

CHR Paris, Feb 2020 (version 6.1.1 and 6.1.1r)
 CHR Paris, 4-10 May 1995 (version 6.1)
 CHR Paris, 23 Sept 94 (version 6.0 started 1 September 1994)
 copyright Charles H. Robert

A general way to approach the problem of chain hydrodynamics for mixed globular-chain molecules using my approximate approach. Initial development was with *Mathematica* 2.2. Please note: I don't want to hear any of those catty, thinly-veiled references to the varieties of pasta-code found in programs, etc.

Version 6.1.1 introduces changes resulting from the ReScience 2020 "10-year Reproducibility Challenge", using my article Robert (1995) Biophy J. as the subject of the tests. For this challenge I used *Mathematica* 5.2 and 12.0.

- Boolean variable MonteCarlo changed to montecarlo (*Mathematica* 5.2 defines an object called MonteCarlo)
- Added missing semicolon in function "assembleelements".
- Copied in some functions from earlier versions: getDtrans
- Copied in functions defined previously in s6.0: Monte, MontePK, MonteZimm, doKirkwood0.
- Added warning in comment for use of global variables... which unfortunately are everywhere

Version 6.1 incorporates a correction function to apply to the $\langle r^2 \rangle$ values. As I mention in my paper describing this work, for a stiff chain the difference between $\langle 1/r \rangle$ and $1/\sqrt{\langle r^2 \rangle}$ is not large until the chain is long. When the chain is very long the difference between the two averages is maximal, and one over the average inverse can be calculated to be about 28% of the root mean square. I did a series of Monte carlo studies of Porod-Kratky chains used in these studies to evaluate how this correction behaves as a function of the length of the chain. A sigmoidal representation of this correction is now included in the program in the "correctr" and "doKirkwood" routines.

Version 6.1 also incorporates a less inefficient way of calculating r_{ij} between any two frictional centers of the chain. Instead of calculating all possible transformation matrices between any two elements, I have now simply programmed what I was too lazy to code initially: from a single list of successive products of rotation matrices progressing along the chain, $R_1, R_1R_2, R_1R_2R_3$, etc, which are correspondingly called $A(1), A(2), A(3), \dots$, and along with a list of the inverses $A_{inv}(1), A_{inv}(2)$, etc., one can get from any i to any j in the chain by calculating $A_{ij} = A_{inv}(i) \cdot A(j)$. The inverse cancels off all the unneeded preceding rotations in the product.

Finally, 6.1 loses some of the useless code that was spawned from early trials of 6.0.

■ Chain generators and conformational routines

Needed:

- vectors for two types of chain segments in the reference coordinate system: free-chain

segments (vbp0, for basepairs) and chain displacement (virtual) segments (vnu0, for 'nucleosome'),

- vector for the center of the chain-displacement segment in the reference coordinate system (vc0)
- transformation matrix for any free-chain segment junction (P)
- transformation matrix for any net change in chain direction across the displacement region (R)

Higher level things:

- Set of general mathematical tools

Lower level things:

- Set of definitions for free-chain regions (variables and functions to define free segment objects vbp0 and P)
- Set of definitions for chain displacement regions (variables and functions to define free segment objects vnu0, R)

□ General definitions

```
Off[General::"spell"];
Off[General::"spell1"];
```

```
Ax[w_] := {{1, 0, 0}, {0, Cos[w], Sin[w]}, {0, -Sin[w], Cos[w]}};
Ay[w_] := {{Cos[w], 0, -Sin[w]}, {0, 1, 0}, {Sin[w], 0, Cos[w]}};
Az[w_] := {{Cos[w], Sin[w], 0}, {-Sin[w], Cos[w], 0}, {0, 0, 1}};
```

```
AO := Az[Random[Real, N[{0, 2 π}]]].
      Ax[Random[Real, N[{0, 2 π}]]].Az[Random[Real, N[{0, 2 π}]]];
```

```
norm2[v_] := v.v;
```

```
length[v_] :=  $\sqrt{v.v}$ ;
```

```
rotate[A_, o_] := Module[{i, j, depth}, depth = Depth[N[o]];
  Which[depth == 1, Print["Not done: ", depth];
  Return[o], depth == 2, Return[N[A].o], depth == 3,
  Return[Transpose[N[A].Transpose[o]]], depth == 4, Return[
    (Transpose[N[A].Transpose[#1]] &) /@ o], depth ≥ 5, Print["Not done: ", depth];
  Return[o]]];
```

```
translate[vt_, o_] := Module[{i, j}, depth = Depth[N[o]];
```

```
Which[depth == 1, Print["Not done: ", depth];
Return[o], depth == 2, Return[vt + o], depth == 3, Return[(vt + #1 &) /@ o],
depth == 4, Return[Table[(vt + #1 &) /@ o[[i]], {i, Length[o]}]],
depth ≥ 5, Print["Not done: ", depth];
Return[o]]];
```

□ Free-chain segment (basepair) definitions:

Defines segment transformation matrix P for appropriate model. Here use a chain bending by an angle beta in the direction phi, with a fixed z-axis twist increment alpha. Pav is the same matrix averaged over the bending angle phi, for calculating average properties of a Porod-Kratky-like isotropically bending chain with a fixed twist increment. Otherwise for Monte-Carlo purposes P can be called with phi0, which randomly assigns a bending direction from the interval {0, 2 Pi}.

Constants

```
e10 = 3.4;
persistence = 150 e10;
repeat0 = 10.4;
```

Functions

```
alpha0 := N[ $\frac{2 \pi}{\text{repeat0}}$ ];
phi0 := Random[Real, N[{0, 2 π}]];
beta0 := N[ArcCos[1 -  $\frac{e10}{\text{persistence}}$ ]];
PPK[phi_, beta_] := Az[-phi].Ax[beta].Az[phi];
Pbp[alpha_, phi_, beta_] := Az[-phi].Ax[beta].Az[phi].Az[alpha];
Pavg[alpha_, beta_] := {{Cos[alpha] Cos[ $\frac{\text{beta}}{2}$ ]2, Cos[ $\frac{\text{beta}}{2}$ ]2 Sin[alpha], 0},
  {- (Cos[ $\frac{\text{beta}}{2}$ ]2 Sin[alpha]), Cos[alpha] Cos[ $\frac{\text{beta}}{2}$ ]2, 0}, {0, 0, Cos[beta]}};
definebasepair := Module[{i, j}, Id = IdentityMatrix[3];
  vbp0 = N[e10 {0, 0, 1}];
  cbp0 = N[ $\frac{1}{2}$  e10 {0, 0, 1}];
  P[phi_] := N[Pbp[-alpha0, -phi, -beta0]];
  Pr := N[Pbp[-alpha0, -phi0, -beta0]];
  Prfast := N[PPK[-phi0, -beta0]];
  Pav = N[Pavg[-alpha0, -beta0]]];
```

□ Displacement segment (nucleosome) definitions

The transformation matrix across the nucleosome regions will include a basepair transformation, since at the far end of the nucleosome a bp-bp step occurs. This is arguably not necessary, since the transition from a nucleosomal bp to a free bp is not necessarily the same as that between two free bp, but hell, to first order they are the same.

Constants

```
repeat1 = 10.15;
radius = 44.916;
turns = 1.75;
pitch = -28.571;
```

Functions

```
helix[angle_, radius_, pitch_] := N[ $\sqrt{1 + \left(\frac{\text{pitch}}{2\pi \text{radius}}\right)^2}$  Abs[angle] radius];

wrapbp := Floor[N[ $\frac{\text{helix}[2\pi \text{turns}, \text{radius}, \text{pitch}]}{3.4}$ ]];

dyad :=  $\frac{\text{wrapbp}}{2}$ ;

alpha1 := N[ $\frac{2\pi \text{wrapbp}}{\text{repeat1}}$ ];

beta1 := N[2  $\pi$  turns];

depth1 := N[turns pitch];

angle := N[ArcSin[ $\frac{\text{pitch}}{2\pi \text{radius}}$ ]];

Tya := N[Ay[-angle]];
Tyb := N[Ay[angle]];

wrap[n_, option_] := Block[{i, j, dbeta1, dalpha1, dl, r, vz, vz0}, r = {0, 0, 0};
  spiral = {r};
  dbeta1 = N[ $\frac{\text{beta1}}{n}$ ];
  dalpha1 = N[ $\frac{\text{alpha1}}{n}$ ];
  dl =  $\sqrt{N[(2 \text{radius} \text{Cos}[\frac{\pi - \text{dbeta1}}{2}])^2 + (\frac{\text{depth1}}{n})^2]}$ ;
  Print["n = ", n, " ; dl = ", dl];
  vz0 = {0, 0, dl};
  If[option == 1, vz0 = {0, 1, dl}];
  If[option == -1, vz0 = {0, -1, dl}];
```

```

Do[vz = rotate[N[Ax[-i dbeta1].Tya.Az[-i dalpha1]], vz0];
r = r + vz;
spiral = Append[spiral, r], {i, 1, n}];
Return[rotate[N[Tyb.Ax[ $\frac{dbeta1}{2}$ ]], spiral]]];
Rn[alpha_, beta_] := Tyb.Ax[beta].Tya.Az[alpha];
definenuke := Module[{i, j}, onuke = N[wrap[20, 0]];
vnu0 = Last[onuke];
cnu0 = Tyb.{ $\frac{depth1}{2}$ , -radius, 0};
R = Rn[-alpha1, -beta1];
RPr := N[R.Pr];
RPav = N[R.Pav];];

```

Some nucleosome extras for testing, etc.

```

gamma[n_] := N[ $\frac{1}{2}(\pi - \frac{beta1}{n})$ ];
giveel[r_, n_] :=  $\sqrt{(2r \cos[\text{gamma}[n]])^2 + (\frac{depth1}{n})^2}$ ;
giver[el_, n_] :=  $\frac{\sqrt{el^2 - (\frac{depth1}{n})^2}}{2 \cos[\text{gamma}[n]]}$ ;

```

□ **Individual-chain definitions: vectors, transformation matrices, chain generators, $|r(i,j)|$ length functions, drawing routines**

In these routines i, j, and k are indices of the chain segments. They indicate actual segment positions. The list 'np' contains the chain length and a list of of segments (positions) at which a chain displacement occurs (nucleosomal segments).

```

v0[k_, positions_] := If[MemberQ[positions, k], Return[vnu0], Return[vbp0]];
c0[k_, positions_] := If[MemberQ[positions, k], Return[cnu0], Return[cbp0]];
T[k_, kiQ, positions_] :=
  If[kiQ, Return[Id], If[MemberQ[positions, k - 1], Return[RPr], Return[Pr]]];

```

```

rij[i_, j_] := r[[j - 1]] + clist[[j]] - (r[[i - 1]] + clist[[i]]);
lrij[i_, j_] := length[rij[i, j]];

```

```

fastPKr2[n_] := Module[{M, r}, M = IdentityMatrix[3];
r = {0, 0, 0};

```

```

Do [M = M.Prfast;
    r = r + M.vbp0, {k, n}];
Return [norm2[r]];

```

Define a function of the average end-to-end length for a chain with a given persistence length.

```

rootr2[n_] := Sqrt[2 persistence n e10
    (1 - persistence(1 - Exp[-n e10/persistence])/(n e10))];

```

□ Averaged-chain definitions: matrices and end-to-end length functions.

```

Ta[k_, kiQ_, positions_] :=
    If[kiQ, Return[Id], If[MemberQ[positions, k - 1], Return[RPav], Return[Pav]]];

```

```

makeAs[np_] := Module[{x}, n = np[[1]];
    positions = np[[2]];
    A = Table[0, {n}];
    A[[1]] = IdentityMatrix[3];
    Ai = Table[0, {n}];
    Ai[[1]] = IdentityMatrix[3];
    SAa = Table[0, {n}];
    SAa[[1]] = Pav;
    Do[A[[j]] = A[[j - 1]].Ta[j, False, positions];
        Ai[[j]] = Inverse[A[[j]]];
        SAa[[j]] = SAa[[j - 1]] + MatrixPower[Pav, j], {j, 2, n}];

```

```

rcij[i_, j_, np_] := Module[{v, vh, vi, s, positions}, positions = np[[2]];
    vk = -c0[i, positions] + v0[i, positions];
    s = vk.vk;
    s = s + Sum(vk = v0[k, positions];
        vk.vk);
    vk = c0[j, positions];
    s = s + vk.vk;
    Do[If[h == i, vh = -c0[h, positions] + v0[h, positions], vh = v0[h, positions]];
        nointerrupt = (Range[h, j - 1]) ∩ positions == {};
        If[nointerrupt, If[h == j - 1, s2 = vh.Pav.c0[j, positions],
            s2 = vh.SAa[[j - 1 - h]].vbp0 + vh.MatrixPower[Pav, j - h].c0[j, positions]],
        s2 = Sum(If[k == j, vk = c0[k, positions], vk = v0[k, positions]];
            vh.Ai[[h]].A[[k]].vk)];

```

```
s = s + 2 s2, {h, i, j - 1}];
Return[ $\sqrt{s}$ ];];
```

■ Friction calculations (22 Sept 94)

We'll use a notation in which the letter 'a' denotes properties for segments that can be treated individually, such as a nucleosome, and 'b' denotes the properties of a collection of segments, like a chain of basepairs. An individual basepair has 'a' properties as well, since if necessary a chain of basepairs can be treated at the level of the individual basepairs. This is time-consuming, however, and is not necessarily better. Any legitimate element is represented by a single spherical frictional element with a certain mass, Stokes radius, and a 'segment address' in the superchain, which is used to calculate the distances between pairs of elements. These distances will be calculated using the chain conformational properties defined in the preceding sections.

As far as units are concerned, I will try to use the cgs units for convenience, since all of the frictional properties seem to be defined this way. But I will retain the simpler units of Angstroms for Stokes radii and bp for chain lengths, in order to make defining the properties of the DNA chain and the nucleosome easier and more intuitive. These are converted where necessary inside the various friction routines.

□ General Definitions

Physical parameters are typical values relative to water taken from van Holde's Physical Biochemistry book (2nd ed., p.134 and p.166). In as much as water's values would be $\rho = 1 \text{ g/cm}^3$ and $\eta = 1 \text{ centipoise}$, they will be reasonable ballpark estimates for a buffer solution near 20 C (293 K). These values should be changed to reflect the true experimental conditions.

```
NAv =  $6.02 \times 10^{23}$ ;
ρ = 1.003;
η = 0.01016;
```

```
gettrans[x_, tk_] := Module[{i}, kT =  $\frac{8.31433 \text{ tk kg m}^2}{(\text{mol s}^2) (6.023 \times 10^{23})}$ ;
Return[ $\frac{kT 1000 \text{ g} (\frac{100 \text{ cm}}{\text{m}})^2}{\text{kg x}}$ ];
```

Added getDtrans from s6.0

```

getDtrans[f_]:=Module[{i},
  (* Convert a frictional coefficient (g/s) into a
    translational diffusion coefficient (cm^2/s)
    at 20 degrees C. *)
  kT=(8.31433 mol^-1 293.15 kg m^2/s^2)/(6.023 10^23 mol^-1);
  Return[(kT 1000 g/kg (100 cm/m)^2)/f1]
];

```

- **Chain elements: particle definitions (basepairs) mbpa,rbpa
and collective definitions (uniform chains) mbpb[n], rbpb[n]**

Constants

```

mbpa = 660;
rbpa = 1.8294;
vDNA = 0.55;
dDNA = 27;
switchbp = 50;

```

Functions

```

mbpb[n_] := n mbpa;

rellipsoid[n_, offset_] := Module[{p, pdiff, foverf0, requiv, rb}, p =  $\frac{n \text{el0}}{dDNA}$ ;

  If[p > 1, pdiff =  $\sqrt{1 - \left(\frac{1}{p}\right)^2}$ ;

    foverf0 =  $\frac{pdiff}{\left(\frac{1}{p}\right)^{2/3} \text{Log}[p (1 + pdiff)]}$ , p =  $\frac{1}{p}$ ;

    pdiff =  $\sqrt{p^2 - 1}$ ;

    foverf0 =  $\frac{pdiff}{p^{2/3} \text{ArcTan}[pdiff]}$ ;

    requiv =  $\frac{(n \text{el0 } dDNA^2)^{1/3}}{2 \times 10^8}$ ;

    fb = foverf0 (6  $\pi$  visc requiv);

    rb = N[ $\frac{fb 10^8}{6 \pi \text{ visc}}$  + offset];

    Return[rb]];

rYF[n_] := Module[{masstrue, massapp, sb, fb, rb}, masstrue = mbpb[n];

  massapp =  $\frac{masstrue 575}{\text{persistence}}$ ;

```



```

sb =  $\frac{1}{10^{13}}$  (N[3.620
      (Log[10, masstrue] - 3.552 +  $\frac{3.23 \text{ massapp}}{10^7}$  +  $\frac{1.63 \text{ massapp}^2}{10^{13}}$  -  $\frac{7.85 \text{ massapp}^3}{10^{20}}$ )]);
fb =  $\frac{\text{masstrue} (1 - \text{ro vDNA})}{\text{NAvo sb}}$ ;
rb = N[ $\frac{\text{fb}}{6 \pi \text{ visc}}$ ];
Return[rb 108];];
findoffset := Module[{x}, offset = rYF[switchbp] - rellipsoid[switchbp, 0];
  Print["Stokes offset ", offset, " A at ", switchbp, " bp"];];
rbpb[n_] := If[n > switchbp, Return[rYF[n]],
  If[n > 0, Return[rellipsoid[n, offset]], Return[0]]];

definebasepair;
findoffset

Stokes offset 3.00764 A at 50 bp

rbpb[10]; Print[N[fb]]

-8
2.80519 10

```

□ **Chain-displacement elements (nucleosomes): particle definitions mnua, rnua only.**

Constants

```

mhistone = 108 500;
vhistone = 0.75;
score =  $\frac{10.7}{10^{13}}$ ;
mcore = 146 mbpa + mhistone;
vcore =  $\frac{146 \text{ mbpa vDNA} + \text{mhistone vhistone}}{\text{mcore}}$ ;

```

Functions

```

mnu[n_] := mhistone + n mbpa;
vbar[n_] :=  $\frac{n \text{ mbpa vDNA} + \text{mhistone vhistone}}{\text{mnu}[n]}$ ;
rcore := N[ $\frac{\text{mcore} (1 - \text{ro vcore}) 10^8}{6 \pi \text{ visc NAvo score}}$ ];
rnua[n_] := N[rcore (  $\frac{\text{vbar}[n] \text{ mnu}[n]}{\text{vcore mcore}}$  )1/3];

```

```

definephysicalnuke := Module[{x}, mnu = mnu[wrapbp];
  rnua = rnu[wrapbp];];

report1 := Module[{x, names, values}, names = {"el0",
  "persistence\nlength", "repeat0", "repeat1", "radius", "turns", "pitch"};
  values = {el0, persistence, repeat0, repeat1, radius, turns, pitch};
  units = {"A", "A", "bp/turn", "bp/turn", "A", "turns", "A/turn"};
  Print[MatrixForm[Transpose[{names, values, units}]]];
report2 := Module[{x}, Print["Array of chain lengths (bp)"];
  Print[bns];
  Print["Array of friction element positions (bp)"];
  Print[segs];
  Print["Array of Stokes radii (A)"];
  Print[radii];];

```

□ Create frictional description of chain

As far as the conformational routines were concerned, a chain is specified by its length and by the positions of the displacement segments. For the frictional calculations, one needs to properly assign the Stokes radii of the elements and the segment at which the frictional effect is placed. Each frictional element in a list of elements can be either an individual segment or a collection of segments, so it is important to assign the properties appropriately.

I will assume that there are two choices to be made: first, whether the uniform chain regions will be treated as individual links or as a collection, and second whether the conformational properties should be calculated by $1/\text{Sqrt}(\langle r^2 \rangle)$ (faster) or by Monte-Carlo simulation of $\langle 1/|r| \rangle$ for a large sample of pseudorandom chains. These choices are indicated by the Boolean variables 'collective' and 'montecarlo'.

Note there is no error checking, so the chain lengths and nuke positions should be consistent! This is not a commercial program. Remember, the nucleosome occupies one segment. A segment is not a basepair, it is a logical segment of the whole superchain.

```

assembleelements[np_] := Module[{n, p, pp}, n = np[[1];
  p = np[[2];
  If[progress, Print["Chain of ", n, " segments,"];
    Print["with displacements at segments ", p]];
  If[collective, asecs = p;
    aStokes = Table[rnua, {i, Length[p]}];
    findoffset;
    pp = Prepend[Append[p, n + 1], 0];
    bns = Table[pp[[i]] - pp[[i - 1]] - 1, {i, 2, Length[pp]}];
    bsecs = Take[pp, Length[pp] - 1] + Floor[ $\frac{bns}{2}$ ];
    bStokes = rbp / @ bns;
    bb = Transpose[{bsecs, bStokes}];
    bb = Select[bb, #1[[2]] > 0 &];
    {bsecs, bStokes} = Transpose[bb];
    x = Join[asecs, bsecs];
    y = Join[aStokes, bStokes];
    {secs, radii} = Transpose[Sort[Transpose[{x, y}]]], secs = Range[n];
    radii = Table[If[MemberQ[p, i], rnua, rbpa], {i, n}]; report1;
    report2;];

```

```

correctr[n_] := Module[{a, b, m}, a = 0.7238;
  b = 0.002040;
  m = 1.9436;
  Return[a +  $\frac{1 - a}{1 + b^m n^m}$ ];

```

```

sediment[np_, f1_] := Module[{n, positions, nnukes, nchain, mass1, vbar1}, n = np[[1];
  positions = np[[2];
  nnukes = Length[positions];
  nchain = n + nnukes (wrapbp - 1);
  mass1 = mbpb[nchain] + nnukes mhistone;
  vbar1 =  $\frac{mbpb[nchain] \text{vDNA} + nnukes mhistone \text{vhistone}}{mass1}$ ;
  s1 =  $\frac{mass1 (1 - ro \text{vbar1}) g}{NAvo f1}$ ;
  If[test, Print[{"np", "nchain", "mass1", "vbar1"}];
    Print[{np, nchain, mass1, vbar1}]];
  Return[s1];

```

```

doKirkwood[np_] :=
Block[{i, i0, j, j0, n, positions, ntot, rirj, delsum, sum, sumradii, ffree},
  If[montecarlo, rxij = rij, rxij = rcij];
  ntot = Length[radii];
  If[progress, Print["...doKirkwood..."]];

  sumradii =  $\sum_{i=1}^{ntot} radii[[i]]$ ;

  sum = 0;
  Do[i0 = segs[[i]];
    Do[rirj = radii[[i]] radii[[j]];
      j0 = segs[[j]];
      If[rirj > 0, rijav = rxij[i0, j0, np]];

      delsum =  $\frac{rirj}{correctr[j0 - i0] rijav}$ , delsum = 0];
  If[test, If[nointerrupt, rr = "**", rr = " "];
    Print["i,j,rij:", rr, i, " ",
      j, " ", rijav, " corrected:", correctr[j0 - i0] rijav]];
  sum = sum + delsum, {j, i + 1, ntot}], {i, ntot}];

  ffree =  $\frac{N[\frac{6 \pi \text{visc sumradii}}{10^8}] g}{s}$ ;

  f1 =  $\frac{N[\frac{6 \pi \text{visc sumradii}}{10^8 (1 + \frac{2 \text{sum}}{\text{sumradii}})] g}{s}$ ;

  s1 = sediment[np, f1];
  If[progress, Print["  Kirkwood results: {f free-draining, f1, s1}"];
    Print["    ", N[{ffree, f1, s1}]]];];

```

Added doKirkwood from s6.0 (without a small correction for r^2 average) as doKirkwood0

```

doKirkwood0[np_] := Block[{i, i0, j, j0,
  ntot, rirj, delsum, sum, sumradii, ffree},
  (* Calculates friction and sedimentation coefficients, f1 and
    s1, according to the standard Kirkwood-Riseman theory. *)
  If[montecarlo, rxij=rj, rxij=rcij];
  ntot=Length[radii];
  If[progress, Print["\n...doKirkwood..."]];
  sumradii=Sum[radii[[i]], {i, ntot}];
  sum=0;
  Do[ i0=segs[[i]];
    Do[ rirj=radii[[i]] radii[[j]];
      j0=segs[[j]];
      If[ rirj>0,
        delsum=rirj/rxij[i0, j0, np],
        delsum=0
      ];
      If[ test,
        If[nointerrupt, rr="*", rr=" "];
        Print["i,j,rj:", rr, i, " ", j, " ", rxij[i0, j0, np]]
      ];
      sum=sum+delsum,
    {j, i+1, ntot}],
  {i, ntot}];
  (* Notice we only counted the distance of each pair once,
    but we multiplied by two to make up for it. Now make
    frictional coeff after converting from A to cm: *)
  ffree=N[6 Pi visc sumradii 10^-8] g/s;
  f1=N[6 Pi visc sumradii 10^-8/(1 + 2 sum/sumradii)] g/s;

  (* s1=mass g (1-ro vbar[nukes])/(NAvo f1) g;
    s1K=s1; *)

  If[progress, Print["  Kirkwood results: {f free-draining, f1, s1}"];
    Print["  ", N[{ffree, f1, s1}]]];
];
];

```