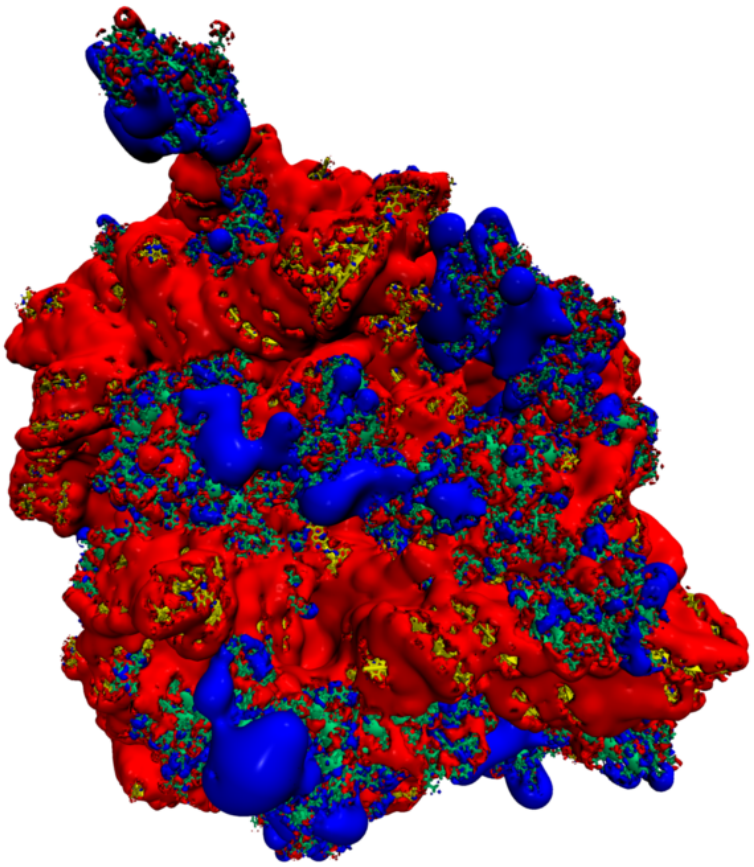
Potential Derived Charges

version 1.0

Users' Manual

Maciej Długosz
Joanna Trylska



Contents

1 Introduction **3**

2 Theory **4**

3 Supported Platforms **4**

4 Requirements **5**

5 Compilation **5**

6 Input Files **5**
6.1 PQR File 5
6.2 DX Files 7
6.3 Text Input File 7

7 Running PDC **8**

8 Output Files **9**

9 Adding New Effective Charges' Definitions **9**

10 Example **17**

11 To-Do List **22**

1 Introduction

PDC (Potential Derived Charges) is a package to describe macromolecules as a limited set of optimized, discrete effective charges immersed in a uniform dielectric, according to concepts that were previously described [1, 2]. These effective charges are fitted to best reproduce the molecular electrostatic potential calculated from the numerical solution of the Poisson-Boltzmann equation. By using effective charges one can greatly reduce the computational cost of evaluating the electrostatic forces and energies in molecular simulations, especially for the systems that are too large to be studied on the atomic level. Effective charges can be also employed in various coarse-grained approaches.

The PDC source code is written solely in the C language and parallelized with the OpenMP technology. The code was tested on various Linux/UNIX platforms.

Please direct your comments and questions about PDC to:

mdlugosz@icm.edu.pl
Interdisciplinary Centre for Mathematical and Computational Modelling,
University of Warsaw
Żwirki i Wigury 93
02-089 Warsaw
Poland

The source code of the PDC package can be downloaded from:

http://bionano.icm.edu.pl/software

2 Theory

The effective charges are derived by fitting the electrostatic potential resulting from the Debye-Hückel approximation to the external molecular potential obtained as a numerical solution of the Poisson-Boltzmann equation. Fitting is performed by solving a linear system of equations:

$$\mathbf{A}\vec{q} = \vec{b}$$

with:

$$A_{ij} = \int_\Omega d^3\vec{r} \frac{e^{-\kappa|\vec{r}-\vec{r}_i|}}{\epsilon|\vec{r}-\vec{r}_i|} \frac{e^{-\kappa|\vec{r}-\vec{r}_j|}}{\epsilon|\vec{r}-\vec{r}_j|}$$

$$\vec{q} = (q_1, ..., q_n)$$

$$\vec{b} = (b_1, ..., b_n)$$

$$b_i = \int_\Omega d^3\vec{r} \frac{e^{-\kappa|\vec{r}-\vec{r}_i|}}{\epsilon|\vec{r}-\vec{r}_i|} \Phi^{PB}(\vec{r})$$

where Ω denotes the volume outside the molecule where the electrostatic potential is fitted, $\Phi^{PB}(\vec{r})$ is the value of the electrostatic potential at a given point $\vec{r}$ obtained by numerically solving the Poisson-Boltzmann equation, ϵ is the dielectric constant of the solvent, κ is the inverse of the Debye length and $\vec{q}$ is the vector of effective charges.

To maintain physical magnitudes of effective charges, a penalty function is used that modifies the diagonal elements of the $\mathbf{A}$ matrix as follows:

$$A_{ii} = A_{ii}(1 + \delta)$$

where δ is a restraint value ($\delta > 0.0$).

The accuracy of the fit is defined as:

$$\chi = 1 - \frac{\int_\Omega d^3\vec{r} |\Phi^{PB}(\vec{r}) - \sum q_j \frac{e^{-\kappa|\vec{r}-\vec{r}_j|}}{\epsilon|\vec{r}-\vec{r}_j|}|^2}{\int d^3\vec{r} |\Phi^{PB}(\vec{r})|^2}$$

Effective charges are positioned on particular atoms of the input molecule (for example on the `NZ` atom of `LYS`, `OD1` and `OD2` atoms of `ASP` and `OE1` and `OE2` atoms of `GLU`). However, different definitions of effective charges are possible that require only small modifications of the PDC source code, making the application of PDC to nonstandard systems straightforward. Modifications of the source code needed to introduce new definitions of effective charges are described in Section 9.

3 Supported Platforms

PDC is distributed as a `tar.gz` archive containing the C source of the package. Additionally, the PDC archive contains the documentation and examples. Under Linux/UNIX systems a standard `make` tool is needed to build the PDC binary from the source. PDC was developed and tested on Linux/UNIX platforms. However, PDC should also work on other different platforms. It is possible to run PDC, for example, under MS Windows either directly or using the Cygwin environment.

4 Requirements

Linear Algebra Pack (LAPACK http://www.netlib.org/lapack/) is required to compile and run PDC.
A compiler with an OpenMP implementation is needed if the user plans to run PDC in parallel.

5 Compilation

After downloading, the PDC.tar.gz archive should be unpacked

```
gzip -d PDC.tar.gz
tar -xvf PDC.tar
```

as a result the `PDC` directory is created. Two Makefiles are provided in the `PDC/src` directory: `Makefile.gcc` and `Makefile.intel` The user should choose one of the Makefiles (depending on the user's compiler, i.e. GNU C Compiler or Intel C Compiler), verify that the path to a compiler is correct, and rename (or copy) the chosen file to `Makefile`. Next, from the `src` directory the

```
make all
```
and
```
make install
```

commands should be issued. As a result, the PDC binary, `pdc` will be placed in the `PDC/bin` directory.

6 Input Files

Four files are needed to run PDC: a text input file, a PQR file containing the positions, partial charges and radii of all atoms of the studied molecule, a DX-formatted file with the electrostatic potential of the molecule, and a DX-formatted file containing the definition of the volume where the electrostatic potential is to be fitted. Both DX files can be generated as an output from the APBS [3] package.

6.1 PQR File

PQR format resembles the PDB format, however the PQR files are rather loosely formated. PDC reads PQR files on a per-line basis using the following format:

```
Field_name Atom_number Atom_name Residue_name Residue_number x y z Charge Radius
```

with the whitespace being the delimiter. PDC ignores the lines that do not begin with the `ATOM` field. An example of the PQR file is given below:

```
ATOM      1 N    NTR     1      20.822 -31.315 -30.260  0.1305   1.550
ATOM      2 H1   NTR     1      21.587 -30.672 -30.405  0.2083   1.200
ATOM      3 H2   NTR     1      20.248 -31.331 -31.091  0.2083   1.200
ATOM      4 H3   NTR     1      21.190 -32.177 -29.884  0.2083   1.200
ATOM      5 CA   ARG     1      19.956 -30.765 -29.199 -0.0223   1.700
ATOM      6 HA   ARG     1      18.964 -30.601 -29.621  0.1242   1.200
ATOM      7 CB   ARG     1      19.822 -31.688 -27.963  0.0118   1.700
ATOM      8 HB2  ARG     1      19.036 -31.275 -27.330  0.0226   1.200
ATOM      9 HB3  ARG     1      19.520 -32.672 -28.321  0.0226   1.200
ATOM     10 CG   ARG     1      21.103 -31.813 -27.161  0.0236   1.700
ATOM     11 HG2  ARG     1      21.862 -32.219 -27.830  0.0309   1.200
ATOM     12 HG3  ARG     1      21.390 -30.807 -26.856  0.0309   1.200
ATOM     13 CD   ARG     1      20.980 -32.725 -25.898  0.0935   1.700
ATOM     14 HD2  ARG     1      21.954 -32.789 -25.414  0.0527   1.200
ATOM     15 HD3  ARG     1      20.262 -32.279 -25.210  0.0527   1.200
ATOM     16 NE   ARG     1      20.535 -34.070 -26.229 -0.5650   1.550
ATOM     17 HE   ARG     1      19.559 -34.281 -26.074  0.3592   1.200
ATOM     18 CZ   ARG     1      21.331 -35.039 -26.638  0.8281   1.700
ATOM     19 NH1  ARG     1      22.626 -34.847 -26.837 -0.8693   1.550
ATOM     20 HH11 ARG     1      23.029 -33.935 -26.674  0.4494   1.200
ATOM     21 HH12 ARG     1      23.205 -35.613 -27.151  0.4494   1.200
ATOM     22 NH2  ARG     1      20.810 -36.211 -26.935 -0.8693   1.550
ATOM     23 HH21 ARG     1      19.814 -36.354 -26.847  0.4494   1.200
ATOM     24 HH22 ARG     1      21.409 -36.961 -27.249  0.4494   1.200
ATOM     25 C    ARG     1      20.471 -29.388 -28.796  0.7214   1.700
ATOM     26 O    ARG     1      21.681 -29.155 -28.776 -0.6013   1.500
ATOM    888 N    ALA    58      13.551 -36.873 -30.609 -0.3821   1.550
ATOM    889 H    ALA    58      13.600 -36.413 -29.711  0.2681   1.200
ATOM    890 CA   ALA    58      12.251 -37.347 -31.074 -0.1747   1.700
ATOM    891 HA   ALA    58      12.386 -38.302 -31.582  0.1067   1.200
ATOM    892 CB   ALA    58      11.275 -37.555 -29.869 -0.2093   1.700
ATOM    893 HB1  ALA    58      11.123 -36.606 -29.356  0.0764   1.200
ATOM    894 HB2  ALA    58      10.318 -37.926 -30.236  0.0764   1.200
ATOM    895 HB3  ALA    58      11.703 -38.278 -29.175  0.0764   1.200
ATOM    896 C    CTR    58      11.656 -36.367 -32.099  0.7731   1.700
ATOM    897 O    CTR    58      12.129 -35.223 -32.306 -0.8055   1.500
ATOM    898 OXT  CTR    58      10.666 -36.716 -32.749 -0.8055   1.500
```

6.2 DX Files

PDC reads electrostatic potentials and volumes, given as discretized scalar data, using the OpenDX format (http://www.opendx.org). It also outputs discretized data using this format:

```
comments
object 1 class gridpositions counts nx ny nz
origin xmin ymin zmin
delta hx 0.0 0.0
0.0 hy 0.0
delta 0.0 0.0 hz
object 2 class gridconnections counts nx ny nz
object 3 class array type double rank 0 times n data follows
u(0,0,0) u(0,0,1) u(0,0,2)
u(0,0,nz-3) u(0,0,nz-2) u(0,0,nz-1)
u(0,1,0) u(0,1,1) u(0,1,2)
...
u(0,1,nz-3) u(0,1,nz-2) u(0,1,nz-1)
...
u(0,ny-1,nz-3) u(0,ny-1,nz-2) u(0,ny-1,nz-1)
u(1,0,0) u(1,0,1) u(1,0,2)
...
attribute "dep" string "positions"
object "regular positions regular connections" class field
component "positions" value 1
component "connections" value 2
component "data" value 3
```

where the variables are:

`comments` - any number of comment lines, each line starting with the # symbol
`nx ny nz` - the number of grid points in the x-, y-, and z-directions
`xmin ymin zmin` - the coordinates of the grid lower corner
`hx hy hz` - the grid spacings in the x-, y-, and z-directions
`n` - the total number of grid points
`u(*,*,*)` - the data values, ordered with the z-index increasing most quickly, followed by the y-index, and then the x-index

PDC reads DX files created with the APBS [3] package (http://www.poissonboltzmann.org/apbs/). Both DX files needed to run PDC must contain the same number of points. Moreover, grid sizes used during APBS computations, their spacings, and origins should be the same. The volume file should contain only 1s (in the region of fitting) and 0s (elsewhere). The volume files can be easily created with the APBS package (for details see Section 10).

6.3 Text Input File

The text input file contains such variables as the temperature, the solvent dielectric constant, ionic strength, as well as the number of processors to be used in the calculation (if OpenMP is supported). This text file

should be created adhering to the following format:

`keyword` *value*

The currently recognized keywords are:

`diel` - the dielectric constant of the solvent
`temp` - the temperature in kelvins
`debye` - the Debye length in Å
`penalty` - δ restraint value
`points` - this keyword allows the user to change the spacing (i.e., the resolution) of the input DX files, if `points` is set to 1, the original spacing is used; bigger values increase the spacing by `points` (hence decrease the resolution) - consequently the number of points at which potentials are to be evaluated/compared is smaller
`CPU` - the number of CPUs to be used for the calculation

An example of the PDC input file is given below (lines that do not begin with a recognizable keyword are ignored):

```
# an exemplary PDC input file

diel 78.54
temp 298.15
debye 7.8566
penalty 0.05
points 1
CPU 2

# solvent dielectric constant
# Temperature
# Debye length
# restraint
# number of points to skip in each direction (x,y,z)
# number of CPUs
```

7 Running PDC

To run PDC the following command should be used:

```
pdc <txt input file> <pqr file> <dx potential file> <dx volume file>
```

8 Output Files

As a result of a PDC run, four files are created:
`effective_charges.dat` - a text file with the values of effective charges
`effective_charges.pqr` - a PQR file containing only charged atoms (i.e. effective charges) of the input molecule
`potential_fit.dx` - the electrostatic potential generated by the optimized effective charges in the volume specified by the user
`potential_volume.dx` - input electrostatic potential file in the volume specified by the user

The accuracy of the fit is printed to the standard output.

9 Adding New Effective Charges' Definitions

Currently, only the positions of protein effective charges are defined in the PDC source code (see the table given at the end of this section). However, adding a new definition of an effective charge or modifying the existing one is straightforward and does not require any programming skills. To add new effective charges' definitions the user should modify two blocks, denoted in the source file `pdc.c` as `A` and `B`.

The two fragments of the code given below (blocks `A` and `B`) define the effective charges positioned on the `NZ` atoms of `LYS` residues and the `NH1` atoms of `ARG` residues:

```
/* BLOCK A: COUNTING EFFECTIVE CHARGES */

    k = 0;

     for (k = 0; k < PQR_N_atoms; k++) {
        clean_name(resname, 100);
        strcpy(resname, PQR_residue_name[k]);

        clean_name(atomname, 100);
        strcpy(atomname, PQR_atom_name[k]);

        if (resname[0] == 'L' &&
            resname[1] == 'Y' &&
            resname[2] == 'S' &&
            atomname[0] == 'N' &&
            atomname[1] == 'Z') {
            N_eff += 1;
        } else if (resname[0] == 'A' &&
                   resname[1] == 'R' &&
                   resname[2] == 'G' &&
                   atomname[0] == 'N' &&
                   atomname[1] == 'H' &&
                   atomname[2] == '1') {
            N_eff += 1;
        }
```

```
....

    }

    fprintf(stderr, "\nNUMBER␣OF␣EFFECTIVE␣CHARGES:␣%d\n", N_eff);

/* END OF BLOCK A: COUNTING EFFECTIVE CHARGES */
```

```
/* BLOCK B: EFFECTIVE CHARGES DEFINITIONS */

    i = 0;

    for (k = 0; k < PQR_N_atoms; k++) {
        clean_name(resname, 100);
        strcpy(resname, PQR_residue_name[k]);

        clean_name(atomname, 100);
        strcpy(atomname, PQR_atom_name[k]);

        if (resname[0] == 'L' &&
            resname[1] == 'Y' &&
            resname[2] == 'S' &&
            atomname[0] == 'N' &&
            atomname[1] == 'Z') {
            pos_x[i] = PQR_atom_x[k];
            pos_y[i] = PQR_atom_y[k];
            pos_z[i] = PQR_atom_z[k];
            tot_q[i] = 0.0;


            EFF_atom_x[i] = pos_x[i];
            EFF_atom_y[i] = pos_y[i];
            EFF_atom_z[i] = pos_z[i];
            EFF_atom_radius[i] = PQR_atom_radius[k];
            EFF_atom_id[i] = PQR_atom_id[k];
            EFF_residue_id[i] = PQR_residue_id[k];
            strcpy(EFF_atom_name[i], PQR_atom_name[k]);
            strcpy(EFF_residue_name[i], PQR_residue_name[k]);
            i = i + 1;

        } else if (resname[0] == 'A' &&
                   resname[1] == 'R' &&
                   resname[2] == 'G' &&
                   atomname[0] == 'N' &&
                   atomname[1] == 'H' &&
                   atomname[2] == '1') {
            pos_x[i] = PQR_atom_x[k];
            pos_y[i] = PQR_atom_y[k];
            pos_z[i] = PQR_atom_z[k];
            tot_q[i] = 0.0;


            EFF_atom_x[i] = pos_x[i];
            EFF_atom_y[i] = pos_y[i];
            EFF_atom_z[i] = pos_z[i];
            EFF_atom_radius[i] = PQR_atom_radius[k];
```

```
            EFF_atom_id[i] = PQR_atom_id[k];
            EFF_residue_id[i] = PQR_residue_id[k];
            strcpy(EFF_atom_name[i], PQR_atom_name[k]);
            strcpy(EFF_residue_name[i], PQR_residue_name[k]);
            i = i + 1;
        }
....

    }

    free(q_x);
    free(q_y);
    free(q_z);
    free(q);

/* END OF BLOCK B: EFFECTIVE CHARGES DEFINITIONS */
```

Let us assume that the user needs to define an effective charge positioned on the YYY atom of the XXX residue. For this purpose the blocks A and B of the pdc.c source code should be modified as follows (compare with the code fragment presented above):

```
/* BLOCK A: COUNTING EFFECTIVE CHARGES */

    k = 0;

     for (k = 0; k < PQR_N_atoms; k++) {
        clean_name(resname, 100);
        strcpy(resname, PQR_residue_name[k]);

        clean_name(atomname, 100);
        strcpy(atomname, PQR_atom_name[k]);

        if (resname[0] == 'L' &&
            resname[1] == 'Y' &&
            resname[2] == 'S' &&
            atomname[0] == 'N' &&
            atomname[1] == 'Z') {
            N_eff += 1;
        } else if (resname[0] == 'A' &&
                   resname[1] == 'R' &&
                   resname[2] == 'G' &&
                   atomname[0] == 'N' &&
                   atomname[1] == 'H' &&
                   atomname[2] == '1') {
            N_eff += 1;
        }  else if (resname[0] == 'X' &&
                   resname[1] == 'X' &&
                   resname[2] == 'X' &&
                   atomname[0] == 'Y' &&
                   atomname[1] == 'Y' &&
                   atomname[2] == 'Y') {
            N_eff += 1;
        }
....

    }

    fprintf(stderr, "\nNUMBER OF EFFECTIVE CHARGES: %d\n", N_eff);

/* END OF BLOCK A: COUNTING EFFECTIVE CHARGES */
```

```
/* BLOCK B: EFFECTIVE CHARGES DEFINITIONS */

    i = 0;

    for (k = 0; k < PQR_N_atoms; k++) {
        clean_name(resname, 100);
        strcpy(resname, PQR_residue_name[k]);

        clean_name(atomname, 100);
        strcpy(atomname, PQR_atom_name[k]);

        if (resname[0] == 'L' &&
            resname[1] == 'Y' &&
            resname[2] == 'S' &&
            atomname[0] == 'N' &&
            atomname[1] == 'Z') {
            pos_x[i] = PQR_atom_x[k];
            pos_y[i] = PQR_atom_y[k];
            pos_z[i] = PQR_atom_z[k];
            tot_q[i] = 0.0;


            EFF_atom_x[i] = pos_x[i];
            EFF_atom_y[i] = pos_y[i];
            EFF_atom_z[i] = pos_z[i];
            EFF_atom_radius[i] = PQR_atom_radius[k];
            EFF_atom_id[i] = PQR_atom_id[k];
            EFF_residue_id[i] = PQR_residue_id[k];
            strcpy(EFF_atom_name[i], PQR_atom_name[k]);
            strcpy(EFF_residue_name[i], PQR_residue_name[k]);
            i = i + 1;

        } else if (resname[0] == 'A' &&
                   resname[1] == 'R' &&
                   resname[2] == 'G' &&
                   atomname[0] == 'N' &&
                   atomname[1] == 'H' &&
                   atomname[2] == '1') {
            pos_x[i] = PQR_atom_x[k];
            pos_y[i] = PQR_atom_y[k];
            pos_z[i] = PQR_atom_z[k];
            tot_q[i] = 0.0;


            EFF_atom_x[i] = pos_x[i];
            EFF_atom_y[i] = pos_y[i];
            EFF_atom_z[i] = pos_z[i];
            EFF_atom_radius[i] = PQR_atom_radius[k];
```

```
            EFF_atom_id[i] = PQR_atom_id[k];
            EFF_residue_id[i] = PQR_residue_id[k];
            strcpy(EFF_atom_name[i], PQR_atom_name[k]);
            strcpy(EFF_residue_name[i], PQR_residue_name[k]);
            i = i + 1;
        } else if (resname[0] == 'X' &&
                   resname[1] == 'X' &&
                   resname[2] == 'X' &&
                   atomname[0] == 'Y' &&
                   atomname[1] == 'Y' &&
                   atomname[2] == 'Y') {
            pos_x[i] = PQR_atom_x[k];
            pos_y[i] = PQR_atom_y[k];
            pos_z[i] = PQR_atom_z[k];
            tot_q[i] = 0.0;


            EFF_atom_x[i] = pos_x[i];
            EFF_atom_y[i] = pos_y[i];
            EFF_atom_z[i] = pos_z[i];
            EFF_atom_radius[i] = PQR_atom_radius[k];
            EFF_atom_id[i] = PQR_atom_id[k];
            EFF_residue_id[i] = PQR_residue_id[k];
            strcpy(EFF_atom_name[i], PQR_atom_name[k]);
            strcpy(EFF_residue_name[i], PQR_residue_name[k]);
            i = i + 1;
        }
....

    }

    free(q_x);
    free(q_y);
    free(q_z);
    free(q);

/* END OF BLOCK B: EFFECTIVE CHARGES DEFINITIONS */
```

There is an additional `else if(){}` statement in both blocks. This new statement differs from the other `else if(){}` statements only by the condition enclosed within parentheses specifying the name of the residue and the name of an atom belonging to this residue. If the specified atom of the specified residue is present in the input PQR file, an effective charge will be placed at its position.

Currently implemented effective charges definitions

LYS	NZ
ARG	NH1
ARG	NH2
ASP	OD1
ASP	OD2
GLU	OE1
GLU	OE2
ASN	OD1
ASN	CG
GLN	CD
GLN	OE1
TYR	OH
SER	OG
THR	OG1
MET	SD
HIS(E,D)	ND
HIS(E,D)	NE
NTR	N
CTR	O (OT,OXT)
MG	MG
GDP	P
GTP	P

10 Example

In this example we compute the effective charges for the bovine pancreatic trypsin inhibitor (BPTI protein). We fit the electrostatic potential in the 3Å thick skin around the molecule. The total number of BPTI atoms is 898. The number of effective charges needed to fit the electrostatic potential of this protein with the accuracy greater than 0.99 is only 44.

All files needed to run this example are provided in the `PDC/example` directory.

First, using APBS, we compute the electrostatic potential generated by BPTI at 150mM ionic strength. We run APBS with the following input file (`PDC/example/apbs_inputs/input`):

```
read
    mol pqr bpti.pqr                    # Read molecule 1
end

# A POTENTIAL
elec name A
    mg-auto
    dime 129 129 129
    cglen 175 175 175
    fglen 70 70 70
    fgcent mol 1
    cgcent mol 1
    mol 1
    lpbe
    bcfl mdh
    ion charge 1 conc 0.150 radius 2.0
    ion charge -1 conc 0.150 radius 2.0
    pdie 4.0
    sdie 78.54
    chgm spl2
    srfm mol
    srad 0.0
    swin 0.3
    sdens 10.0
    temp 298.15
    calcenergy total
    calcforce no
    write pot dx apbs_potential
end
```

The resulting electrostatic potential is written to the `apbs_potential.dx` file.

Second, we need to create a volume file defining the region where we fit the electrostatic potential. Two APBS runs are needed for this purpose. The first run creates the file vol_A.dx describing the van der Waals surface of BPTI, inflated by 5Å (PDC/example/apbs_inputs/vol_A.input):

```
read
    mol pqr bpti.pqr                        # Read molecule 1
end

# A POTENTIAL
elec name A
    mg-auto
    dime 129 129 129
    cglen 175 175 175
    fglen 70 70 70
    fgcent mol 1
    cgcent mol 1
    mol 1
    lpbe
    bcfl mdh
    ion charge 1 conc 0.150 radius 5.0
    ion charge -1 conc 0.150 radius 5.0
    pdie 4.0
    sdie 78.54
    chgm spl2
    srfm mol
    srad 0.0
    swin 0.3
    sdens 10.0
    temp 298.15
    calcenergy no
    calcforce no
    write kappa dx vol_A
end
```

The second run creates the file vol_B.dx describing the van der Waals surface of BPTI, inflated by 8Å (PDC/example/apbs_inputs/vol_B.input):

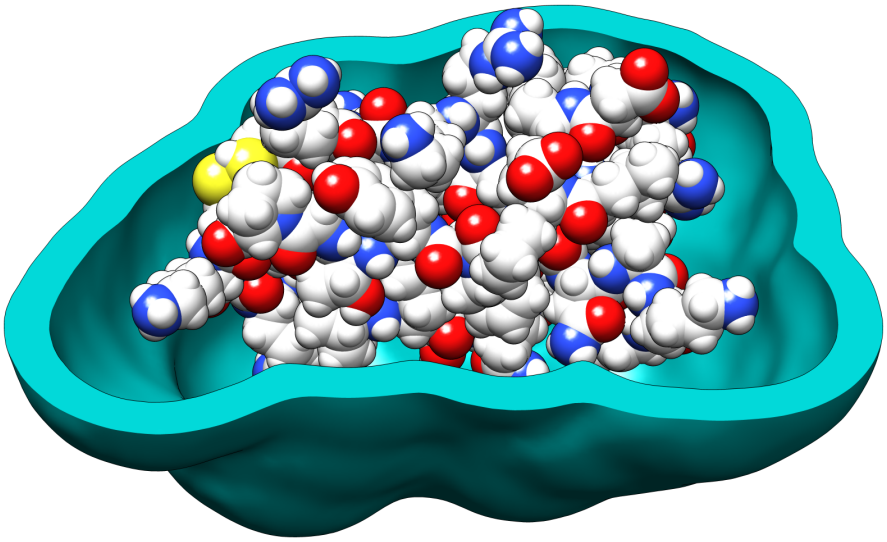
```
read
    mol pqr bpti.pqr                    # Read molecule 1
end

# A POTENTIAL
elec name A
    mg-auto
    dime 129 129 129
    cglen 175 175 175
    fglen 70 70 70
    fgcent mol 1
    cgcent mol 1
    mol 1
    lpbe
    bcfl mdh
    ion charge 1 conc 0.150 radius 8.0
    ion charge -1 conc 0.150 radius 8.0
    pdie 4.0
    sdie 78.54
    chgm spl2
    srfm mol
    srad 0.0
    swin 0.3
    sdens 10.0
    temp 298.15
    calcenergy no
    calcforce no
    write kappa dx vol_B
end
```

Next, we need to subtract the vol_A.dx and vol_B.dx files by applying the APBS dxmath utility and using the following input (PDC/example/apbs_inputs/dxmath.inp):

```
vol_A.dx
vol_B.dx -
volume.dx =
```

The resulting file, volume.dx, contains 1s in the region between 5Å and 8Å from the protein's van der Waals surface and 0s elsewhere. The electrostatic potential will be fitted in this region as shown in the following figure.



Finally, we are ready to run PDC using the following input (PDC/example/input.txt):

```
diel 78.54
temp 298.15
debye 7.8566
penalty 0.05
points 1
CPU 2
```

After issuing the command:

```
pdc input.txt bpti.pqr apbs_potential.dx volume.dx
```

the following information is printed to the standard output:

```
        ******************************
        *          pdc 1.0           *
        ******************************

OpenMP is supported

CONVERSION FACTOR:  560.354610
SOLVENT DIELECTRIC:  78.540000
TEMPERATURE:  298.150000
DEBYE LENGTH:  0.127282
RESTRAINT:  0.050000
NUMBER OF CPUs:  2

READ 898 LINES

NUMBER OF EFFECTIVE CHARGES: 44

DX DATA OK

RESOLUTION:
0.546875 0.000000 0.000000
0.000000 0.546875 0.000000
0.000000 0.000000 0.546875

Non-zero grid points 114813 out of total 2146689

FILLING [A] MATRIX ....

FILLING [b] VECTOR ....

SOLVING [A][x]=[b] ....

COMPUTING ACCURACY ....

FIT QUALITY: 0.9934160941

Time to fill A matrix = 18 seconds
```

and files: `effective_charges.dat`, `effective_charges.pqr`, `potential_fit.dx`, `potential_volume.dx` are created.

The effective_charges.pqr file contains optimized effective charges.
Two DX files, potential_fit.dx and potential_volume.dx can be visualized using for example the VMD package (http://www.ks.uiuc.edu/Research/vmd/) or UCSF Chimera http://www.cgl.ucsf.edu/chimera.

11 To-Do List

A simpler way to define effective charges, without the need to modify the PDC source code.
This feature will be added soon.

References

[1] R. R. Gabdoulline and R. C. Wade. Effective charges for macromolecules in solvent. *J. Phys. Chem.*, 100:3868–3878, 1996.

[2] D.A. Beard and T. Schlick. Modeling salt-mediated electrostatics of macromolecules: The discrete surface charge optimization algorithm and its application to the nucleosome. *Biopolymers*, 58:106–115, 2001.

[3] N.A. Baker, J.D Sept, S. Joseph, M.J Holst, and J.A. McCammon. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. USA*, 98:10037–10041, 2001.