Supporting Information for

"Fitting of dihedral terms in classical force fields as an analytic linear least squares problem"

Chad W. Hopkins† *and Adrian E. Roitberg**,‡

† Department of Physics, Quantum Theory Project, University of Florida, Gainesville, Florida 32611, United States

‡ Department of Chemistry, Quantum Theory Project, University of Florida, Gainesville, Florida 32611, United States

Corresponding Author

*E-mail: roitberg@ufl.edu

<u>Description of the LLS dihedral parameters fitting routine</u>

<u>Input</u>:

- *N*
    - Integer
    - Number of data points
- *d*
    - Integer
    - Number of dihedral angles being fit in the molecule
- *p*
    - Integer
    - Number of unique dihedral types (unique sequences of 4 atom types) out of the dihedral angles being fit
- *qme*
    - array[*N*]
    - QM energy for each data point
- *mme*
    - array[*N*]
    - MM energy for each data point
- *ang*
    - 2D array[*d*][*N*]
    - dihedral angle values for each dihedral angle being fit, for each data point
- *type*
    - 2D array [*p*][variable length]
    - List of unique dihedral types (unique sequence of atom types); each entry contains a list of dihedral indices that are of that type
- *nmax*
    - Integer
    - Maximum multiplicity being fit
- *phase*
    - Flag
    - Specifies whether phase constants are allowed to vary freely or are restricted to 0 or 180°
- *ini*
    - array[2][2**p***nmax*]
    - Initial guesses for parameters used in creating *mme*, as well as force constants for harmonic potentials to be enforced on varying these initial parameters. *ini*[1][1:*p*nmax*] should be the force constants and *ini*[1][*p*nmax*+1:2**p*nmax*] should be the phase constants. The force constants for the first dihedral type being fit should be in *ini*[1:*nmax*], the second type's should be in *ini*[*nmax*+1:2**nmax*], and so on, with the phase constants section formatted identically. The second column will be the harmonic potential force constant

applied to the corresponding parameter in the first column – if it is desired to allow a parameter to vary freely, then set the force constant to zero. Only the second column of the dihedral force constant section (*ini*[2][1:*p***nmax*]) is read (see below note).

    - VERY IMPORTANT NOTE: the same harmonic potential is applied to the (dihedral) force constant and corresponding phase constant; i.e., it is impossible to specify a different harmonic potential force constant for the dihedral force constant and corresponding (same dihedral, same periodicity) phase constant – this is due to the fact that the original parameters are being mixed and converted to the linear form for the fitting, and a harmonic potential on the original force constant/phase constant only makes sense if taken pairwise (per corresponding force/phase constant pair)

- *w*
    - array[*N*]
    - Weight values for each data point (default: all 1)

<u>Output</u>:

- *a*
    - array[2**p***nmax*]
    - Fitted parameters, defined in Eq. 10. The formatting will be identical to that of the first column of *ini* (see above).

<u>Procedure</u>:

```
if phase then npar = 2*p*nmax, else npar = p*nmax
initialize array k[npar] to 0
initialize 2D array C[npar][npar] to 0
initialize array ini_alt[2*p*nmax] to 0
for m = 1 to p*nmax:
  ini_alt[m] = ini[1][m]*cos(ini[1][p*nmax+m])
  ini_alt[p*nmax+m] = ini[1][m]*sin(ini[1][p*nmax+m])
for tti = 1 to p:
  for n = 1 to nmax:
    k[(tti-1)*nmax+n] += ini[2][(tti-1)*nmax+n]*ini_alt[(tti-1)*nmax+n]
    if phase then
      k[p*nmax+(tti-1)*nmax+n] += ini[2][(tti-1)*nmax+n]*ini_alt[p*nmax+(tti-1)*nmax+n]
    for i = 1 to N:
      for ti = 1 to length(type[tti]):
        k[(tti-1)*nmax+n] += w[i] * (qme[i] – mme[i]) * cos(n*ang[type[tti][ti]][i])
        if phase then
          k[p*nmax+(tti-1)*nmax+n] += w[i] * (qme[i] – mme[i]) * sin(n*ang[type[tti][ti]][i]) +
for tti1 = 1 to p:
  for n1 = 1 to nmax:
    C[(tti1-1)*nmax+n1][(tti2-1)*nmax+n2] += ini[2][(tti1-1)*nmax+n1]
    if phase then
      C[p*nmax+(tti1-1)*nmax+n1][p*nmax+(tti1-1)*nmax+n1] += ini[2][(tti1-1)*nmax+n1]
    for tti2 = tti1 to p:
      for n2 = n1 to nmax:
        for i = 1 to N:
          for ti1 = 1 to length(type[tti1]):
            for ti2 = 1 to length(type[tti2]):
              C[(tti1-1)*nmax+n1][(tti2-1)*nmax+n2] +=
                        w[i] * cos(n1*ang[type[tti1][ti1]][i]) * cos(n2*ang[type[tti2][ti2]][i])
             if phase then
               C[p*nmax+(tti1-1)*nmax+n1][(tti2-1)*nmax+n2] +=
                        w[i] * cos(n1*ang[type[tti1][ti1]][i]) * sin(n2*ang[type[tti2][ti2]][i])
               C[p*nmax+(tti1-1)*nmax+n1][p*nmax+(tti2-1)*nmax+n2] +=
                        w[i] * sin(n1*ang[type[tti1][ti1]][i]) * sin(n2*ang[type[tti2][ti2]][i])
      C[(tti2-1)*nmax+n2][(tti1-1)*nmax+n1] = C[(tti1-1)*nmax+n1][(tti2-1)*nmax+n2]
      if phase then
        C[(tti2-1)*nmax+n2][p*nmax+(tti1-1)*nmax+n1] =
            C[p*nmax+(tti1-1)*nmax+n1][(tti2-1)*nmax+n2]
        C[p*nmax+(tti2-1)*nmax+n2][p*nmax+(tti1-1)*nmax+n1] =
            C[p*nmax+(tti1-1)*nmax+n1][p*nmax+(tti2-1)*nmax+n2]
C_inv = invert C
initialize array delta[2*p*nmax] to 0
for m = 1 to npar:
  for l = 1 to npar:
    delta[m] += C_inv[m][l] * k[l]
initialize array a_alt[2*p*nmax] to 0
for m = 1 to 2*p*nmax:
  a_alt[m] = delta[m]
  if ini[m] == 0 then
    a_alt[m] += ini_alt[m]
initialize array a[2*p*nmax] to 0
for m = 1 to p*nmax:
  a[m] = sqrt(a_alt[m]^2 + a_alt[p*nmax+m]^2)
  a[p*nmax+m] = arctan(a_alt[p*nmax+m]/ a_alt[m])

return a
```