bob v1.0 : A code to compute linear rheology of Branch-On-Branch polymers

December 10, 2005

Contents

Chapter 1

Overview

BOB is a c++ code to calculate linear rheology of polymer melts of arbitrary architecture. This chapter gives a brief overview about how to install the code, how to get help and how to use the code.

1.1 Installation

Being written in standard c++, you should be able to import in any operating system with a native c++ compiler.

For any flavor of Unix, download the current version bob_1.0.tar.gz and use gunzip and untar to retrieve the directory structure. Type ./configure from the directory bob_1.0. If it does not find c++ compiler in the current path, it will prompt you to input the c++ compiler. Next it creates the binaries in bob_1.0/bin. It also creates paths required to access the binary files and man files and prompts you if you want to add them in your .cshrc.

For windows platform, download the executable (it has been compiled via MinGW/gcc route). You can also download the source files and compile under MinGW/MSYS (http://www.mingw.org) or Cygwin/X (http://x.cygwin.com) in the same fashion as for Unix machines. If you are willing to put in some effort in resolving the paths of the different components, you can also use any c++ compiler that you happen to have.

1.2 Usage

Assuming that bob_1.0/bin is in your $path, you are ready to use bob. **bob** starts a text based interface which asks you some questions to find out what it is supposed to do. **bob** $--$**help** gives you a brief help on usage and directs you to more detailed man pages. **bob** $--$**version** shows the version information.

If you are using the code repeatedly, the easier option will be running in batch mode. **bob -b** starts a batch mode where the input is read from

a file instead of the terminal. Default input file is *inp.dat*. You can supply a different input file, ex. *myinp.dat* by typing **bob -b -i myinp.dat**. **bob** can create a number of simple and some not-so-simple polymer architectures. You can also supply a configuration file with information about the polymers you want to consider by typing **bob -b -c myconf.dat** - the default is *polyconf.dat*. Details about the formats for input file and configuration file appears later in this document.

1.3 Issues

- **bob** can not handle cyclic molecules or gels. If any of the polymers remain unrelaxed after $10^{20}\tau_e$ (which amounts to centuries for most polymers), the program is instructed to give up with an warning message.
- The prefactor for compound arm retraction probably need some consideration.
- A graphical user interface is missing.

Some or all of this will be resolved in future versions.

1.4 Credits

This program was developed with a funding from EPSRC.
Internally it uses a Mersenne Twister random number generator:
`http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html`

1.5 Bug reports

If you discover any bugs please report to `chinmaydas@yahoo.com`. Also you can report any additions you make to the code and want it to be considered in future versions to the above mentioned address.

1.6 Reference

"Computational linear rheology of general branch-on-branch polymers", Chinmay Das, Nathanael Inkson, Daniel J. Read, Mark A. Kelmanson and T. C. B. McLeish; to be submitted to Journal of Rheology (2005).

Chapter 2

Using bob

This chapter details the usage of bob including the formats for input and polymer configuration file.

2.1 Input

bob can be used interactively or in a batch mode (with the flag -b). For using in batch mode the input should be in a file *inp.dat* in the same directory from which you start the run. You can specify a different input filename (not longer than 70 characters) with the flag -i.

The structure of the input file is given here. For interactive mode, you require the same information - but you will be led through a number of questions and prompted to type in the information. Unless you are familiar with the input file structure, interactive mode is less prone to error in giving input parameters.

2.1.1 Memory

The first line of the input file has maximum number of polymers and maximum number of segments you want to consider. Linears are considered as made up of two segments. So, the maximum number of segments should be atleast twice the maximum number of polymers. These two numbers are used to allocate memory for representing the polymers. If you specify less than what you use, the program will behave unpredictably.

2.1.2 Relaxation parameters

The second line has a single double precision number : dynamic dilation exponent α. Different scaling arguments have been used to fix α as 1 or alternatively 4/3.

A number of assumptions are required to model the relaxation process. The third entry is an integer : if it is 1, a predetermined choice for the assumptions will be made. Any other integer will require detailed inputs.

Here we are assuming that you wanted to provide detailed input. In that case, the next line has two double precision numbers and two integers : R_L, p^2, $pref_{mode}$ and $Rept_{scheme}$. Each of these are discussed below in some details.

R_L models when a star arm becomes disentangled. When the (dilated) length left to retract becomes smaller than R_L - the arm is considered to be disentangled and it relaxes completely. Similarly, if the length to reptate becomes smaller than $2R_L$, the linear reptates instantaneously. Traditionally R_L is considered to be zero.

When a side-arm collapses, the branch-point associated with the side-arm diffuses with hop-size pa - where p is a scalar number to be fixed from observation and a is the tube diameter. The input requires value of p^2. For $R_L \approx 0$, a value of $1/40$ was found to model a diverse ensemble of polymer rheology data correctly.

For a compound arm retraction, the prefactor in the retraction time is not yet known. We handle this heuristically : If you set $pref_{mode} = 0$, the code uses the same prefactor as the outer-most arm. For $pref_{mode} = 1$ the same form of prefactor as a simple arm is used, but all instances of arm length is replaced by effective arm length. $pref_{mode} = 2$ uses the full effective friction from collapsed side-arms. Possibly an interpolation from mode 0 to mode 2 is required to model deep retraction. For time being, $pref_{mode} = 1$ is found to give reasonable agreement for some comb molecules and metallocene-catalyzed branched polyethylene resins.

$Rept_{scheme} = 1$ or 2 uses reptation in thin and current tube respectively. Current tube performs badly and thin tube is preferred for most cases. However, for extreme binary blends, thin tube reptation will fail. $Rept_{scheme} = 3$ and 4 tries to handle that. For $Rept_{scheme} = 3$, reptation is considered to happen in a tube diameter from the past where a linear molecule was able to reptate by a fixed amount. While, $Rept_{scheme} = 4$ considers reptation from time when a linear molecule was able to reptate by a fixed fraction of its length. For $Rept_{scheme} = 3$ or 4, the next entry is a double precision number $Rept_{amount}$ which tells what length (or fraction of chain length) the linears should be able to reptate at which time the tube diameter is stored and used in reptation. For $Rept_{scheme} = 3$, a value of unity for $Rept_{amount}$ has been used in literature. Probably more detailed analysis is wanted here. The default choice is reptation in thin tube.

We follow relaxation in time starting from a small number t_0 and taking snapshots at multiplicative time steps with step size multiplied by m at each step. The next entry are t_0 and m. Typical values are $t_0 = 1e-4$ and $m = 1.002$.

2.1.3 Material specific parameters

Next two lines in the input file requires material specific parameters. The first of these needs mass of a monomer M_0 (in atomic units), ex. M_0(PE)= 28.0; Number of monomers in an entanglement length N_e, ex. N_e(PE)$\approx$ 35.0 and mass-density of the polymer ρ in (Kg/m^3), ex. ρ(PE)$\approx$ 785.0.

The next line contains the entanglement time τ_e (in seconds) and the temperature (Kelvin).

2.1.4 Polymer configuration

The next line contains number of components to be considered. If number of component is 0, the program reads saved configuration from a file (unless instructed otherwise by using -c flag, from *polyconf.dat* from the run directory).

If you consider n component blends, there will be subsequent n blocks detailing the components. Each of them starts with a double precision entry : weight fraction of the current component. The next entry are two integers. The first is number of polymers of the current component. The second integer selects the polymer type : 0(linear), 1(star), 2(asymmetric star), 3(H), 4(Comb), 5(Cayley tree), 6(Metallocene-catalyzed PE). Depending on your choice about the polymer, you will need to provide more information.

Linear

The first entry for linear block is type of arm you want. 0 considers strictly monodisperse arms. 1,2,3,4 draws the arms from Gaussian, Lognormal, living polymer ensemble and Flory distribution respectively. Living polymer ensemble is a Poisson distribution with the amount of monomer acting as a single unit being determined to reflect the experimentally determined polydispersity.

In batch mode, for any of the above choice you need to supply weight-averaged mass (M_W) for the molecules and polydispersity index (PDI). PDI is required but neglected for monodisperse case.

Star

Once again, the first entry is arm type. The second entry is M_W and PDI. Here M_W is the weight of a single arm and not the whole molecule. The third entry is the number of arms. The built in polymer generation routine can create only 3, 4 or 6 arm stars.

Asymmetric star

Only three arm asymmetric stars are considered : two of the arms have the same length. First line has the arm type of the two similar arms. Second

line contains M_W and PDI for those two arms. The next two lines contain the same information for the odd arm.

H

arm type, M_W and PDI for the side arms. Next the same informations for the cross bar.

Comb

Backbone armtype, backbone M_W and backbone PDI. Next side-arm armtype, M_W, PDI and number of side arms.

Cayley

Number of generations (0 signifies a star). For each generation, arm type, M_W and PDI

M-PE

M_W and number of branches per molecule β. Note that there is no information on arm type in this case. This is because arms are always sampled from Flory distribution. Also M_W refers to mass of a molecule and not segments.

2.1.5 Sample input files

Cayley tree of generation two

The lines are commented below in c++ style. In actual input file they should not be there.

```
100  5000  //maximum polymer=100, maximum segment = 5000
1.0        //Alpha = 1.0
0          //providing detailed choice
1e-5 0.025 1 1 //R_L, p^2, prefactor mode, thin tube reptation
1e-4 1.002  // t_0 and m
28.0 30.0 800.0   // M_0, N_e and mass density
1e-8 350     // tau_e and temperature
1           // single component
1.0         //occupies weight fraction unity
10  5       // actual number of polymers 10 of type 5(Cayley tree)
2           // generation 2
2 10000 1.01 // gen 0: inner star: lognormal, M_W = 10000, PDI=1.01
2 8000 1.02  // gen 1: lognormal, M_W=8000, PDI=1.02
0 5000 0.0   // gen 2: Monodisperse, M_W=5000, PDI is ignored
```

We can let the program use default values for some of the parameters:

```
100  5000
1.0
1          //Use default choice
28.0 30.0 800.0
1e-8 350
1
1.0
10  5
2
2 10000 1.01
2 8000 1.02
0 5000 0.0
```

We can store the polymer configuration in a file (default file *polyconf.dat*) and read from the file :

```
100  5000
1.0
1
28.0 30.0 800.0
1e-8 350
0         // read stored configuration
```

Cayley tree of generation two blended with 4 arm stars

Suppose we want to find out how Cayley tree polymers considered before behave when diluted with some 4 arm stars. For example let us consider 30% Cayley tree with 70% stars:

```
100  5000  //maximum polymer=100, maximum segment = 5000
1.0        //Alpha = 1.0
0          //providing detailed choice
1e-5 0.025 1 1 //R_L, p^2, prefactor mode, thin tube reptation
1e-4 1.002  // t_0 and m
28.0 30.0 800.0   // M_0, N_e and mass density
1e-8 350     // tau_e and temperature
2           // two components
0.30         //First component occupies 30% weight
10  5       // actual number of polymers 10 of type 5(Cayley tree)
2           // generation 2
2 10000 1.01 // gen 0
2 8000 1.02  // gen 1
0 5000 0.0   // gen 2
```

```
0.70        //second component 70%
15  1       // 15 polymers of type star
2           // stars arms from lognormal distribution
30000 1.01  // M_W and PDI of star arms
4           // number of arms
```

2.2 Output and files

So far we have talked about input file and polymer configuration files. These file names you can choose by using flags while running the code. The program also creates a file called *info.txt* which has the informations from different stages of the execution. It contains the choice of parameters used, the kind of polymers selected, a descriptive analysis about the topology of the polymers considered and few numbers like zero-shear viscosity.

If some of the molecules fail to relax, information about it is dumped in a file called *debg.dat*.

supertube.dat contains ascii data with time t, unrelaxed fraction ϕ, supertube fraction ϕ_{ST} and actual unrelaxed fraction ϕ_{true}.

gtp.dat contains frequency ω, $G^{'}(\omega)$ and $G^{''}(\omega)$. A file called *bobsv* is created which contain just enough information to compute $G^{'}(\omega)$ and $G^{''}(\omega)$ for the polymer ensemble considered with out going through the relaxation process again.

Without tinkering with the source code, you can not change these output filenames. Any file with same names in the execution directory will be overwritten.

2.3 Polymer configuration file

The first line of polymer configuration file should have a string naming the configuration. The string should be of length less than 10 characters.

The second line is a double precision number giving N_e. The segment lengths are stored in units of M_e - while M_0 is fixed, N_e is a fit parameter. Hence N_e appears here which enables one to get the actual segment lengths if one wishes to.

The third line is the number of polymers in the file.

Each polymer starts with number of segments n_s and then n_s lines of description of the segments. The segment descriptions are segments connected on the left and on the right, the segment length (in units of M_e) and weight fraction of the segment in the whole ensemble.

For each polymer we number the segments from zero upwards. Each segment has four connectors two for connecting on the left (L_1 and L_2), two for the right(L_1 and L_2). The easiest way to input the values is by drawing

the polymer on a paper, numbering them and noting the connection. If any of the connectors are not connected, they are *earthed* by putting a -1.

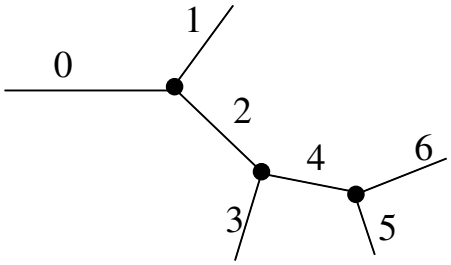
In what follows, we consider the length of nth segment as l_n and the total length of all the polymers is L (both in units of N_e).

First let us consider a linear molecule:

The configuration is represented as:

```
2 // number of segments
-1 -1  1 -1 l_0 l_0/L  // 0 is connected to 1 by R_1
 0 -1 -1 -1 l_1 l_1/L  // 1 is connected to 0 by L_1
```

Slightly more complicated molecule, a comb:




```
7 // number of segments
-1 -1  1  2  l_0 l_0/L   // 0 is connected to 1 (2) by R_1 (R_2)
 0  2 -1 -1  l_1 l_1/L   //1
 0  1  3  4  l_2 l_2/L   //2
-1 -1  2  4  l_3 l_3/L   //3
 2  3  5  6  l_4 l_4/L   //4
 4  6 -1 -1  l_5 l_5/L   //5
 4  5 -1 -1  l_6 l_6/L   //6
```

Chapter 3

Program structure

This chapter is for someone who wants to change the source code. I am assuming familiarity of object oriented programming. The working of **bob** can be subdivided in distinct groups. Firstly a set of files in subdirectory *parser* does exactly what implied - it parses the arguments passed from command line and decides about what to do. Next a set of routines in subdirectory *UI* finds out either interactively or from an input file about the actual polymers to consider. The files in *polygen* generates the polymers. Routines in directory *dyn* follows the relaxation over time and *calc* routines calculates the relaxation spectrum.

3.1 Datatype

subdirectory *include* and *arm_pool* :

Main types used:

```
class arm {
  public :
  int L1,L2,R1,R2,up,down;

  double arm_len, vol_fraction;
  bool relaxing, free_end, compound;

  int relax_end, nxt_relax;

/* The following are used only for free ends */
  int free_up, free_down, nxtbranch1, nxtbranch2;
  double z, dz, pot, gamma2, tau_K;

  double arm_len_eff, arm_len_end, deltazeff;
```

```
  double zeff_numer, zeff_denom;

  bool collapsed, ghost, prune, freeze_arm_len_eff;

  double tau_collapse, phi_collapse, extra_drag, pot_int;

  int next_friction;
};
```

and

```
class polymer{
  public :
   int first_end, first_free, num_branch;
   bool alive, linear_tag;
   double relaxed_frac, ghost_contrib;
   bool rept_set;
   double phi_rept;
};
```

These two classes are used as global variables:

```
arm * arm_pool; int first_avail_in_pool;
polymer * branched_poly;
```

The routines in *arm_pool* are used to initialize and handle the contents of arm_pool.

3.2 Parser

Find out if the mode is batch. If file names are supplied, open them. Else open default files.Main routine

```
int parser(int argc, char *argv[])
```

3.3 UI

Main routine:

```
void user_interface(void)
```

3.4 Polymer configuration

Input about type x is taken by *genx.cpp* and the actual connectivity and lengths are provided by *polygenx.cpp*. Try to keep the naming conventions for the routines out here.

3.5 Relaxation

subdirectory *dyn*. Toplevel code *time_step.cpp*. It calls *retraction*, *extend_arm* and *reptation* in that order.

3.6 Calculation

Calcutate $G(t)$, $G^{'}(\omega)$, $G^{''}(\omega)$.