

# **SolvMol: Code for the calculation of solvation thermodynamics by NRF method**

Center for Biological Physics, Arizona State University,  
PO Box 871604, Tempe, AZ 85287-1604

September 3, 2016

Major publications and the history of the project:

- Chem. Phys. **174**, 199 (1993): The perturbation scheme for the calculation of electron transfer reorganization energy was suggested. The solvation parameters were expressed in terms of the longitudinal structure factor of dipolar polarization and the density structure factor. The main experimentally-testable prediction was a negative slope of the reorganization energy with increasing temperature.
- J. Phys. Chem. A **103**, 7888 (1999); J. Am. Chem. Soc. **121**, 7108 (1999); J. Phys. Chem B **103**, 9130 (1999); J. Phys. Chem. A **104**, 2626 (2000), the theory prediction was confirmed by experiment.
- J. Phys. Chem. **107**, 14509 (2003) application of the theory to ET in DNA
- JCP **120**, 7532 (2004) extension of the theory to both longitudinal and transverse components of the polarization response. This is the main theoretical work behind the development of the code's algorithm.
- JCP **122**, 044502 (2005), extension of the theory to solvation dynamics
- CP **324**, 172 (2006), theory tested on simulations of polypeptide in TIP3P water
- JCP **122**, 155106 (2008), application of the code to the solvation and redox thermodynamics of plastocianin.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>List of Routines</b>	<b>3</b>
<b>3</b>	<b>Important Parameters</b>	<b>4</b>
<b>4</b>	<b>Solute input</b>	<b>4</b>
<b>5</b>	<b>Input solvent</b>	<b>4</b>
<b>6</b>	<b>Subroutine dGibbssolvent</b>	<b>4</b>
<b>7</b>	<b>Subroutine dynamics</b>	<b>6</b>
<b>8</b>	<b>Subroutine FOURIER</b>	<b>7</b>
8.1	Charged solutes and Chebyshev approximation . . . . .	8
<b>9</b>	<b>Code Testing</b>	<b>9</b>
9.1	Ion test . . . . .	9
9.2	Dipole test . . . . .	9
<b>10</b>	<b>Subroutine deltaGdens</b>	<b>10</b>
10.1	Subroutine DIRECT-CORR . . . . .	10
10.2	Subroutine cPY . . . . .	10
<b>11</b>	<b>Subroutine SPHERE</b>	<b>10</b>
<b>12</b>	<b>Subroutine FIELD</b>	<b>11</b>
<b>13</b>	<b>Configuration and Installation</b>	<b>12</b>
13.1	Fast Fourier Transform Library . . . . .	12
13.2	Supported Fortran Compilers . . . . .	14
<b>14</b>	<b>Running</b>	<b>14</b>
14.1	Polypeptide . . . . .	14
14.1.1	Output File . . . . .	15
14.2	Plastocyanin . . . . .	15
14.2.1	Output file . . . . .	16
14.3	p-nitroaniline . . . . .	16
14.3.1	Output file . . . . .	17
14.4	Quinoxaline . . . . .	17
<b>A</b>	<b>Code Listing</b>	<b>19</b>
A.1	Main code . . . . .	19
A.2	Electric Field Calculation . . . . .	36
A.3	Fourier Transform . . . . .	40
A.4	Density Response . . . . .	49
A.5	Stokes shift dynamics . . . . .	55

# 1 Introduction

The aim of this code is to calculate the *free energy of solvation* and *solvent reorganization energy* of an arbitrary solute in the polar solvent of hard sphere molecules with point dipoles. It is assumed that the solute can be approximated by a number of spheres, which are allowed to overlap. The coordinate and charges are read from either a `pdb` file or from externally supplied files. The solute geometry is stored in the array `solute(n_sphere,4)`:

x,y,z,radius

The code calculates the difference of solvation free energies in the Ox and Red states of a molecule,  $\Delta G_s$ , and the solvent reorganization energy of electron transfer (ET),  $\lambda_s$ :

$$\Delta G_s = -\Delta \mathbf{E}_0 * \chi * \bar{\mathbf{E}}_0 \quad (1)$$

and

$$\lambda_s = \frac{1}{2} \Delta \mathbf{E}_0 * \chi_n * \Delta \mathbf{E}_0. \quad (2)$$

In these equations,  $\Delta \mathbf{E}_0$  is the difference of electric fields in the two ET states and  $\bar{\mathbf{E}}_0 = (\mathbf{E}_1 + \mathbf{E}_2)/2$  is the mean field. All fields are calculated in reciprocal  $\mathbf{k}$ -space by taking the integral over the volume  $\Omega$  outside the solute repulsive core:

$$\mathbf{E}_0 = \int_{\Omega} \mathbf{E}_0(\mathbf{r}) e^{i\mathbf{k}\cdot\mathbf{r}} s\mathbf{r}. \quad (3)$$

The response function  $\chi$  corresponds to the total response of the solvent (permanent+induced dipoles) and  $\chi_n$  gives the response of permanent dipoles only. Correspondingly, the charge distribution is given by two arrays `delta_charge(n_mean_charge,4)` and `mean_charge(n_delta_charge,4)`:

x,y,z,charge

Equations 1 and 2 give the response of the solvent by orientational motions of the dipoles. In addition, there is a response produced by density fluctuations,  $\Delta G_d$  and  $\lambda_d$ .<sup>2,3</sup>

## 2 List of Routines

- `deltaG.f` (main routine)
- `fieldMFdeltaG.f` (calculates real-space electric field on 3D lattice)
- `fourierDeltaG.f` (Fourier transform of the field)
- `deltaGdens.f` (calculates the density component of the solvent response)
- `scfy.f` (calculates  $y_{eff}$  and  $y_{\infty}$  according to Wertheim's theory)
- `input_full.f` (includes the list of properties of more than 100 common laboratory and model solvents with their parameters needed for calculations. A shorter list gives temperature derivatives of the high-frequency and static dielectric constants)
- `pmsa.f` (calculates polarity parameters needed for the polarization structure factors of polar liquids)
- `sphereDeltaG.f` (calculates the Fourier transform of the electric field outside the spherical cutoff)
- `random.f` (calculates the volume of the solute by Monte Carlo integration)

### 3 Important Parameters

The size of the grid is determined by the parameter `nc` which is determined in the main routine `deltaG.f`

```
parameter (nc = 7, n_grid = 2**nc, n_k = 200)
```

The use of `nc = 9` corresponds to  $512^3$  grid, which requires  $> 3$  Gb of memory.

`dr` is the grid step. It is defined from the largest dimension of the solute by multiplying by `factor` and dividing by the number of grid points.

```
size0 = size + sigma
size  = factor * size ! the size of the integration box is scaled
                      ! with the FACTOR

dr  = size/float(n_grid) ! increment in the coordinate space
```

`radius` is the effective radius of the solute calculated from its volume. The volume is calculated by 3D MC integration.

```
c calculation of the effective vdW radius of the molecule
write(*,916)
call volume1(epsilon, volume)
write(*,919) volume, epsilon
radius = (3./4./pi*volume)**0.3333333
```

### 4 Solute input

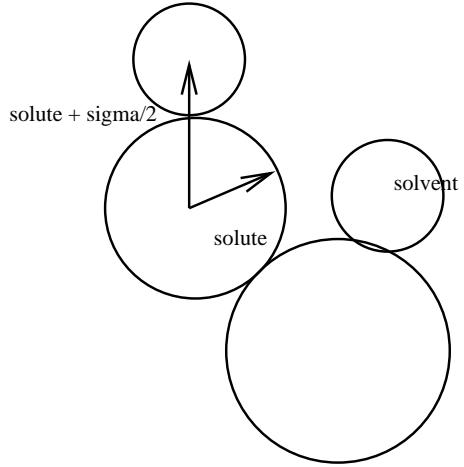
The solute is approximated by a set of spheres of the radii `solute(i,4)`; the number of spheres is given by `n_sphere`.

### 5 Input solvent

The `input_solvent` routine contains the database of the solvent properties in `database`. The main program provides the name of the solvent and, if it is found in the database, its parameters are read into the array `solvents`.

### 6 Subroutine dGibbssolvent

The routine main starts with reading the geometric parameters of the solute and the solvent parameters, coordinates of the charges, etc. Then a loop over the solvents specified in the input is performed. First, the radius of the solvent is added to each sphere representing the solute. This is the distance of the closest approach to solute `solute=solute+sigma/2`.



The center of the molecules with coordinates `x_center`, `y_center`, `z_center` is taken as the geometrical center of the molecule:

$$\mathbf{r}_c = N^{-1} \sum_j \mathbf{r}_j. \quad (4)$$

The maximum dimension of the solute (`size0`) is established as the maximum distance from  $\mathbf{r}_c$  to the edge of the molecule  $+\sigma$ . This is done with the calculation of proteins in mind where charges are often located close to the surface. The size of the box on which the Fourier transform is performed is obtained by multiplying `size0` by the variable `factor`. Test runs suggest that `factor` between 5 and 10 gives sufficient accuracy of the direct Fourier transform on one hand and sufficiently small step in the inverted space on the other. The call `call geometry(1.)` calculates the solute volume and effective radius. If 0. is specified in the variable, volume is not calculated and the effective radius (`radius`) is equal to `size0`.

After the `geometry` call, the temperature loop starts.

```

temperature: do ii = 1, n_temperature

    tempr = T_initial + (real(ii) - 1.) * T_step

    call scfy(tempr, 298., mu, 0., sigma, 0., eta, alphap,
    :           alpha, sqrt(ei0), eidT, yeff, yinf, yquad)

```

This calculates the effective density of dipoles `yeff` followed by the dielectric constants and density in the linear approximation as a function of temperature.

```
call deltaGdens(etaT, dr)
```

This calculates the density components  $\Delta G_d$  and  $\lambda_d$ . The calculation is done at each temperature because of the solvent expansion.

The longitudinal and transverse projections of the field are calculated once, at the first temperature. The longitudinal and transverse projectios are fitted to Chebyshev polinomials and then used for all subsequent calculations at different temperatures.

```

allocate(delta_field_x(0:n1+2,0:n1,0:n1),
:           delta_field_y(0:n1+2,0:n1,0:n1),

```

```

:           delta_field_z(0:n1+2,0:n1,0:n1)  )

call  fourier(dr, n_cheb, chebL, chebT, chebLd, chebTd )

deallocate(delta_field_x, delta_field_y, delta_field_z)
deallocate(mean_field_x, mean_field_y, mean_field_z)
deallocate(scDelta, siDelta, scMean, siMean)

```

The call of `fourier` return Chebyshev interpolates for the L and T components of  $\Delta\mathbf{E}_0 \cdot \bar{\mathbf{E}}_0$  (`chebL`, `chebT`) and of  $\Delta\mathbf{E}_0 \cdot \Delta\mathbf{E}_0$  (`chebLd`, `chebTd` ).

## 7 Subroutine dynamics

The routine `dynamics(namesolv, dr, n_cheb, chebLd, chebTd)` calculates the Laplace transform of the Stokes shift correlation function according to the procedure developed in Ref. 4. The Laplace transform of the electrostatic solute-solvent interaction can be written as:

$$E(s) = -s^{-1} \Delta\tilde{\mathbf{E}}_0 * \chi(s) * \Delta\tilde{\mathbf{E}}_0, \quad (5)$$

where  $\chi(\mathbf{k}', \mathbf{k}'', s)$  is the polarization response function and the Fourier transform of the difference field  $\Delta\tilde{\mathbf{E}}_0(\mathbf{k})$  is calculated by the code (routine `FOURIER`). Since the above equation has singularity at  $s = 0$ , the output of the code is in fact the function

$$F(s) = -sE(s) \quad (6)$$

from which the Stokes shift correlation function is obtained as

$$S(t) = \frac{\int_t^\infty F(t')dt'}{\int_0^\infty F(t')dt'}. \quad (7)$$

The input required for the calculations is the frequency-dependent dielectric constant. The experimental results for  $\epsilon(\omega)$  for a few solvents are listed in the routine `dielectric_data(namesolv)`. The dielectric constant is defined as a linear combination of the Cole-Davidson functions

$$\epsilon(s) = \epsilon_\infty + \sum \frac{\Delta\epsilon_i}{(1 + s\tau_i)^{\beta_i}} \quad (8)$$

The first relaxation time is set up to one so that the data are reported in units of  $s\tau_1$ . The dielectric constant is calculated in routine `function epsilon_s(s)` which codes Eq. (8). The output of Stokes shift calculations is written to the file `StokesFile = 'Stokes.dat'` from `DYNAMICS` routine.

The tabulation of the response function depends on the values of the structure factors adopted at  $k = 0$ . Those are calculated from the prescription used in JCP'05 paper

$$S^L(0) = c_0/3y_n, \quad (9)$$

where  $c_0$  is the Pekar factor and  $y_n$  is the density of permanent dipoles  $y_n = (4\pi/9)\beta\rho m'^2$ . The transverse structure factor is calculated from the Kirkwood factor

$$g_K = \frac{[\epsilon(s) - 1][2\epsilon(s) + 1]}{9\epsilon(s)y_{eff}}, \quad (10)$$

where  $y_{eff} = y_n + (4\pi/3)\rho\alpha$ . The  $k = 0$  value of the transverse structure factor is then

$$S^T(0) = (3g_K - S^L(0))/2. \quad (11)$$

## 8 Subroutine FOURIER

The routine FOURIER is the main computational unit. It uses the fast Fourier library routines from fftw to calculate the Fourier transforms of the electric fields.

```
write(*,915)
call field(r0, dr)
write(*,920)
```

First, the routine calls FIELD to create the  $x, y, z$ -components of the field in the space between the donor-acceptor complex and a sphere of the radius `size0`. This is the most memory-intensive part of the code. Then it uses fftw to calculate the Fourier transform of the field:

```
call rfftw3d_f77_create_plan(plan,nn,nn,nn,
:           FFTW_REAL_TO_COMPLEX, FFTW_MEASURE + FFTW_IN_PLACE)
call rfftwnd_f77_one_real_to_complex(plan, delta_field_x, 0)
write(*,*) 'FFT of delta_field_x on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, delta_field_y, 0)
write(*,*) 'FFT of delta_field_y on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, delta_field_z, 0)
write(*,*) 'FFT of delta_field_z on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_x, 0)
write(*,*) 'FFT of mean_field_x on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_y, 0)
write(*,*) 'FFT of mean_field_y on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_z, 0)
write(*,*) 'FFT of mean_field_z on ',nn,'^3 lattice complete.'
call rfftwnd_f77_destroy_plan(plan)
write(*,*) 'PLAN DESTROYED, FFTW exited normally.'
write(*,910)
```

It creates six arrays defined on the 3D grid of  $k$ -values. The routine ANGULAR forms one-dimensional arrays for the angular averaged fields. The array `field_lt(n2,4)` contains all the angular averaged projections. `field_lt(n2,1)` is the longitudinal field

$$\langle \Delta \tilde{\mathbf{E}}_0 \cdot \mathbf{k} \bar{\mathbf{E}}_0 \cdot \mathbf{k} \rangle_{\Omega_k} \quad (12)$$

`field_lt(n2,2)` is the transverse component

$$\langle \Delta \tilde{\mathbf{E}}_0 \cdot \bar{\mathbf{E}}_0 \rangle_{\omega_k} - \langle \Delta \tilde{\mathbf{E}}_0 \cdot \mathbf{k} \bar{\mathbf{E}}_0 \cdot \mathbf{k} \rangle_{\Omega_k} \quad (13)$$

`field_lt(n2,3)` is the modified longitudinal field obtained from structure factors describing the whole solvent response

$$\mathcal{E}^{eff}(\mathbf{k}) = |\Delta \tilde{E}_0^L|^2 - |\Delta \tilde{E}_0^T|^2 \frac{f_p}{3} \frac{\mathbf{F}_0 \cdot \hat{\mathbf{k}} \Delta \tilde{E}_0^L}{\mathbf{F}_0 \cdot \Delta \tilde{E}_0^T}. \quad (14)$$

In eq 14,

$$\mathbf{F}_0 = \frac{S^T(0) - S^L(0)}{4\pi g_K} \int_{\Omega} \Delta \mathbf{E}_0(\mathbf{r}) \cdot \mathbf{D}_r \frac{d\mathbf{r}}{r^3}. \quad (15)$$

and  $\mathbf{D}_r = 3\hat{\mathbf{r}}\hat{\mathbf{r}} - \mathbf{1}$ .

`field_lt(n2,4)` is the transverse field

$$|\Delta E_0^T|^2 = \langle k^2 |\Delta E_0|^2 \rangle_{\Omega_k} - \langle |\Delta \tilde{\mathbf{E}}_0 \cdot \mathbf{k}|^2 \rangle_{\Omega_k} \quad (16)$$

## 8.1 Charged solutes and Chebyshev approximation

Charged solutes constitute a significant problem since the longitudinal field does not go to zero at  $k = 0$ . The  $k = 0$  limit for  $\langle |\tilde{\mathbf{E}}_0 \cdot \mathbf{k}|^2 \rangle_{\Omega_k}$

$$(4\pi Q)^2 \quad (17)$$

where  $Q$  is the total charge, `ChargeTotal` in the code. This value is assigned to the longitudinal component of  $|\tilde{\mathbf{E}}_0|^2$  at  $k = 0$ .

The other problem appears for large solutes. The field decays very quickly and the step `kstep` is too large for a proper integration. For this, the results of angular integration from the subroutine `angular` are approximated by Chebyshev polynomials and then integrated. The results of such approximation for the elements of `field_lt` array are shown in Fig. 1.

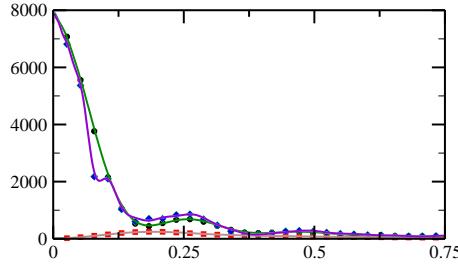


Figure 1: Comparison of the Chebyshev appr. and the actual calculations. Plastocianin with 128 points grid and factor = 9.

Since the step `kstep` is fairly large, the errors of calculations are reproduced in the Chebyshev fit. The main part of the maximum  $\langle |\tilde{\mathbf{E}}_0 \cdot \mathbf{k}|^2 \rangle_{\Omega_k}$  comes, however, from the trivial field of the total charge

$$\langle |\tilde{\mathbf{E}}_0 \cdot \mathbf{k}|^2 \rangle_{\Omega_k} = (4\pi j_0(kR_0))^2, \quad (18)$$

where  $R_0$  is the effective radius of the solute. Therefore, this function with the `radius` from the `geometry` subroutine is subtracted from the longitudinal part of `field_lt` (`field_lt(:,1,3,4)`) before Chebyshev approximation is done. This reduces the part which is actually Chebyshev approximated to about 25 % of the total numerical value (Fig. 2).

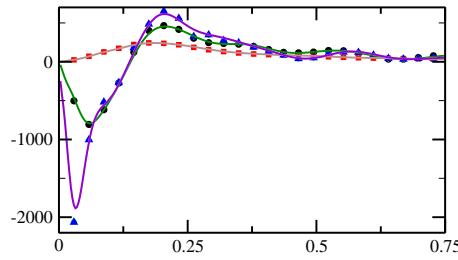


Figure 2: Comparison of the Chebyshev appr. to the actual calculations. Plastocianin with 128 points grid and factor = 9.

After the Chebyshev approximation is done, each `kstep` is divided by the `scaling` factor in order to integrate with a finer step. The one-dimensional integration is then performed with longitudinal and transverse structure factors. The output gives the free energy of solvation of the given charge distribution by the total solvent polarization (nuclear+electronic) and the nuclear component only. These are given as the solvation Gibbs energy and the solvent reorganization energy in the output.

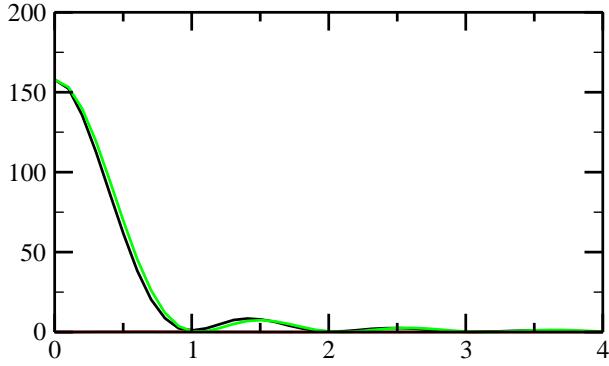


Figure 3: Comparison of the numerical and analytical results for the spherical ion.

## 9 Code Testing

Testing is performed on two limiting cases: (1) a charge in the center of a spherical solute with the Fourier transform of the field

$$\Delta E(\mathbf{k}) = -\frac{4\pi z i}{k^2} \mathbf{k} j_0(ka), \quad (19)$$

where  $a$  is the radius of the sphere and (2) dipole inside a spherical solute with the field

$$\Delta E(\mathbf{k}) = 4\pi m_0 \left[ 3\hat{\mathbf{k}}(\hat{\mathbf{k}} \cdot \hat{\mathbf{s}})\hat{\mathbf{s}} \right] \frac{j_1(ka)}{ka}. \quad (20)$$

### 9.1 Ion test

The input file contains the name of the solute file, solvent file, charge file, temporary file, output file, number of solvent molecules, number of charges, and the reorganization calculation parameter which can be either 0 or 1: 0 is no density reorganization, 1 is the density reorganization added to the orientational reorganization.

input5.dat (dhs.dat specifies the solvent as a dipolar hard spheres liquid with  $(m^*)^2 = 4.0$ ):  
s5.dat:

c5.dat:

In Figure 9.1 the numerical calculation of  $k^2 E^L(k)$  is compared to the analytical result

$$k^2(E^L(k))^2 = 16\pi^2 j_0(kR_1)^2 \quad (21)$$

### 9.2 Dipole test

This test compares the field of a non-point dipole with the field of a point dipole.

input1.dat (dhs.dat specifies water as the solvent):

dipole.dat:

cDelta.dat:

cMean.dat:

Longitudinal field:

$$(\mathbf{k} \cdot \mathbf{E})^2 = \frac{64\pi^2}{3} m^2 \left( \frac{j_1(kR_1)}{kR_1} \right)^2 \quad (22)$$

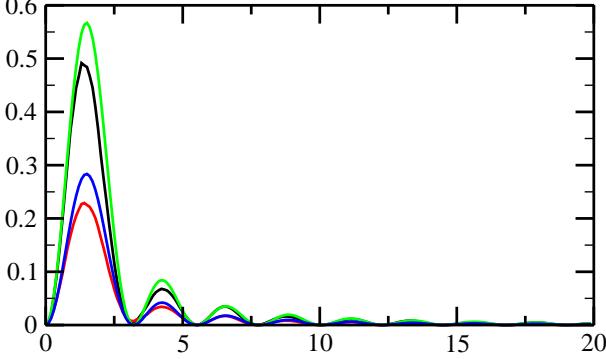


Figure 4: Comparison of the numerical and analytical results for the point dipole.

Transverse field (angular average):

$$\langle (E^T)^2 \rangle_\Omega = \frac{32\pi^2}{3} m^2 \left( \frac{j_1(kR_1)}{kR_1} \right)^2 \quad (23)$$

## 10 Subroutine deltaGdens

This routine calculates the density reorganization energy and the density component of solvation energy according to the equation

$$\lambda_D = \frac{3y}{8\pi} \int |\Delta E_0(\mathbf{r})|^2 h_{0s}(\mathbf{r}) \theta_V d\mathbf{r} \quad (24)$$

where  $h_{0s}$  is the solute-solvent pair correlation function (a similar equation is for  $\Delta G_d$ ). The main part of the code is the calculation of  $h_{0s}$ . Routine `direct_corr` gives the direct correlation  $c_{0s}$  function  $c_{0s}$  on a three-dimensional grid. Then  $c_{0s}$  is Fourier transformed and multiplied by the PY structure factor in the routine `py`. The back Fourier transform of this function gives  $h_{0s}$ . This latter is then integrated with  $|\Delta E_0(\mathbf{r})|^2$  of a direct-space grid.

### 10.1 Subroutine DIRECT-CORR

This routine assigns  $c_{0s}$  a negative constant value inside the solute core and puts it equal to zero outside the core.

### 10.2 Subroutine cPY

Here,  $c_{0s}$  is approximated as a sum of PY direct correlation functions defined on each sphere making the solute core

$$c_{0s}(\mathbf{r}) = \sum_j c_{0s}^{PY}(\mathbf{r} - \mathbf{r}_j) \quad (25)$$

## 11 Subroutine SPHERE

The direct Fourier transform of the electric field created by a charge inside the solute is not converging (numerically). The actual calculations are performed on the field in the space between the solute

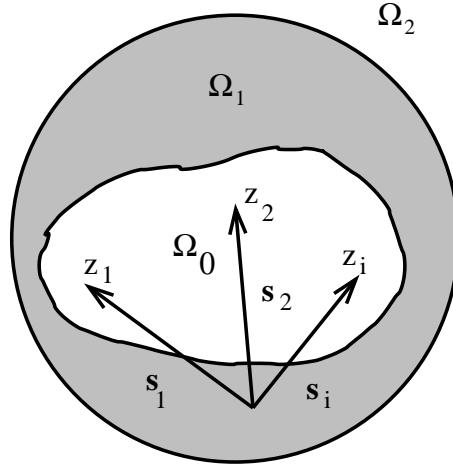


Figure 5: Donor-acceptor complex occupying the region of space  $\Omega_0$  with charges  $z_i$  located at  $\mathbf{s}_i$ . The shaded area is  $\Omega_1$ , and the area outside the sphere encircling the DAC is  $\Omega_2$ .

and the sphere of the size `size0` encompassing the solute. The subroutine SPHERE calculates analytically the Fourier transform of a given set of charges outside the sphere of radius `size0`.

To avoid numerical divergence of the Fourier integral in Eq. (3), the solvent is represented by a sum of Fourier transforms over two volumes  $\Omega_1$  and  $\Omega_2$  which together make the solvent volume  $\Omega$  (Fig. 5). The total difference field is then a sum of Fourier transforms from each region

$$\Delta\tilde{\mathbf{E}}_0 = \Delta\tilde{\mathbf{E}}_0(\Omega_1) + \Delta\tilde{\mathbf{E}}_0(\Omega_2). \quad (26)$$

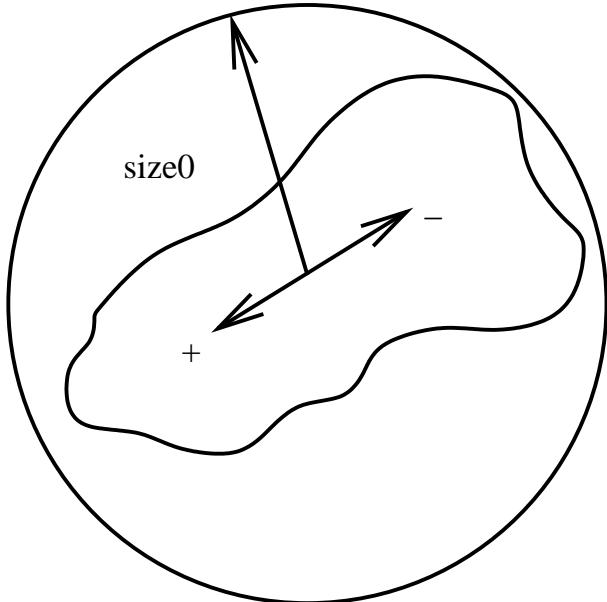
The first summand in Eq. (26) is calculated numerically by the FFT<sup>6</sup> and the second summand is calculated analytically for an arbitrary distribution of charges  $z_i$  with the coordinates  $\mathbf{s}_i$ ,  $s_i < L$  (Fig. 5):

$$\begin{aligned} \Delta\tilde{\mathbf{E}}_0(\Omega_2) = -4\pi e \sum_i z_i \sum_{n=1}^{\infty} (-i)^n \left(\frac{s_i}{L}\right)^{n-1} \frac{j_{n-1}(kL)}{k} \\ \left[ \hat{\mathbf{s}}_i P'_{n-1}(\cos \theta_i) - \hat{\mathbf{k}} P'_n(\cos \theta_i) \right]. \end{aligned} \quad (27)$$

Here,  $\cos \theta_i = \hat{\mathbf{s}}_i \cdot \hat{\mathbf{k}}$  and  $L$  is the radius of the sphere enclosing the volume  $\Omega_1$  (Fig. 5),  $j_n(x)$  is the spherical Bessel function, and  $P_n(\cos \theta_i)$  is the Legendre polynomial.

## 12 Subroutine FIELD

This routine calculates the (direct-space) vector of the field outside the solute and inside the sphere of radius `size0`. The field is put equal to zero for points falling outside the sphere of radius `size0`.



## 13 Configuration and Installation

This section details the operations needed to install the SolvMol code. First, the section details how to install the libraries, and then shows how to compile and use the code itself.

### 13.1 Fast Fourier Transform Library

The code uses the numerical Fourier transform routines provided by the *Fastest Fourier Transform in the West* (FFTW), version 3.3.5, which can be downloaded from the FFTW website at <http://www.fftw.org/download.html>. At the time of this manual update, this is the most stable version of the FFTW libraries with Fortran support. The FFTW library can be compiled for most linux operating system.

The following five steps describe how to install FFTW library:

**Step 1** Download a new version of fftw-3.3.5.tar from  
<http://www.fftw.org/download.html>

**Step 2** Untar the fftw-3.3.5

```
>>> tar -zxvf fftw-3.3.5.tar.gz
```

**Step 3** Create a fftw directory in your home directory

```
>>> mkdir ~/fftw
```

**Step 4** Change directory into fftw-3.3.5 and run the configuration command. On one of our local workstations, we were able to build FFTW 3.3.5 with on 24x Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz, 32GB memory, Ubuntu 14.04.4 LTS using the following configuration:

```
>>> ./configure CC=gcc CFLAGS="" --prefix=<ABSOLUTE-PATH-TO-HOME>  
/fftw --enable-float --enable-shared --disable-fast-install
```

Here you may have to give the absolute path to your home directory on your machine and place it in the option after `-prefix`.

**Step 5** Make the installation.

```
>>> make install
```

Once installed, the link to `/etc/ld.so.conf` should be updated with the path to the FFTW 3.3.5 library, and then `ldconfig` should be run (as root) to update the database. The library itself is referenced in the linking stage of compilation with the variable `OPT2` defined in the `makefile` in the source code directory.

The main options one needs are `-enable-float`, which provides access to the real-to-complex transforms, and `-enable-shared` for dynamic library linking. For this particular test installation, after '`make install`' has been executed, all `fftw` libraries are located in `/usr/local/fftw-2/lib`, which must be linked to `/etc/ld.so.conf` as previously described. Due to the non-standard library location, an additional option `-L/usr/local/fftw-2/lib -I/usr/local/fftw-2/include` is given in the `makefile`. It may be modified.

In this code, all Fourier transforms except for those for the solvation density component are calculated as the FFTW manual describes as *in place*, or rather the output array of every transform is given in the same array as the input. The density code however only handles a small amount of storage for the direct correlation function, in contrast to `fourierDeltaG.f`, which calculates the Fourier transform on a total of  $6 \times 3$ -dimensional arrays.

The options `FFTW_MEASURE + FFTW_IN_PLACE` are employed in the Fourier transform of the electric field to make the most highly optimized multiple FFT's (with `FFTW_MEASURE`), but require less memory (due to `FFTW_IN_PLACE`) at the slight sacrifice of performance. The density solvation code does not use this method for the calculation of the Fourier transform.

In the `fourierDeltaG.f` subroutine, the 3D real to complex transforms are calculated in the following manner:

```
call rfftwnd_f77_one_real_to_complex(plan, field_array, 0)
```

Due to the *in place* option, the code does not pass a meaningful last argument, but instead passes "0". The reason is because the FFTW library ignores this parameter anyway for the *in place* transforms (see page 32 of the FFTW 3.3.5 manual for more information). One should also note the size of the field arrays used in the `fourierDeltaG.f` subroutine requires padding in the 1<sup>st</sup>-dimension equal to  $N+2$ , not just  $N$  elements of data (see page 32, in section 3.5.3 of the FFTW 3.3.5 manual).

The following five steps are installation instructions for SolvMol package:

**Step 1** Download the SolvMol (Gfortran compiler) from

<http://theochemlab.asu.edu/>

**Step 2** Untar the archive.

```
>>> tar -xvf SolvMolCR.tgz
```

**Step 3** Change directory into the SolvMolCR/code directory and clean the original installation.

```
>>> make clean
```

**Step 4** Modify the make file to include the referenced the newly created library links.

```
FFTWLIB = $<ABSOLUTE-PATH-TO-HOME>$/fftw/  
FFTWINC = $<ABSOLUTE-PATH-TO-HOME>$/fftw/
```

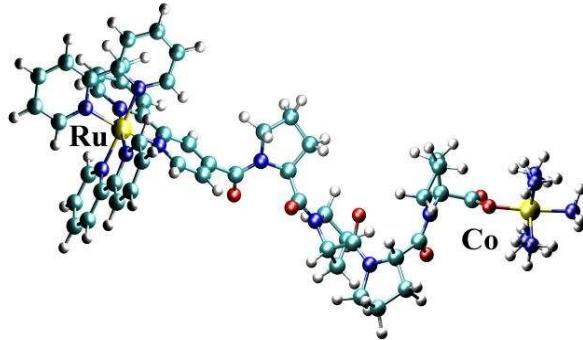


Figure 6: Donor-acceptor complex studied in Ref. 5

**Step 5** Make the installation.

```
>>> make
```

## 13.2 Supported Fortran Compilers

A number of Fortran compilers have been used to run the code, most notably PGI and `gfortran`. This version of the code assumes the use of `gfortran`. It can be compiled on Linux and Mac machines.

# 14 Running

Three examples are given below: (1) calculations of the reorganization energy of polypeptide-linked donor-acceptor complex studied in Ref. 5 (see Fig. 6), (2) calculations of the solvation Gibbs energy and ET reorganization energy of plastocyanin metalloprotein, and (3) reorganization energy of charge transition in *p*-nitroaniline studied in Ref. 1.

## 14.1 Polypeptide

Input file:

```
peptide.dat
h2o.dat
298.out
r298.dat
0
1
298.
298.
5.
298.
0.
0.
0.
0.
```

Solute file (peptide.dat):

```
s.dat
delta_charge.dat
delta_charge.dat
44
44
143
Y
```

#### 14.1.1 Output File

File out.

## 14.2 Plastocyanin

Input file input280.dat:

```
plastocya.dat (solute file)
tip3p.dat      (file containing the names of the solvents used)
280.out        (output)
redoxG280.dat (output)
0              (flag to tell whether to use a PDB input)
1              (number of solvents)
280.           (initial temperature)
280.           (final temperature)
5.             (temperature step)
298.           (reference temperature for the data in the database)
1.             (parameter determinig whether to load a structure factor)
0.             (parameter determinig whether to load a structure
factor)
0.             (switching the Stokes shift calculations)
0.             (looking for dielectric constant derivatives from the datadbase if >0.)
4005           ( number of points in the structure factor file)
280.slt.25ns.dat (name of the file)
```

The code assumes a possibility that the structure factor from simulations can be loaded for the  $k$ -integration for either the total polarization response or for the nuclear component. If this parameter is  $> 0$ , the file is loaded.

Solute file:

```
solute.dat
delta_charge.dat
mean_charge.dat
4              (number of delta charges)
1434          (number of mean charges)
1434          (number of atoms)
N              (specifies whether to make DelPhi input files)
```

`solute.dat` includes  $\{x, y, z, r\}$  data for the solute; `delta_charge.dat` includes difference charges; `mean_charge.dat` includes mean charges.

```
open(unit=in,file=input_file)
  read(in,'(a)') solute_xyz ! solute configuration in XYZ
  read(in,'(a)') chargeDelta_file ! coordinates of difference charges
  read(in,'(a)') chargeMean_file ! coordinates of mean charges
  read(in,*) n_delta_charge ! number of difference charges
  read(in,*) n_mean_charge ! number of mean charges
  read(in,*) n_sphere
  read(in,'(a)') DelPhi
close(in)
```

#### 14.2.1 Output file

File PCtest/out280. In this calculations, the structure factor from simulations is used in integration.

### 14.3 p-nitroaniline

In this example the reorganization energy of charge transfer in p-nitroaniline (PNA) is calculated at  $T = 242$  K. Input file:

```
pna.dat
tip3p.dat
pnitro300.dat
lambdas300.dat
0
1
300.
300.
5.
300.
0.
0.
0.
0.
```

Solute file:

```
coordinates.dat
charges.dat
charges.dat
16
16
16
N
```

Coordinates of atoms (file `coordinates.dat`):

5.201991	3.692969	7.447498	1.235
5.420828	5.489859	6.190092	1.235
10.160594	1.971182	3.917766	1.625
5.764136	4.346843	6.54808	1.515
9.03879	2.533609	4.530267	1.775
8.411448	1.882589	5.607069	1.775
7.327806	2.461856	6.260981	1.775
6.908694	3.733919	5.871161	1.775
7.514418	4.410958	4.812671	1.775
8.596379	3.818169	4.168174	1.775
8.752819	0.894826	5.91263	1.21
6.834008	1.954221	7.083371	1.21
7.164255	5.39737	4.525762	1.21
9.081726	4.343778	3.34734	1.21
10.219709	0.958982	3.974255	0.
10.346197	2.299372	2.974903	0.

Difference charges between the ground and charge-transfer states (file `charges.dat`):

5.201991	3.692969	7.447498	-0.045694000000000001
5.420828	5.489859	6.190092	-0.056790000000000001
10.160594	1.971182	3.917766	0.081875000000000003
5.764136	4.346843	6.54808	-0.129127
9.03879	2.533609	4.530267	-0.096674000000000001
8.411448	1.882589	5.607069	0.152165
7.327806	2.461856	6.260981	-0.16203499999999998
6.908694	3.733919	5.871161	0.305430000000000003
7.514418	4.410958	4.812671	-0.15935499999999997
8.596379	3.818169	4.168174	0.149919
8.752819	0.894826	5.91263	-0.012342999999999993
6.834008	1.954221	7.083371	-0.003664000000000006
7.164255	5.39737	4.525762	-0.003477000000000008
9.081726	4.343778	3.34734	-0.012197000000000013
10.219709	0.958982	3.974255	-0.003992999999999969
10.346197	2.299372	2.974903	-0.0040400000000000436

### 14.3.1 Output file

File `pna/out242`

## 14.4 Quinoxaline

This is an example of calculations of the Stokes shift dynamics of quinoxaline in supercooled MTHF as implemented in Ref. 4 (subdirectory “mthf”). Input file:

`quinox.dat`  
`mthf.dat`  
`quinox92.out`

```
lambda92.dat
0
1
92.
92.
5.
92.
0.
0.
1.
0.
```

Solute input:

```
radii_qu.dat
charges_qu.dat
charges_qu.dat
16
16
16
N
```

The atomic coordinates were calculated by DFT/BLY3P, the radii are from OPLS parametrization. The atomic charges are differences in excited triplet and ground singlet states from DFT/BLY3P. The Stokes shift correlation function  $F(s)$  is written to `Stokes.dat.92` where “92” stands for temperature at which the calculation was done.

## References

- [1] GHORAI, P. K., AND MATYUSHOV, D. V. Solvent reorganization entropy of electron transfer in polar solvents. *J. Phys. Chem. A* **110** (2006), 8857–8863.
- [2] MATYUSHOV, D. V. Reorganization energy of electron transfer in polar liquids: Dependence on the reactant size, temperature, and pressure. *Chem. Phys.* **174** (1993), 199–218.
- [3] MATYUSHOV, D. V. Solvent reorganization energy of electron transfer in polar solvents. *J. Chem. Phys.* **120** (2004), 7532–7556.
- [4] MATYUSHOV, D. V. On the microscopic theory of polar solvation dynamics. *J. Chem. Phys.* **122** (2005), 044502.
- [5] MILISCHUK, A. A., MATYUSHOV, D. V., AND NEWTON, M. D. Activation entropy of electron transfer reactions. *Chem. Phys.* **324** (2006), 172–194.
- [6] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical recipes in Fortran 77: The art of scientific computing*. Cambridge University Press, Cambridge, 1996.

# A Code Listing

## A.1 Main code

```
module fields
integer :: n_grid, nc, n_delta_charge, n_mean_charge, n_k
integer :: n_sphere, n2, n_appr
parameter (nc = 8, n_grid = 2**nc, n_k = 200)
parameter (n2 = n_grid/2, n_appr = 4)
real,pointer,dimension(:,:) :: delta_charge, mean_charge
    ! charge distributions in the initial and final states
real,pointer, dimension(:,:,:) :: delta_field_x,
:                               delta_field_y, delta_field_z
    ! difference of electric fields of the solute in the final and
    ! initial states
real, pointer, dimension(:,:,:,:) :: mean_field_x, mean_field_y,
:                               mean_field_z
    ! mean electric field in the initial and final redox states
real,pointer, dimension(:,:) :: solute, solute0 ! file of atomic coordinates and ra
real, dimension(3) :: deltaE0, deltaE1
real,dimension(3) :: CenterCharge
real :: size0, radius, TotalDeltaCharge, TotalMeanCharge
real :: x_center, y_center, z_center
real :: lambda_d0, g_contact, deltaGd0, deltaGind
real,pointer,dimension(:,:) :: scDelta, scMean
real,pointer,dimension(:) :: siDelta, siMean
real,dimension(0:n2,n_appr) :: field_lt
end module fields

module solv_parameters
integer :: n_dielectric
real :: mu, eta, ei, es, sigma, y0, s0L, s0T, tempr, alpha
real :: yeff, yinf, yquad, alphap, T_initial, T_final, T_step
real :: yeoffn, sNL, sNT, gNK, gK, eidT, esdT, p_param, T_ref
real,allocatable,dimension(:) :: beta, tau, epsilon_omega
end module solv_parameters

program dGibbs_solvent
use fields
use solv_parameters
!#####
! this program calculates the Gibbs solvation energy in a dipolar
! polarizable liquid
! n_sphere    number of spheres representing the solute
! n_solvent   number of solvents considered
! n_param     number of solvent parameters
```

```

!+++++
      implicit none
! integer variables
  integer :: n_solvent, i_delta_charge, inputFlag
  integer :: i,j, k, i_temp, i_out, i_solv, isolv
  integer :: nn, i_in, ii, i_deltaG, n_temperature, n_cheb, n1
  integer :: unitScrn
  parameter( i_deltaG=27, i_out = 31, i_temp = 33, i_in =34)
  parameter( unitScrn = 6, i_solv=39 )
! character variables
  character(len=20) :: out_file, temp_file, deltaG_file,solvent_file
  character(len=30) :: input_file
  character(len=15) :: namesolv
! real variables
  real :: dr, es0, ei0, etaT
  real :: pi, factor, x, y, z, deltaG0, constant, deltaGL, deltaGT
  real :: deltaGd, timearray(2), delapse, dtime
  real :: structureFparam1, structureFparam2, dynamicsParam
  real :: dielectricT, lambdaD, lambdaL, lambdaT, lambda
  real,pointer,dimension(:) :: chebL, chebT, chebLd, chebTd
  real, external :: funl, funt
  parameter(factor = 9., constant = 14.3995)

! derived types

  type physprop
    character(len=15) :: name
    real :: sigma
    real :: eta
    real :: mu
    real :: alpha
    real :: es
    real :: ei
    real :: alphap
  end type physprop
  type(physprop), pointer, dimension(:) :: solvents

  type dielprop
    character(len=15) :: name
    real :: einfT
    real :: esT
  end type dielprop
  type(dielprop), pointer, dimension(:) :: epsilonT

! reading in the name of the external file with input information

```

```

read(*,'(a)') input_file
read(*,'(a)') solvent_file      ! solvent parameters
read(*,'(a)') out_file          ! output file
read(*,'(a)') deltaG_file       ! results of the calculations
read(*,*) inputFlag            ! flag to tell whether to use a PDB input file
                                ! inputFlag = 0 take input from files
                                ! inputFlag = 1 take input from PBD
                                ! inputFlag = 2 take L and T components of
                                !               energy from a previous run
read(*,*) n_solvent             ! number of solvents
read(*,*) T_initial              ! initial temperature
read(*,*) T_final                ! final temperature
read(*,*) T_step                 ! temperature step
read(*,*) T_ref                  ! reference temperature for the data
                                ! listed in the database
read(*,*) structureFparam1      ! determines whether to use the PPSF or to
                                ! use the structure factors from simulations
! ***** One needs to put the name of the file with the structure factors if
! ***** structureFparam > 0
read(*,*) structureFparam2
read(*,*) dynamicsParam          ! if > 0, the call of the Stokes shift routine
                                ! is executed at each temperature, gives the output
                                ! of the Laplace transform of the Stokes shift
                                ! correlation function
                                ! if > 0, looks for the dielectric constant derivatives
                                ! in the calculations of the solvation entropy.
                                ! Derivatives are available for fewer solvents than
                                ! other properties

read(*,*) dielectricT

! execution time

delapse = dtime(timearray)

! parameters

pi = 4.*atan(1.)
n_temperature = int(abs(T_initial - T_final)/T_step) + 1
nn = n_grid
n1 = nn - 1

! Chebyshev arrays are used to fit the longitudinal and transverse
! projections of the fields for the calculation of the k-integral
! with the corresponding structure factors

n_cheb = min0(n_grid/2 - 2, 40)
allocate( chebL(1:n_cheb), chebT(1:n_cheb))
allocate( chebLd(1:n_cheb), chebTd(1:n_cheb) )

```

```

! input of solute structural information

allocate(solvents(n_solvent), epsilonT(n_solvent) )

if(inputFlag== 0) then
  call ReadFile(input_file)
else
endif

! Saving the initial definition of the solute

solute0 = solute

! Parameters output to standard output and disk file

open(i_out,     file = out_file)
open(i_deltaG, file = deltaG_file, status='unknown')

! Write out all copyright info

write(*,901) n_solvent
write(*,903) n_sphere
write(*,907) n_mean_charge, TotalMeanCharge - TotalDeltaCharge/2.
write(*,911) n_delta_charge, TotalMeanCharge + TotalDeltaCharge/2.
write(*,912) TotalMeanCharge, TotalDeltaCharge
write(*,909) CenterCharge
write(*,902) solvent_file
write(*,910)

write(i_out,901) n_solvent
write(i_out,903) n_sphere
write(i_out,907) n_mean_charge,
:           TotalMeanCharge - TotalDeltaCharge/2.
write(i_out,911) n_delta_charge,
:           TotalMeanCharge + TotalDeltaCharge/2.
write(i_out,909) CenterCharge
write(i_out,902) solvent_file
write(i_out,910)

! Reading in solvent parameters from the database

call input_solvent(solvent_file, n_solvent, solvents)

if(dielectricT > 0.) then
  call dielectric(solvent_file, n_solvent, epsilonT )
else

```

```

    esdT = 0.
    eidT = 0.
endif

! Calculation of the total and nuclear solvation free energy for given solvent
! parameters

! writing solvent properties in the "solvent.dat" file

open(unit=i_solv,file='solvent.dat')

solvent: do isolv = 1, n_solvent
    sigma      = solvents(isolv)%sigma
    eta        = solvents(isolv)%eta
    mu         = solvents(isolv)%mu
    namesolv   = solvents(isolv)%name
    es0        = solvents(isolv)%es
    ei0        = solvents(isolv)%ei
    alpha       = solvents(isolv)%alpha
    alphap     = solvents(isolv)%alphap

! temperature derivatives of the dielectric constants

if(dielectricT > 0.) then
    eidT      = epsilonT(isolv)%einfT
    esdT      = epsilonT(isolv)%esT
endif

write(*,917)
write(*,918) isolv, namesolv, sigma, mu, es0, ei0, alpha, eta
write(*,910)
write(i_out,918) isolv, namesolv, sigma, mu, es, ei, alpha, eta
write(i_out,910)

! add radius of the solvent molecule to the radius of each solute sphere

solute(:,4) = solute0(:,4) + sigma/2.

call geometry(1.)

! starting calculation of the temperature variation of Delta_G

temperature: do ii = 1, n_temperature

    tempr = T_initial + (real(ii) - 1.) * T_step

    call scfy(tempr, T_ref, mu, 0., sigma, 0., eta, alphap,

```

```

        :           alpha, sqrt(ei0), eidT, yeff, yinf, yquad)

! temperature variation of the high-frequency and static dielectric constants

es = es0 + esdT *(tempr - T_ref)
ei = ei0 + eidT *(tempr - T_ref)

write(*,990)
write(*,919) tempr, es, ei

! k=0 structure factors of full (nuclear+induced) and nuclear polarization

s0L   = (1. - 1./es)/3./yeff
s0T   = (es - 1.)/3./yeff
gK    = (s0L + 2.*s0T)/3.

! nuclear component of the dielectric constants

yeffn = yeff - yinf
sNL   = (1./ei - 1./es)/3./yeffn
! gNK  = (es - 1.)*(2.*es + 1.)/9./es/yeff
! sNT  = (3. * gNK - sNL)/2.
sNT  = (es - ei)/3./yeffn
gNK  = (sNL + 2.*sNT)/3.

! etaT takes care of temperature expansion

etaT = eta - alphap*eta*(tempr - T_ref)

! calculation of the density fluctuation component of the
! Gibbs energy, see eq 128 in JCP 120 (2004) 7532

write(i_solv,*) tempr, yeff, yinf, eta, es, ei

write(*,989)
write(*,980) etaT

call deltaGdens(etaT, dr)

if(ii == 1 .and. inputFlag < 2) then

allocate(delta_field_x(0:n1+2,0:n1,0:n1),
:           delta_field_y(0:n1+2,0:n1,0:n1),
:           delta_field_z(0:n1+2,0:n1,0:n1) )
delta_field_x = 0.
delta_field_y = 0.
delta_field_z = 0.

```

```

allocate(mean_field_x(0:n1+2,0:n1,0:n1),
:           mean_field_y(0:n1+2,0:n1,0:n1),
:           mean_field_z(0:n1+2,0:n1,0:n1)  )
mean_field_x = 0.
mean_field_y = 0.
mean_field_z = 0.

c arrays used in the calculation of the field outside the cut-off sphere

allocate(scDelta(n_delta_charge,3),siDelta(n_delta_charge),
:           scMean(n_mean_charge,3), siMean(n_mean_charge) )

do i = 1, n_delta_charge
    scDelta(i,1) = delta_charge(i,1) - x_center
    scDelta(i,2) = delta_charge(i,2) - y_center
    scDelta(i,3) = delta_charge(i,3) - z_center
    siDelta(i)   = sqrt(scDelta(i,1)*scDelta(i,1) +
:                           scDelta(i,2)*scDelta(i,2) +
:                           scDelta(i,3)*scDelta(i,3))
enddo

do i = 1, n_mean_charge
    scMean(i,1) = mean_charge(i,1) - x_center
    scMean(i,2) = mean_charge(i,2) - y_center
    scMean(i,3) = mean_charge(i,3) - z_center
    siMean(i)   = sqrt(scMean(i,1)*scMean(i,1) +
:                           scMean(i,2)*scMean(i,2) +
:                           scMean(i,3)*scMean(i,3)  )
enddo

call fourier(dr, n_cheb, chebL, chebT, chebLd, chebTd )

deallocate(delta_field_x, delta_field_y, delta_field_z)
deallocate(mean_field_x, mean_field_y, mean_field_z)
deallocate(scDelta, siDelta, scMean, siMean)

endif

call free_energy

! Output

write(*,910)
write(*,915) tempr
write(*,920) deltaG0, deltaGd
write(*,922) deltaGL, deltaGT
write(*,990)

```

```

write(*,921) lambda, lambdad

write(i_out,910)
write(i_out,915) tempr
write(i_out,920) deltaG0, deltaGd
write(i_out,922) deltaGL, deltaGT
write(i_out,990)
write(i_out,921) lambda, lambdad
write(i_deltaG,992) tempr, deltaG0, deltaGd, deltaGind,
:           lambda, lambdad

! executing the call of the Stokes shift correlation function
! files are written into Stokes.dat.tempr

if(dynamicsParam > 0. .and. ii == 1) then
  call dynamics(namesolv, dr, n_cheb, chebLd, chebTd)
endif

enddo temperature
enddo solvent
close(i_solv)

deallocate(solute, solvents, solute0, mean_charge, delta_charge)
deallocate( chebL, chebT, chebLd, chebTd )

! execution time

delapse = dtime(timearray)
write(*,*)"***** T = ", tempr,' execution time = ',delapse
write(*,990)
write(*,981)
write(*,990)

close(i_out)
close(i_deltaG)

!      stop ' ===== Program is terminated by normal STOP ====='

901 format(70('*')/5X,'Starting calculation of the',
: ' Gibbs solvation energy for ',i3,' solvents ')
902 format(5X,'Solvent file    ',a20)
903 format(5X,'Solute is represented by ',i5,' fused spheres')
904 format(5X,'Solute file    ',a20,' Solute file 2 ',a20)
900 format(5X,'Solute xyz-file ',a20)
907 format(5x,'Number of mean charges ',i5,' Total Charge 1 ',f10.4)
911 format(5x,'Number of diff. charges ',i5,' Total Charge 2 ',f10.4)

```

```

912 format(5x,'Total mean charge      ',  

913      :      f10.4,' Total Diff.Ch. ',f10.4)  

915 format(5x,'Results of calculations at T = ',f10.4)  

909 format(5x,'Center of charge: x',f10.4,' y ',f10.4,' z ',f10.4)  

910 format(70('*'))  

917 format(5x,' Solvent properties at from the database: ')  

918 format(5X,i3,'   Name ',a11,' sigma ',f10.4,' mu ',f10.4/  

919 :      10X,' es ',f10.4,' ei ',f10.4,' alpha',f10.4' eta ',f10.4)  

919 format(5x,' Temperature loop  T = ',f10.4,' es = ',f10.4,  

920 :      ' ei = ', f10.4)  

920 format(5X,' Delta_G  = ',e14.6,' Delta_Gd  = ',e14.6)  

922 format(5x,' Delta_G^L = ',e14.6,' Delta_G^T = ',e14.6)  

921 format(5X,' lambda    = ',e14.6,' lambda_d  = ',e14.6)  

980 format(5x,' Starting calculation of the density comp., eta = ',  

981 :      f10.4)  

981 format(5x,'          (c) 2008 Arizona Board of Regents ')  

989 format(70('.'))  

990 format(70('---'))  

991 format(4(e14.5))  

992 format(f10.4,5(e13.5))

```

contains

```

subroutine free_energy
integer :: ix, nfin, npoints
real  :: kstep, a, b, intGL, intGT, scaling, k, fl, ft
real  :: xiL, xiT, empL, ssL, empT, ssT, qL, qT
real  :: chebev, j0x, z0, al, bl, at, bt, constant
real  :: aNl, bNl, aNt, bNt, xiLN, xiTN, qLN, qTN
real  :: intLamL, intLamT, fld, ftd
parameter (constant = 14.3995)
real,dimension(n_cheb) :: coeff, slCheb, stCheb, slNCheb, stNCheb
character*20:: structureF_file

```

```

kstep  = 2.*pi/float(nn)/dr

```

```

write(*,917)      tempr, yeff, yinf, s0L, s0T
write(*,918)      yeffn, sNL, sNT
write(*,919)      gK, gNK
write(i_out,917)  tempr, yeff, yinf, s0L, s0T
write(i_out,918)  yeffn, sNL, sNT
write(i_out,919)  gK, gNK

```

```

! reads in the total structure factors if structureFparam1 > 0

```

```

if(structureFparam1 > 0.) then
  read(*,*) npoints

```

```

        read(*,'(a)') structureF_file
        call inputStructureF(npoints, structureF_file, n_cheb,
:                           slCheb, stCheb, al, bl, at, bt)
        else

        call root(s0L, funl, xiL)
        empL = 0.95 ! empirical parameter rescaling the L-response
        ssL  = 2.    ! factor to be used in the Baxter Q-function
        call root(s0T, funt, xiT)
        empT = 1.      ! no need for rescaling for T-response
        ssT  = -1.
        write(*,930)
        write(*,931) xiL, xiT

        endif

! reads in the nuclear structure factors if structureFparam2 > 0

        if(structureFparam2 > 0.) then
            read(*,*) npoints
            read(*,'(a)') structureF_file
            call inputStructureF(npoints, structureF_file, n_cheb,
:                           slNCheb, stNCheb, aNl, bNl, aNt, bNt)
            else

            call root(sNL, funl, xiLN)
            empL = 0.95 ! empirical parameter rescaling the L-response
            ssL  = 2.    ! factor to be used in the Baxter Q-function
            call root(sNT, funt, xiTN)
            empT = 1.      ! no need for rescaling for T-response
            ssT  = -1.
            write(*,932) xiLN, xiTN

            endif

! [a, b] is the interval of Chebyshev approximation

        a = 0.
        b = n2*kstep

! Integration with Chebyshev approximated fields

        open(23,file='slt.dat')
        open(24,file='f.dat')

! The first point of the trapezoid integration comes with
! 0.5 coefficient, one does not need to worry about the end point

```

```

call qu(0., ssL*xiL, qL)
call qu(0., ssT*xiT, qT)
call qu(0., ssL*xiLN, qLN)
call qu(0., ssT*xiTN, qTN)
intGL    = 16.*pi**2*TotalDeltaCharge*TotalMeanCharge/2./qL ! L component of Gibbs
intGT    = 0.                                         ! T component of Gibbs
intLamL  = 16.*pi**2*TotalDeltaCharge*TotalDeltaCharge/2./qLN
intLamT  = 0.

scaling = 10.
nfin = (n2-1)*nint(scaling) - 1
do ix = 1, nfin
  k = ix * kstep/scaling

! total dielectric response

if(structureFparam1==0.) then
  call qu(empL*k*sigma, ssL*xiL, qL)
  call qu(empT*k*sigma, ssT*xiT, qT)
  qL = 1./qL
  qT = 1./qT
else
  qL = chebev(al,bl,slCheb,n_cheb,k)
  qT = chebev(at,bt,stCheb,n_cheb,k)
endif

! nuclear dielectric response

if(structureFparam2==0.) then
  call qu(empL*k*sigma, ssL*xiLN, qLN)
  call qu(empT*k*sigma, ssT*xiTN, qTN)
  qLN = 1./qLN
  qTN = 1./qTN
else
  qLN = chebev(aNl,bNl,slNCheb,n_cheb,k)
  qTN = chebev(aNt,bNt,stNCheb,n_cheb,k)
endif

! integral for Delta_G

z0 = (4.*pi*j0x(k*radius))**2*TotalMeanCharge*
      TotalDeltaCharge
:

f1 = chebev(a, b, chebL, n_cheb, k)

```

```

        intGL = intGL + (z0 + fl) *qL

        ft = chebev(a, b, chebT, n_cheb, k)
        intGT = intGT + ft*qT

! integral for lambda

        z0 = (4.*pi*j0x(k*radius))**2*TotalDeltaCharge*
        :
        TotalDeltaCharge

        fld = chebev(a, b, chebLd, n_cheb, k)
        intLamL = intLamL + (z0 + fld) *qLN

        ftd = chebev(a, b, chebTd, n_cheb, k)
        intLamT = intLamT + ftd*qTN

        write(24,933) k, fl, ft, fld, ftd
        write(23,933) k, qL, qT, qLN, qTN

enddo
close(23)
close(24)

deltaGL = -intGL *kstep/16./pi**3/scaling
deltaGT = -intGT *kstep/16./pi**3/scaling

deltaGL = deltaGL * 3.*yeff*constant
deltaGT = deltaGT * 3.*yeff*constant
deltaGd = -deltaGd0 * 3. * yeff * constant
deltaG0 = (deltaGL + deltaGT*sOL/gK + deltaGd + deltaGind)

lambdaL = intLamL *kstep/16./pi**3/scaling
lambdaL = lambdaL *3.*yeffn*constant
lambdaT = intLamT *kstep/16./pi**3/scaling
lambdaT = lambdaT *3.*yeffn*constant
lambda_dad = lambda_d0 * 3. * yeffn * constant
lambda = lambdaL + lambdaT*sNL/gNK + lambda_dad

write(*,934)
write(*,*) ' GL      ',deltaGL,      ' GT      ', deltaGT
write(*,*) ' Gd      ',deltaGd
write(*,*) ' lamL    ',lambdaL,      ' lamT    ', lambdaT
write(*,*) ' lamd    ',lambda_dad

917 format(5X,'T = ',f10.4,' yeff = ',f10.4,' yinf = ',f10.4,
:           ' SOL = ',f10.4,' SOT = ', f10.4)

```

```

918 format(5X, yeffn = ',f10.4,
      : , SNL = ',f10.4, ' SNT = ', f10.4)
919 format(5x, gK     = ',f10.4, gKN = ',f10.4)
930 format(5x, ' Polarity parameters of the MSA: ')
931 format(5X, xiL  = ', f10.4, xiT  = ',f10.4)
932 format(5X, xiLN = ', f10.4, xiTN = ',f10.4)
901 format(5X, 'Longitudinal', 3(f10.4,2X), 'Transverse ', 3(f10.4,2X))
933 format(f10.4,4e12.4)
934 format(70('.'))

end subroutine free_energy

! Geometric parameters relevant to the solute

subroutine geometry(callparam)
integer :: i_min
real :: epsilon, dx, dy, dz, r2, sum_r, callparam
real :: r2max, size, r, volume

epsilon = 5.e-4

! geometrical center of the molecule
x_center = 0.
y_center = 0.
z_center = 0.
do i = 1, n_sphere
    x_center = x_center + solute(i,1)
    y_center = y_center + solute(i,2)
    z_center = z_center + solute(i,3)
enddo

x_center = x_center/real(n_sphere)
y_center = y_center/real(n_sphere)
z_center = z_center/real(n_sphere)

10   do i = 1, n_mean_charge

        r2 = (mean_charge(i,1) - x_center)**2 +
        :       (mean_charge(i,2) - y_center)**2 +
        :       (mean_charge(i,3) - z_center)**2
        if(r2 == 0.) goto 20
    enddo
    goto 30
20   x_center=x_center + 0.01
    goto 10
! determining the maximum dimension of the solute

30   size = 0.

```

```

do i = 1, n_sphere

    r2 = (solute(i,1) - x_center)**2 +
    :      (solute(i,2) - y_center)**2 +
    :      (solute(i,3) - z_center)**2
    r = sqrt(r2) + solute(i,4)
    if(r.gt.size) size = r
enddo

! maximum extention of the siolute relative to the geometrical center
! is changed by adding sigma

size0 = size + sigma
size  = factor * size ! the size of the integration box is scaled
                      ! with the FACTOR

dr  = size/float(n_grid) ! increment in the coordinate space
write(*,912) n_grid, dr
write(*,910)

if(callparam > 0.) then

! calculation of the effective vdW radius of the molecule
    write(*,916)
    call volume1(epsilon, volume)
    write(*,919) volume, epsilon
    radius = (3./4./pi*volume)**0.3333333
    g_contact = 1.
else
! solute radius is defined by the maximum extesion
    radius = size0
    g_contact = 1.
endif

! if the center of the molecules falls into solvent,
! it should be shifted in order to avoid divergencies in the field
! calculation

100 r2max = size0 * size0
    do i = 1, n_sphere
        r2 = (solute(i,1) - x_center)**2 +
        :      (solute(i,2) - y_center)**2 +
        :      (solute(i,3) - z_center)**2
        if ( r2 < solute(i,4)**2) goto 200
    ! this defines the number of the sphere closest
    ! to the center of the molecule
    if( r2 < r2max) then

```

```

        i_min = i
        r2max = r2
    endif
enddo
x_center = (x_center + solute(i_min,1))/2.
y_center = (y_center + solute(i_min,2))/2.
z_center = (z_center + solute(i_min,3))/2.
goto 100
200 continue
write(*,907) radius, size0, size
write(*,911) x_center, y_center, z_center

907 format(5X,' Effective radius           =' ,f10.4/
:      5X,' Solute maximum extension   =' ,f10.4/
:      5X,' Real space box, side       =' ,f10.4)

911 format(5X,' Coordinates of the center of charge:',
:      ' X= ',f10.4,' Y= ',f10.4,' Z= ',f10.4)
912 format(5X,'Three dimentional lattice of',i4,' points'/
:      5X,'is built with the increment',e12.4)
910 format(70('*'))
916 format(5X,'Starting Monte Carlo calculation of the vacuum energy'/
:      70('`'))
919 format(5X,' Solute volume           =' ,f10.4/
:      5X,' Relative error          =' ,f10.4)
end subroutine geometry

end

subroutine inputStructureF(npoints, file_name, n_cheb,
:                           slCheb, stCheb, al, bl, at, bt )
implicit none
! ****
! This subroutine reads in the simulation file and performs its Chebyshev
! approximation
! ****
integer :: npoints, n_cheb, i
real    :: al, bl, at, bt
real,dimension(npoints,2) :: sl, st
character (len=20) :: file_name
real,dimension(n_cheb) :: slCheb, stCheb

! opening simulation file

write(*,920) file_name

```

```

open(43,file=trim(file_name))
do i = 1, npoints
    read(43,*) sl(i,1), sl(i,2), st(i,2)
    st(i,1) = sl(i,1)
enddo
al = sl(1,1)
bl = sl(npoints,1)
at = st(1,1)
bt = st(npoints,1)

write(*,910)
write(*,*) '**** chebyshev appr. for sl(k) ',npoints,' k-values'

call chebishev(npoints, n_cheb, sl, slCheb)

write(*,*) '**** chebyshev appr. for st(k) ',npoints,' k-values'

call chebishev(npoints, n_cheb, st, stCheb)
close(43)
return
910 format(5x,70('-'))
920 format(5x, ' Opening structure factor file: ',a20)
end subroutine inputStructureF

subroutine ReadFile(input_file)
! ****
! This routine reads data from hard drive files
! ****
use fields
use solv_parameters
implicit none
integer :: in, startTemp, endTemp, stepTemp, i_mean_charge, i
integer :: i_delta_charge
parameter( in = 34, i_mean_charge = 35, i_delta_charge = 36)
character(len=30) :: input_file, chargeDelta_file,
:                           chargeMean_file, solute_xyz

open(unit=in,file=input_file)
read(in,'(a)') solute_xyz ! solute configuration in XYZ
read(in,'(a)') chargeDelta_file ! coordinates of difference charges
read(in,'(a)') chargeMean_file ! coordinates of mean charges
read(in,*) n_delta_charge ! number of difference charges
read(in,*) n_mean_charge ! number of mean charges
read(in,*) n_sphere

```

```

    close(in)

TotalDeltaCharge = 0.
TotalMeanCharge = 0.
CenterCharge     = 0.

write(*,910)
write(*,*) 'allocating arrays : n_delta_charge = ', n_delta_charge
write(*,*) '                           n_mean_charge = ', n_mean_charge
write(*,*) '                           n_sphere      = ', n_sphere

allocate(solute(n_sphere,4),
:   solute0(n_sphere,4), delta_charge(n_delta_charge,4),
:   mean_charge(n_mean_charge,4)  )

call input_solute(solute_xyz, solute, n_sphere)

! the CHARGE array contains three coordinates of the charge and the charge value

open(i_delta_charge, file = chargeDelta_file)
do i = 1, n_delta_charge
read(i_delta_charge,*) delta_charge(i,1), delta_charge(i,2),
:                   delta_charge(i,3), delta_charge(i,4)
TotalDeltaCharge = TotalDeltaCharge + delta_charge(i,4)
enddo
close(i_delta_charge)

open(i_mean_charge, file = chargeMean_file)
do i = 1, n_mean_charge
read(i_mean_charge,*) mean_charge(i,1), mean_charge(i,2),
:                   mean_charge(i,3), mean_charge(i,4)
TotalMeanCharge = TotalMeanCharge + mean_charge(i,4)
CenterCharge(1) = CenterCharge(1) + mean_charge(i,1)
CenterCharge(2) = CenterCharge(2) + mean_charge(i,2)
CenterCharge(3) = CenterCharge(3) + mean_charge(i,3)
enddo

close(i_mean_charge)

CenterCharge = CenterCharge/real(n_mean_charge)

return
910 format(70('*'))
end subroutine ReadFile

```

## A.2 Electric Field Calculation

```
subroutine field(r0, dr)
use fields
! ****
! This routine calculates the three-dimensional vector
! of the difference solute field; each component defined on
! a three-dimensional lattice.
! ****
implicit none
integer :: i, j, m, i_center, l, n, jj
real :: fx, fy, fz, dr, dx, dy, dz, r, r2, rc, rc2
real :: x, y, z, sign, sx_delta, sy_delta, sz_delta
real :: sx_mean, sy_mean, sz_mean, mc4, dc4
real :: r0, dotP, ex, ey, ez, emx, emy, emz

deltaE0 = 0.
deltaE1 = 0.

do i = 0, n_grid - 1
    do j = 0, n_grid - 1
        do m = 0, n_grid - 1
            sign = (-1.)**(j + m + i)

! sign is introduced to take into account that the r-integration is performed
! in the range [-a,a] for each of the spatial variables

            x = dr*float(i - n_grid/2)
            y = dr*float(j - n_grid/2)
            z = dr*float(m - n_grid/2)

            sx_delta = 0.
            sy_delta = 0.
            sz_delta = 0.
            sx_mean = 0.
            sy_mean = 0.
            sz_mean = 0.
            ex = 0.
            ey = 0.
            ez = 0.
            emx = 0.
            emy = 0.
            emz = 0.

            do 10 l = 1, n_sphere
                dx = solute(l,1) - x_center - x
                dy = solute(l,2) - y_center - y
                dz = solute(l,3) - z_center - z
                r2 = dx*dx + dy*dy + dz*dz
                if (r2 .gt. rc*rc) then
                    dotP = 0.
                else
                    dotP = 1. / sqrt(r2)
                end if
                mc4 = 1. / (1. + rc*sqrt(r2))
                dc4 = 1. / (1. + rc*sqrt(r2))
                sx_delta = sx_delta + sign * (dotP * mc4 * dx)
                sy_delta = sy_delta + sign * (dotP * mc4 * dy)
                sz_delta = sz_delta + sign * (dotP * mc4 * dz)
                ex = ex + sign * (dotP * mc4 * dx)
                ey = ey + sign * (dotP * mc4 * dy)
                ez = ez + sign * (dotP * mc4 * dz)
                emx = emx + sign * (dotP * mc4 * dx)
                emy = emy + sign * (dotP * mc4 * dy)
                emz = emz + sign * (dotP * mc4 * dz)
            end do 10
            sx_mean = sx_delta / n_sphere
            sy_mean = sy_delta / n_sphere
            sz_mean = sz_delta / n_sphere
            ex = ex / n_sphere
            ey = ey / n_sphere
            ez = ez / n_sphere
            emx = emx / n_sphere
            emy = emy / n_sphere
            emz = emz / n_sphere
        end do m
    end do j
end do i
```

```

dz = solute(l,3) - z_center - z
r2 = dx*dx + dy*dy + dz*dz
if(r2 < solute(l,4)*solute(l,4) ) goto 11
10 continue

! outside the solute

rc2 = x*x + y*y + z*z
rc = sqrt(rc2)

! calculation of the Delta E field

do jj = 1, n_delta_charge

dc4 = delta_charge(jj,4)
dx = x - delta_charge(jj,1) + x_center
dy = y - delta_charge(jj,2) + y_center
dz = z - delta_charge(jj,3) + z_center
r2 = dx * dx + dy * dy + dz * dz
r = sqrt(r2)
fx = dc4*dx/r/r2
fy = dc4*dy/r/r2
fz = dc4*dz/r/r2
sx_delta = sx_delta + fx*sign
sy_delta = sy_delta + fy*sign
sz_delta = sz_delta + fz*sign
ex = ex + fx
ey = ey + fy
ez = ez + fz
enddo

dotP = ex * x + ey * y + ez * z

deltaE1(1) = deltaE1(1) + (3.*x*dotP/rc2 - ex)/rc2/rc
deltaE1(2) = deltaE1(2) + (3.*y*dotP/rc2 - ey)/rc2/rc
deltaE1(3) = deltaE1(3) + (3.*z*dotP/rc2 - ez)/rc2/rc

! calculation of the mean_E field

do jj = 1, n_mean_charge
mc4= mean_charge(jj,4)
dx = x - mean_charge(jj,1) + x_center
dy = y - mean_charge(jj,2) + y_center
dz = z - mean_charge(jj,3) + z_center
r2 = dx * dx + dy * dy + dz * dz
r = sqrt(r2)

```

```

fx = mc4*dx/r/r2
fy = mc4*dy/r/r2
fz = mc4*dz/r/r2
sx_mean = sx_mean + fx * sign
sy_mean = sy_mean + fy * sign
sz_mean = sz_mean + fz * sign
emx      = emx      + fx
emy      = emy      + fy
emz      = emz      + fz

enddo
dotP = emx * x + emy * y + emz * z

deltaE0(1) = deltaE0(1) + (3.*x*dotP/rc2 - emx)/rc2/rc
deltaE0(2) = deltaE0(2) + (3.*y*dotP/rc2 - emy)/rc2/rc
deltaE0(3) = deltaE0(3) + (3.*z*dotP/rc2 - emz)/rc2/rc

if(rc2 > r0 * r0) then
  sx_delta = 0.
  sy_delta = 0.
  sz_delta = 0.
  sx_mean = 0.
  sy_mean = 0.
  sz_mean = 0.
endif

goto 12

! inside the solute

11   r2 = x*x + y*y + z*z
    if(r2 > r0*r0 ) then
      do jj = 1, n_delta_charge
        dc4= delta_charge(jj,4)
        dx = x - delta_charge(jj,1) + x_center
        dy = y - delta_charge(jj,2) + y_center
        dz = z - delta_charge(jj,3) + z_center
        r2 = dx * dx + dy * dy + dz * dz
        r = sqrt(r2)
        fx = - dc4*dx/r/r2
        fy = - dc4*dy/r/r2
        fz = - dc4*dz/r/r2
        sx_delta = sx_delta + fx*sign
        sy_delta = sy_delta + fy*sign
        sz_delta = sz_delta + fz*sign
      enddo

```

```

do jj = 1, n_mean_charge
mc4=mean_charge(jj,4)
dx = x - mean_charge(jj,1) + x_center
dy = y - mean_charge(jj,2) + y_center
dz = z - mean_charge(jj,3) + z_center
r2 = dx * dx + dy * dy + dz * dz
r = sqrt(r2)
fx = - mc4*sign*dx/r/r2
fy = - mc4*sign*dy/r/r2
fz = - mc4*sign*dz/r/r2
sx_mean = sx_mean + fx
sy_mean = sy_mean + fy
sz_mean = sz_mean + fz
enddo

endif

12   delta_field_x(i,j,m) = sx_delta
      delta_field_y(i,j,m) = sy_delta
      delta_field_z(i,j,m) = sz_delta
      mean_field_x(i,j,m) = sx_mean
      mean_field_y(i,j,m) = sy_mean
      mean_field_z(i,j,m) = sz_mean

      enddo
    enddo
  enddo

return
end subroutine field

```

### A.3 Fourier Transform

```
subroutine fourier(dr, n_cheb, chebL, chebT, chebLd, chebTd )
use fields
use solv_parameters
!*****
! This routine calculates the longitudinal and transversal difference
! fields of the donor-acceptor complex to be used in reorganization energy
! calculations
! switch controls allocation and call of calcualtion of the field and taking
! its Fourier transform
!*****
implicit none
! integer variables
integer :: ix, nn, n1, i, n_cheb, j
integer,dimension(0:n_grid) :: number

integer FFTW_FORWARD,FFTW_BACKWARD
parameter (FFTW_FORWARD=-1,FFTW_BACKWARD=1)

integer FFTW_REAL_TO_COMPLEX,FFTW_COMPLEX_TO_REAL
parameter (FFTW_REAL_TO_COMPLEX=-1,FFTW_COMPLEX_TO_REAL=1)

integer FFTW_ESTIMATE,FFTW_MEASURE
parameter (FFTW_ESTIMATE=0,FFTW_MEASURE=1)

integer FFTW_OUT_OF_PLACE,FFTW_IN_PLACE,FFTW_USE_WISDOM
parameter (FFTW_OUT_OF_PLACE=0)
parameter (FFTW_IN_PLACE=8,FFTW_USE_WISDOM=16)

integer FFTW_THREADS
parameter (FFTW_THREADS=128)
integer*8 plan

! real variables

real,dimension(0:n2)      :: nlattice
real,dimension(1:n2, 3) :: zero_array
real,dimension(1:n2, 2) :: temp_array
real :: dr, dr3, pi, kstep, kmax, r0, j0x, k, j1x
real,dimension(1:n_cheb) :: coeff
real,dimension(1:n_cheb) :: chebL, chebT, chebLd, chebTd
parameter( pi = 3.14159)

! subroutine FIELD creates a three-dimensional vector of the difference field
! of the donor-acceptor complex.
! The field components are stored in the three-dimensional arrays
```

```

! delta_field_x, delta_field_y, delta_field_z.
! The mean field is placed to
! mean_field_{x,y,z}

! kstep is the step in corresponding to Fourier transform
! kmax gives the radius of the sphere in the k-space

nn = n_grid
n1 = nn - 1
dr3 = dr*dr*dr
kstep = 2.*pi/float(nn)/dr
kmax = kstep * n2
write(*,916) kstep, kmax

! definition of the radius of the cut-off sphere

r0 = size0

! Calculation of the field outside the donor-acceptor complex and inside the sphere

write(*,915)
call field(r0, dr)
write(*,920)

! This calculates the Fourier transform of the field
write(*,*) ''
write(*,*) '    Initiation of FFTW library ...'

call rfftw3d_f77_create_plan(plan,nn,nn,nn,
 :      FFTW_REAL_TO_COMPLEX, FFTW_MEASURE + FFTW_IN_PLACE)
call rfftwnd_f77_one_real_to_complex(plan, delta_field_x, 0)
write(*,*) 'FFT of delta_field_x on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, delta_field_y, 0)
write(*,*) 'FFT of delta_field_y on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, delta_field_z, 0)
write(*,*) 'FFT of delta_field_z on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_x, 0)
write(*,*) 'FFT of mean_field_x on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_y, 0)
write(*,*) 'FFT of mean_field_y on ',nn,'^3 lattice complete.'
call rfftwnd_f77_one_real_to_complex(plan, mean_field_z, 0)
write(*,*) 'FFT of mean_field_z on ',nn,'^3 lattice complete.'
call rfftwnd_f77_destroy_plan(plan)
write(*,*) 'PLAN DESTROYED, FFTW exited normally.'
write(*,910)

```

```

field_lt = 0.
nlattice = 0.

! angular calculates the energy density of longitudinal and
! transverse components of the field averaged over the directions
! of the wave-vector

call angular

! longitudinal   dE * E_m                      field_lt(:,1)
! transverse     dE * E_m                      field_lt(:,2)
! longitudinal   dE * dE                       field_lt(:,3)
! transverse     dE * dE                       field_lt(:,4)

temp_array = 0.
zero_array = 0.

open(22,file='fields.dat')
do ix = 1, n2-1
    k = real(ix)*kstep
    if(nlattice(ix).ne.0.) then
        field_lt(ix,:)=field_lt(ix,:)/nlattice(ix)
    else
        field_lt(ix,:)=0.
    endif

    temp_array(ix+1,1) = k
    zero_array(ix+1,1) = k
    zero_array(ix+1,2) = (4.*pi*j0x(k*radius))**2*
                           TotalMeanCharge*TotalDeltaCharge
    zero_array(ix+1,3) = (4.*pi*j0x(k*radius))**2*
                           TotalDeltaCharge*TotalDeltaCharge
    :
    if(ix.le.20) then
        write(*,918) k, (field_lt(ix,i), i=1,4)
    endif
    write(22,918) k, field_lt(ix,1)-zero_array(ix+1,2),
                  field_lt(ix,2)
    :
enddo
close(22)

! Zero-k component of the field

field_lt(0,1) = (4.*pi)**2*TotalDeltaCharge*TotalMeanCharge
zero_array(1,2) = (4.*pi)**2*TotalDeltaCharge*TotalMeanCharge
field_lt(0,3) = (4.*pi)**2*TotalDeltaCharge*TotalDeltaCharge
zero_array(1,3) = (4.*pi)**2*TotalDeltaCharge*TotalDeltaCharge
field_lt(0,2) = 0.

```

```

field_lt(0,4)    = 0.
temp_array(1,1)  = 0.

open(23,file='flt.dat')
do ix = 0, n2-1
  k = real(ix)*kstep
  write(23,918) k, field_lt(ix,1), field_lt(ix,2),
:                      field_lt(ix,3), field_lt(ix,4)
enddo
close(23)

! Chebyshev approximation for the field components
! c_cheb is the array of Chebyshev coefficients for each
! angular averaged field

write(*,910)
write(*,*) '**** CHEBISHEV APPROXIMATION FOR ',n_cheb,' POINTS'

temp_array(1:n2,2)=field_lt(0:n2-1,1) - zero_array(1:n2,2)
call chebishev(n2, n_cheb, temp_array, chebL)

temp_array(1:n2,2)=field_lt(0:n2-1,2)
call chebishev(n2, n_cheb, temp_array, chebT)

temp_array(1:n2,2)=field_lt(0:n2-1,3) - zero_array(1:n2,3)
call chebishev(n2, n_cheb, temp_array, chebLd)

temp_array(1:n2,2)=field_lt(0:n2-1,4)
call chebishev(n2, n_cheb, temp_array, chebTd)

do j = 1, n_cheb
  write(*,919) j, chebL(j), chebT(j), chebLd(j), chebTd(j)
enddo
write(*,910)

910 format(70('*'))
914 format(f10.4,5e12.4)
916 format(5X,'Lattice of k-vectors with step ',f10.4,' and length ',
:          f10.4)
915 format(5X,'Starting calculations of the field Fourier transforms')
920 format(5X,' ... calculation of Delta E and mean_E complete... ')
918 format(f10.4,6e12.4)
919 format(10x,i3,4e12.4)

```

contains

```

subroutine angular
c ****
c This subroutine performs angular average of the squared longitudinal and
c transverse Fourier transforms of the field
c ****
integer :: i, j, m, bin, nq
real :: kx, ky, kz, sign, r0
real :: xrd, xid, yrd, yid, zrd, zid
real :: xrm, xim, yrm, yim, zrm, zim, k2, k
real :: r_xd, r_yd, r_zd, i_xd, i_yd, i_zd
real :: r_xm, r_ym, r_zm, i_xm, i_ym, i_zm
real :: dELr, dELi, ELr, ELi, e2L, e2T, e0k, intL
real :: e2Ld, e2Td, e1k, intLd, gg, gNg
real :: dfxr, dfxi, dfyr, dfyi, dfzr, dfzi
real :: mfxr, mfxi, mfyr, mfyi, mfzr, mfzi
complex :: e0r, e1r, eL, eT, eLd, eTd

gg      = 2.* (s0T-s0L)/(s0L + 2.*s0T)
gNg    = 2.* (sNT-sNL)/(sNL + 2.*sNT)
r0     = size0

do i = 1, n2
  kx = real(i - n2)*kstep
  do j = 1, n1 - 1
    ky = real(j - n2)*kstep
    do m = 1, n1 - 1
      kz = real(m - n2)*kstep
      k2 = kx*kx + ky*ky + kz*kz

      if(k2 > kmax*kmax) goto 100
      sign = (-1.)**(i + j + m)
      k = sqrt(k2)
      bin = int( k/kstep )

      if(bin < n2) then
        if(k2 .ne. 0.) then

! depacking the Fourier transformed arrays

        if(i.le.n2) then
          dfxr = delta_field_x(2*i,j,m)*sign*dr3
          dfxi = delta_field_x(2*i+1,j,m)*sign*dr3
          dfyr = delta_field_y(2*i,j,m)*sign*dr3
          dfyi = delta_field_y(2*i+1,j,m)*sign*dr3
          dfzr = delta_field_z(2*i,j,m)*sign*dr3
          dfzi = delta_field_z(2*i+1,j,m)*sign*dr3

```

```

mfxr = mean_field_x(2*i,j,m)*sign*dr3
mfxi = mean_field_x(2*i+1,j,m)*sign*dr3
mfyr = mean_field_y(2*i,j,m)*sign*dr3
mfyi = mean_field_y(2*i+1,j,m)*sign*dr3
mfzr = mean_field_z(2*i,j,m)*sign*dr3
mfzi = mean_field_z(2*i+1,j,m)*sign*dr3
elseif(i > n2) then
    nq = 2*(nn - i)
    dfxr = delta_field_x(nq,nn-j,nn-m)*sign*dr3
    dfxi = - delta_field_x(nq+1,nn-j,nn-m)*sign*dr3
    dfyr = delta_field_y(nq,nn-j,nn-m)*sign*dr3
    dfyi = - delta_field_y(nq+1,nn-j,nn-m)*sign*dr3
    dfzr = delta_field_z(nq,nn-j,nn-m)*sign*dr3
    dfzi = - delta_field_z(nq+1,nn-j,nn-m)*sign*dr3

    mfxr = mean_field_x(nq,nn-j,nn-m)*sign*dr3
    mfxi = - mean_field_x(nq+1,nn-j,nn-m)*sign*dr3
    mfyr = mean_field_y(nq,nn-j,nn-m)*sign*dr3
    mfyi = - mean_field_y(nq+1,nn-j,nn-m)*sign*dr3
    mfzr = mean_field_z(nq,nn-j,nn-m)*sign*dr3
    mfzi = - mean_field_z(nq+1,nn-j,nn-m)*sign*dr3

endif

call sphere(r0, n_delta_charge, delta_charge, scDelta, siDelta,
:           kx, ky, kz, xrd, xid, yrd, yid, zrd, zid)
call sphere(r0, n_mean_charge, mean_charge, scMean, siMean,
:           kx, ky, kz, xrm, xim, yrm, yim, zrm, zim)

r_xd = xrd + dfxr
r_yd = yrd + dfyr
r_zd = zrd + dfzr
i_xd = xid + dfxi
i_yd = yid + dfyi
i_zd = zid + dfzi

r_xm = xrm + mfxr
r_ym = yrm + mfyr
r_zm = zrm + mfzr
i_xm = xim + mfxi
i_ym = yim + mfyi
i_zm = zim + mfzi

dELr = r_xd*kx + r_yd*ky + r_zd*kz
dELi = i_xd*kx + i_yd*ky + i_zd*kz

```

```

ELr    = r_xm*kx + r_ym*ky + r_zm*kz
ELi    = i_xm*kx + i_ym*ky + i_zm*kz

e2L  =(dELr*ELr + dELi*ELi)      ! longitudinal product of the fields
e2Ld =(dELr*dELr + dELi*dELi)

e2T =(r_xd*r_xm +i_xd*i_xm + r_yd*r_ym + i_yd*i_ym +
:      r_zd*r_zm + i_zd*i_zm)*k2 - e2L
e2Td =(r_xd*r_xd +i_xd*i_xd + r_yd*r_yd + i_yd*i_yd +
:      r_zd*r_zd + i_zd*i_zd)*k2 - e2Ld

! Field DeltaE0 is calculated as the dipolar projection of the
! AVERAGE field (E_1 + E_2)/2
! Field DeltaE1 is calculated as the dipolar projection of the
! DIFFERENCE field Delta_E

e0r = cmplx(deltaE0(1)*r_xd + deltaE0(2)*r_yd + deltaE0(3)*r_zd,
:             deltaE0(1)*i_xd + deltaE0(2)*i_yd + deltaE0(3)*i_zd )
e0k = deltaE0(1)*kx + deltaE0(2)*ky + deltaE0(3)*kz

e1r = cmplx(deltaE1(1)*r_xd + deltaE1(2)*r_yd + deltaE1(3)*r_zd,
:             deltaE1(1)*i_xd + deltaE1(2)*i_yd + deltaE1(3)*i_zd )
e1k = deltaE1(1)*kx + deltaE1(2)*ky + deltaE1(3)*kz

eL   = e0k*cmplx(dELr,dELi)
eT   = e0r*k2 - eL
if(eT.ne.0.) then
  intL = e2L - gg*real(eL/eT)*e2T
endif

eLd  = e1k*cmplx(dELr,dELi)
eTd  = e1r*k2 - eLd
if(eTd.ne.0.) then
  intLd = e2Ld - gNg * real(eLd/eTd)*e2Td
endif

endif ! if k = 0.

field_lt(bin,1) = field_lt(bin,1) + intL
field_lt(bin,2) = field_lt(bin,2) + e2T
field_lt(bin,3) = field_lt(bin,3) + intLd
field_lt(bin,4) = field_lt(bin,4) + e2Td
nlattice(bin)   = nlattice(bin)   + 1.
endif

```

```

100    continue

enddo
enddo

if(mod(i,20)==0) then
    write(*,*) ' angular i = ',i, ' L', e2L
endif

enddo

print*, 'Finish calculations of L and T difference field'

end subroutine angular

end subroutine fourier

! Spherical Bessel functions

function j0x(x)
real :: x, j0x
if(x.ne.0.) then
    j0x = sin(x)/x
else
    j0x = 1.
endif
return
end function j0x

function j1x(x)
real :: x, j1x
if(x.ne.0.) then
    j1x = sin(x)/x**3 - cos(x)/x**2
else
    j1x = 1./3.
endif
return
end function j1x

subroutine chebishev(nn, n_cheb, array, c)
! ****
! ** Chebyshev evaluation of the the input array
! ** The ARRAY is supposed to be ordered
! ****
integer :: nn, n_cheb, k, j
real,dimension(nn,2) :: array

```

```

real,dimension(n_cheb) :: c
real,dimension(nn)      :: f
real                      :: bma, bpa, pi, sum_cheb
parameter( pi = 3.1415927 )

bma=0.5*( array(nn,1) - array(1,1) )
bpa=0.5*( array(nn,1) + array(1,1) )

do 1 k = 1,n_cheb
    y = bma*cos(PI*(k - 0.5)/n_cheb) + bpa

    do j = 1, nn
        if(array(j,1) .gt. y) go to 10
    enddo
10   f(k) = array(j - 1,2) + (y - array(j-1,1))*
:           (array(j,2) - array(j - 1,2))/(array(j,1) - array(j-1,1))
1    continue

fac = 2./n_cheb
do 3 j = 1, n_cheb
    sum_cheb = 0.d0
    do 2 k =1, n_cheb
        sum_cheb = sum_cheb + f(k)*cos((PI*(j-1))*((k-0.5d0)/n_cheb))
2    continue
    c(j) = fac*sum_cheb
3    continue

return
end subroutine chebishev

```

## A.4 Density Response

```
subroutine deltaGdens(etaT, dr)
use fields
use solv_parameters
! ****
! This subroutine calculates the density reorganization energy
! of the donor-acceptor complex. The sequence of steps:
! 1) calculate the c_0s on the grid
! 2) calculate the integral with the scalar product of the solute
!    difference and mean fields
! ****
implicit none

integer FFTW_FORWARD,FFTW_BACKWARD
parameter (FFTW_FORWARD=-1,FFTW_BACKWARD=1)

integer FFTW_REAL_TO_COMPLEX,FFTW_COMPLEX_TO_REAL
parameter (FFTW_REAL_TO_COMPLEX=-1,FFTW_COMPLEX_TO_REAL=1)

integer FFTW_ESTIMATE,FFTW_MEASURE
parameter (FFTW_ESTIMATE=0,FFTW_MEASURE=1)

integer FFTW_OUT_OF_PLACE,FFTW_IN_PLACE,FFTW_USE_WISDOM
parameter (FFTW_OUT_OF_PLACE=0)
parameter (FFTW_IN_PLACE=8,FFTW_USE_WISDOM=16)

integer FFTW_THREADS
parameter (FFTW_THREADS=128)
integer*8 plan

integer :: nn, status, n1, i_center, i, j, m, n3
integer :: jj, ix, l
real :: dr, dr3, k2, pi, a1, a2, kstep, kmax, rangek
real :: sign, x, y, z, dx, dy, dz, r2
real :: fx, fy, fz, sum_r, r, kx, j1x, ss, dk, factor_k
real :: sum_e, shield, r1, fxm, fym, fzxm, sum_del
real, pointer, dimension(:,:,:,:) :: c0s
complex, pointer, dimension(:,:,:,:) :: c0sk
parameter( pi = 3.14159, factor_k=20.)
real :: etaT

n3 = n_grid * n_grid * n_grid
nn = n_grid
n1 = nn - 1
dr3 = dr*dr*dr
shield = 0.1*sigma/2.
```

```

r1      = radius

! kstep is the step in corresponding to Fourier transform
! kmax gives the radius of the sphere in the k-space

kstep = 2.*pi/float(nn)/dr
kmax  = kstep * n2
ss    = - g_contact

! ss is the value of the amplitude of the spherical region
! subtracted from the hard core of the solute in order to
! decrease the load of numerical Fourier transform

! allocating the array of direct correlation function c0s

allocate(  c0s(0:n1,0:n1,0:n1), c0sk(0:n2,0:n1,0:n1)  )

! c0s is the step function equal to -1 inside the solute core
! and equal to zero outside. It is multiplied by (-1)^(n1+n2+n3)
! to convert from the [-L/2, L/2] integration range to
! [0,L]
call direct_corr(dr, ss, c0s)

! Fourier transfer call from real to complex array

call rfftw3d_f77_create_plan(plan,nn,nn,nn,
:      FFTW_REAL_TO_COMPLEX, FFTW_ESTIMATE)
call rfftwnd_f77_one_real_to_complex(plan, c0s, c0sk)
call rfftwnd_f77_destroy_plan(plan)
write(*,*) '           ... forward Fourier transform for c0s'

! output is compared to the analytical equation of a spherical
! region with the effective radius of the solute

! do ix = 0, n2, 4
! x = kstep*(ix - n2)
! write(*,*) x, c0sk(ix,n2,n2)*(-1.)**(ix)*dr3+
! :           ss*4.*pi*radius**3*j1x(radius*x),
! :           -4.*pi*radius**3*j1x(radius*x)
! enddo

! this call multiplies c0s with (S(k)-1) where S(k) is
! density structure factor

call py

```

```

    call rfftw3d_f77_create_plan(plan,nn,nn,nn,
     :      FFTW_COMPLEX_TO_REAL, FFTW_ESTIMATE)
call rfftwnd_f77_one_complex_to_real(plan, c0sk, c0s)
    call rfftwnd_f77_destroy_plan(plan)

!           write(*,*) ' ** F^-1[c0s[S(k)-1]] on',nn,'^3 lattice'

deallocate(c0sk)

c Starting integration of h0s with the scalar product of
c the diffdrence field and the mean field

sum_r = 0.
sum_e = 0.
sum_del = 0.
do i = 0, n_grid - 1
    do j = 0, n_grid - 1
        do m = 0, n_grid - 1
            sign = (-1.) ** (i + j + m)
            x = dr*(i - n2)
            y = dr*(j - n2)
            z = dr*(m - n2)

        do 10 l = 1, n_sphere
            dx = solute(l,1) - x_center - x
            dy = solute(l,2) - y_center - y
            dz = solute(l,3) - z_center - z
            r2 = dx*dx + dy*dy + dz*dz

c The par distribution function is very sharp close to the solute surface.
c Therefore a thin protective shield is added top ensure the point does not fall
c inside the solute molecular core

            if(r2 < (solute(l,4)+shield)*(solute(l,4)+shield)) goto 11
10        continue

c I have the alterating sign here and it means I need to get pure c0s
c in the integral over r

            fzm = 0.
            fym = 0.
            fxm = 0.

            do jj = 1, n_mean_charge
                dx = x - mean_charge(jj,1) + x_center
                dy = y - mean_charge(jj,2) + y_center
                dz = z - mean_charge(jj,3) + z_center

```

```

r2 = dx * dx + dy * dy + dz * dz
r = sqrt(r2)
fxm = fxm + mean_charge(jj,4)*dx/r/r2
fym = fym + mean_charge(jj,4)*dy/r/r2
fzm = fzm + mean_charge(jj,4)*dz/r/r2
enddo

fx = 0.
fy = 0.
fz = 0.

do jj = 1, n_delta_charge
    dx = x - delta_charge(jj,1) + x_center
    dy = y - delta_charge(jj,2) + y_center
    dz = z - delta_charge(jj,3) + z_center

    r2 = dx * dx + dy * dy + dz * dz
    r = sqrt(r2)
    fx = fx + delta_charge(jj,4)*dx/r/r2
    fy = fy + delta_charge(jj,4)*dy/r/r2
    fz = fz + delta_charge(jj,4)*dz/r/r2
enddo

sum_r = sum_r + (fx*fxm+fy*fym+fz*fzm)*sign*c0s(i,j,m)
sum_del = sum_del + (fx*fx+fy*fy+fz*fz)*sign*c0s(i,j,m)
sum_e = sum_e + (fx*fxm+fy*fym+fz*fzm)

11      continue
enddo
enddo
enddo

deallocate ( c0s )
lambda_d0 = sum_del * dr3/8./pi
deltaGd0 = sum_r * dr3/8./pi
deltaGind = sum_e * dr3/8./pi * 3. * (ei - 1.)/(ei + 2) ! induction component

write(*,*) ' ===== Density component at T = ',tempr
write(*,*) ' lambda_d ',lambda_d0,' deltaGind ', deltaGind
write(*,*) ' deltaGd ',deltaGd0
write(*,*) ' -----',
910 format(70('*'))
914 format(f10.4,5e12.4)

contains

```

```

subroutine py
! ****
! This routine creates a three-dimentional array on the cube of k-vactors
! (-a,a) x (-a,a) x (-a,a) the the real-type packing of the data in the direct
! Fourier transform
! ****
integer :: ix, iy, iz
real    :: kx, ky, kz, k2, k, q, sign, ff, j1x

do ix = 0, n2
    kx = (ix - n2)*kstep
do iy = 0, n1
    ky = (iy - n2)*kstep
do iz = 0, n1
    kz = (iz - n2)*kstep
    k2 = kx*kx + ky*ky + kz*kz
    k = sqrt(k2)
    call qu(k*sigma, etaT, q)

    ff = c0sk(ix,iy,iz) +
:      (-1.)**(ix+iy+iz)/dr3*4.*pi*ss*radius**3*j1x(k*radius)
    c0sk(ix,iy,iz) = ff /n3 *(1./q - 1.)

enddo
enddo
enddo
end subroutine py

end subroutine deltaGdens

subroutine direct_corr(dr, ss, c0s)
use fields
use solv_parameters
! ****
! This routine calculates the step function for the solute:
! -1 inside the solute, 0 otherwise
! ****
implicit none
integer    :: i, j, m, i_center, l, n, jj
real,dimension(0:n_grid-1,0:n_grid-1,0:n_grid-1)::c0s
real       :: dr, dx, dy, dz, r, r2, x, y, z
real       :: sign, c2, ss

c0s = 0.
c2  = - g_contact
do i = 0, n_grid - 1
    do j = 0, n_grid - 1

```

```

do m = 0, n_grid - 1
    sign = (-1.) ** (i+m+j)

! sign is introduced to take into account that the r-integration is performed
! in the range [-a,a] for each of the spatial variables

    x = dr*(i - n_grid/2)
    y = dr*(j - n_grid/2)
    z = dr*(m - n_grid/2)

do 10 l = 1, n_sphere
    dx = solute(l,1) - x_center - x
    dy = solute(l,2) - y_center - y
    dz = solute(l,3) - z_center - z
    r2 = dx*dx + dy*dy + dz*dz

        if(r2 < solute(l,4)*solute(l,4)) goto 11
10    continue

        if(x*x + y*y + z*z < radius*radius) then
            c0s(i,j,m) = - ss * sign
        else
            c0s(i,j,m) = 0.
        endif
        goto 12

11    if(x*x + y*y + z*z < radius*radius) then
        c0s(i,j,m) = (c2 - ss) * sign
    else
        c0s(i,j,m) = c2 * sign
    endif

12    continue
    enddo
enddo
enddo

return

end subroutine direct_corr

```

## A.5 Stokes shift dynamics

```
subroutine dynamics(namesolv, dr, n_cheb, chebLd, chebTd)
! ****
! This subroutine calculates the Stokes shift dynamics of
! the difference field calculated from the initial and final
! charge distribution. The input is the experimental dielectric
! constant spectrum epsilon(s), where s is the Laplace transform
! variable

! The formalism is published in
! J. Chem. Phys. 122, 044502 (2005)

! The output of the calculations is the function F(s) = -s E(s), where
! E(s) is the Laplace transform of the time-dependent solvation energy
! [Eq. (6)]. The Stokes shift correlation function is then calculated
! from Eqs. (43)-(45)
! The array response is not used in the current implementation
! since the output file is normally used by other programs
! ****

use fields
use solv_parameters
implicit none
integer :: n_cheb, length, it, nout, is, nfin, ix, ns
parameter ( nout = 29 )
real    :: s, intL, intL0, intT, intT0
real    :: sstep, smax, ess, c0, c0T
parameter( smax = 160., ns = 200 )
real    :: fp, epsilon_s, ssmall, empL, empT
real    :: sNLs, sNTs, gNKs, scaling, k, pi, kstep, a, b
real    :: qNL, qNT, ssL, ssT, lamL, lamT, fln, ft
real    :: xiNL, xiNT, j0x, chebev
real    :: cL, cT, cL0, cT0, z0, constant, dr
real    :: lambda, lambda0, lambda_L, lambda_T
real, external :: funl, funt
parameter( scaling = 10., pi = 3.14159, constant = 14.3995 )
real,dimension(n_cheb) :: chebLd, chebTd
real, dimension(ns,3) :: response
character*80          :: StokesFile
character(len=15)      :: namesolv

! parameters of dielectric relaxation for the solvent
! character(len=15) :: namesolv

call dielectric_data(namesolv)
```

```

! creating the file name with temperature extension

StokesFile = 'Stokes.dat'
length      = len_trim(StokesFile) + 1
it          = int(tempr)
write(StokesFile(length:length),'(a)' '.')
! adding temperature to the name extension
if(it > 100) then
    write(StokesFile(length+1:length+4),'(i3)') it
elseif(it<100) then
    write(StokesFile(length+1:length+3),'(i2)') it
endif

open(nout,file=StokesFile)

! effective dipolar density of a polar-polarizable solvent

ei = epsilon_omega(n_dielectric + 1)
es = epsilon_omega(1)
!     write(*,*) 'es',es, ' ei ',ei,' T ',tempr,' n_d ',n_dielectric
!     write(*,*) ' alpha',alpha,' mu',mu, ' sigma',sigma,' eta',eta

call scfy(tempr, tempr, mu, 0., sigma, 0., eta, alphap,
:           alpha, sqrt(ei),0.0, yeff, yinf, yquad)

! k=0 structure factors of nuclear polarization

yeffn = yeff - yinf
sNL   = (1./ei - 1./es)/3./yeffn
gNK   = (es - 1.)*(2.*es + 1.)/9./es/yeff
sNT   = (3. * gNK - sNL)/2.
c0    = sNL * 3. * yeffn
c0T   = sNT * 3. * yeffn
! k=0 structure factors are calculated according from prescription in JCP'05

write(*,915)
write(*,917) tempr, yeff, yinf
write(*,918) sNL, sNT

call root(sNL, funl, xiNL)
lamL = 3.*xiNL/(1+4.*xiNL)
empL = 0.95 ! empirical parameter rescaling the L-response
ssL  = 2.      ! factor to be used in the Baxter Q-function
write(*,*) ' xiNL',xiNL

call root(sNT, funt, xiNT)
if(xiNT < 0.5) then

```

```

    lamT = 1.5*xiNT/(1-2.*xiNT)
else
lamT = 0.
endif
empT = 1.          ! no need for rescaling for T-response
ssT = -1.

write(*,*) ' xiL ',xiNL,' xiT ',xiNT

nfin = (n2-1)*nint(scaling) - 1
kstep = 2.*pi/float(n_grid)/dr

write(*,*) 'dr',dr,' nfin ',nfin

! Calculation of the Laplace transform of the energy gap correlation function

sstep = smax / real(ns)
ssmall = 0.1
s = - ssmall

! generating the Laplace transform of the Stokes shift function
! [a, b] is the inteval of Chebyshev approximation

a = 0.
b = n2*kstep

sloop: do is = 1, ns

if(is < 22) then
    s = s + ssmall
else
    s = s + sstep
endif

! s-dependent parameters at k=0

ess = epsilon_s(s)
sNLs = (1./ei - 1./ess)/3./yeffn
gNKs = (ess - 1.)*(2.*ess + 1.)/9./ess/yeff
sNTs = (3. * gNKs - sNLs)/2.
fp = 1.

!      write(*,*) ' s',s,' ess ',ess, ' sNTs ',sNTs

! Integration with Chebyshev approximated fields
! The first point of the trapesoid integration comes with
! 0.5 coefficient, one does not need to worry about the end point

```

```

call qu(0., ssL*xiNL, qNL)
call chiL(c0, 0., 1./qNL, s, cL)
!   write(*,*) ' cL ',cL
intL      = field_lt(0,3)/2.*cL ! L component of reorg.energy
!   write(*,*) ' intL ',intL

! intL0 calculates the macroscopic dielectric response and intL
! gives the microscopic response

call chiL(c0, 0., sNL, s, cL0)
intL0      = field_lt(0,3)/2.*cL0 ! L component of reorg.energy

call qu(0., ssT*xiNT, qNT)
call chiT(c0T, 0., 1./qNT, s, cT)
intT      = field_lt(0,4)/2.*cT ! T component of reorg.energy
call chiT(c0T, 0., sNT, s, cT0)
intT0      = field_lt(0,4)/2.*cT0 ! T component of reorg.energy

do ix = 1, nfin
  k = ix * kstep/scaling

! longitudinal component of the response

call qu(empL*k*sigma, ssL*xiNL, qNL)
z0 = (4.*pi*TotalDeltaCharge*j0x(k*radius))**2

fln = chebev(a,b,chebLd,n_cheb,k)
call chiL(c0, k, 1./qNL, s, cL)
intL = intL + (fln + z0)*cL
call chiL(c0, 0., sNL, s, cL0)
intL0 = intL0 + (fln + z0)*cL0

! transverse component of the response

call qu(empT*k*sigma, ssT*xiNT, qNT)

ft = chebev(a,b,chebTd,n_cheb,k)
call chiT(c0T, k, 1./qNT, s, cT)
intT = intT + ft*cT
call chiT(c0T, 0., sNT, s, cT0)
intT0 = intT0 + ft*cT0

!   write(39,*) k, fln, ft, cL*4.*pi/3./yeffn, cT*4.*pi/3./yeffn

enddo

```

```

!      write(*,*) ' intL',intL,' intT',intT

lambda =(intL + intT *sNLs/gNKS)*kstep/2./pi**2/scaling*constant
lambda0=(intL0+ intT0*sNLs/gNKS)*kstep/2./pi**2/scaling*constant
lambda_L = intL * kstep/2./pi**2/scaling*constant
lambda_T = intT*sNLs/gNKS*kstep/2./pi**2/scaling * constant

write(*,*) 's',s,' lambda ',lambda,' lambda0 ',lambda0
write(nout,*) s, lambda, lambda0, lambda_L, lambda_T

response(is,1) = s
response(is,2) = lambda
response(is,3) = lambda0

enddo sloop

!      open(36,file='fs.dat')
!      do is = 2, ns
!      s = response(is,1)
!      write(*,*) s,(-response(is,2)+response(1,2))/s,
!      :           (-response(is,3)+response(1,3))/s
!      write(36,*) s,(-response(is,2)+response(1,2))/s,
!      :           (-response(is,3)+response(1,3))/s
!      enddo
!      close(36)

deallocate(beta, tau, epsilon_omega )
close(nout)

return
915 format(5x,' Calculation of the Stokes shift correlation function')
917 format(5x,' T = ',f10.4,' yeff = ',f10.4,' yinf = ',f10.4)
918 format(15x,'      sNL = ',f10.4,' sNT = ',f10.4)

end subroutine dynamics

! response functions from JCP'05 paper

subroutine chiL(c0, k, Sk, s, cL)
use solv_parameters
implicit none
real :: Sk, s, cL, c0, pi, ess, k, epsilon_s, einf
parameter ( pi = 3.14159 )

ess = epsilon_s(s)
einf = epsilon_omega(n_dielectric + 1)

```

```

cL = c0/4./pi/(sNL/Sk + 1./(1.+p_param*(sigma*k)**2)*
:      einf*(es - ess)/es/(ess - einf) )
return
end subroutine chiL

subroutine chiT(c0, k, Sk, s, cT)
use solv_parameters
implicit none
real :: k, Sk, s, cT, ess, pi, c0, epsilon_s, einf
parameter ( pi = 3.14159 )

ess = epsilon_s(s)
einf = epsilon_omega(n_dielectric + 1)

cT = c0/4./pi/(sNT/Sk + 1./(1.+p_param*(sigma*k)**2)*
:      (es - ess)/(ess - einf) )

return
end subroutine chiT

function epsilon_s(s)
use solv_parameters
implicit none
integer :: i
real :: s, ess, epsilon_s
ess = epsilon_omega(n_dielectric + 1)
do i = 1, n_dielectric
ess = ess + (epsilon_omega(i) - epsilon_omega(i+1))/
:           (1 + s*tau(i))**beta(i)
enddo

epsilon_s = ess

end function epsilon_s

```