to augment this code with additional degrees of freedom*.

TECHNICAL DESCRIPTION

- The entire code has been written in `FORTRAN`, in an `objective oriented fashion`, and adheres to specifications laid out for `2003 or later standards`.
- `MPI` based parallelization has also been implemented and the current implementation only handles `SINGLE INSTRUCTION MULTIPLE DATA (SIMD)` mode. This means, for a given set of parameters, you can specify the number of processors `N` and the code will automatically generate data for `N` independent ensembles.
- The code uses `FORTRAN TYPE` structures which are the equivalent of a `typedef` in `C or C++`. The type structures used are as follows:

```
DATASTRUCTURE FOR THE VERTICES ON THE SURFACE

Type vertex
  REAL(KIND=8),DIMENSION(3,1)::splo,spgl,vnor,vcoord,t1,t2
  INTEGER,DIMENSION(10):: vneipt,vneitr
  REAL(KIND=8),DIMENSION(3,3)::L2G
  INTEGER :: nonei,phase,clno,neigh
  REAL(KIND=8)::mcur,cur1,cur2,totarea,spen,op,WN
  REAL(KIND=8):: kap,kpar,kper,cpar,cper,Nemal,Isal
  REAL(KIND=8)::rkap,rkpar,rkper,rcpar,rcper,rNemal,rIsal
  REAL(KIND=8):: ukpar,ukper
  INTEGER :: linkcell,next_vert,prev_vert
End Type vertex
```

```
DATASTRUCTURE FOR THE TRIANGLES ON THE SURFACE

Type triangle
  REAL(KIND=8)::ar,vol
  INTEGER,DIMENSION(3) ::li,vert
  REAL(KIND=8),DIMENSION(3,1)::fnor
End Type triangle
```

```
DATASTRUCTURE FOR THE LINKS ON THE SURFACE

Type link
  INTEGER ::tr
  INTEGER,DIMENSION(2) ::sep
End Type link
```

```
DATASTRUCTURE FOR VARIOUS PROPERTIES OF THE MEMBRANE SURFACE

Type membrane_prop
  .
  .
END Type membrane_prop
```

```
DATASTRUCTURE FOR THE LINKLIST

Type Linklist
  Real(Kind=8) :: start_coord(3,1),end_coord(3,1)
  Integer :: first_vert,last_vert,num_vert,buffer
  Integer :: neigh_cell(27)
End Type Linklist
```

ORGANIZATION

The code is divided into the following modules:

`module_datastruct.f` -- definition of the datastructure
`module_rerun.f` -- modules to start from and dump restart files
`module_dataformat.f` -- vtu and jvx file writers for visualization
`module_initialize_system.f` -- modules to initialize the datastructure
`module_linklist_calc.f` -- linklist definition and calculations
`module_curvcalc.f` -- compute the curvature at each vertex
`module_energy_calculations.f` -- energy calculations
`module_mcsmoves.f` -- definition of the various Monte Carlo moves
`module_compute_analytic_measures.f` -- compute various measures
`maincode.f` -- definition of the `MAIN` function

SYSTEM PARAMETERS

- All the required inputs are defined in a file named `parameters.in`

- **RESTART FILES**
  If `restart_mode` is defined as `MEMBRANE` then the code will look for pre-equilibrated configuration file named `startdet.in`, which is dumped from `module_rerun.f`.

COMPILATION & EXECUTION

- `REQUIRED COMPILER: mpif90`

- There are three makefiles supplied along with the code:
  `Makefile` -- generic make file that makes call to
  `Makefile.gnu` -- for `gnu` based compiler
  `Makefile.intel` -- for `intel` based compiler

- Compile as `make gnu` or `make intel`, depending on your system, which would produce an executable `nematic-membrane`.

- The executable may be executed as:
  `mpirun -np xx ./nematic-membrane`, where `xx` is the required number of

processors.

So far, I have tested this code using `mpif90` provided with `MPICH, OPENMPI, and MVAPICH2`, compiled with both `GNU and INTEL` compilers. *In general, any stock compiler should work without a problem*.

VISUALIZATION

Upon execution, the code dumps a number of files which includes `conf-*.vtu` and `conf-*.jvx` which are VTK/XML based formats for the morphology of the nematic membrane. These files may be visualized either using:

`paraview` (www.paraview.org) for the `.vtu` files
`javaview` (www.javaview.de) for the `.jvx` files

In the case of paraview, in order to view the nematic field, you should load the `glyph` object with `line` representation (from -0.5 to 0.5) and choose `scale as vector`, with a scaling factor of 1.0.
