
APBS

Release 3.1.3

Jun 25, 2021

Contents

1	Overview	3
2	Contents	5
2.1	Getting APBS	5
2.2	Using APBS	10
2.3	Solvation model background	76
2.4	Supporting APBS	80
2.5	Getting help	81
2.6	Further reading	81
2.7	File formats	83
2.8	API reference	88
2.9	Release history	88
2.10	Documentation “to-do” list	109
2.11	Indices and tables	120
	Bibliography	121
	Python Module Index	123
	Index	125

Release 3.1.3

Date Jun 25, 2021

CHAPTER 1

Overview

An understanding of electrostatic interactions is essential for the study of biomolecular processes. The structures of proteins and other biopolymers are being determined at an increasing rate through structural genomics and other efforts while specific linkages of these biopolymers in cellular pathways or supramolecular assemblages are being detected by genetic and proteomic studies. To integrate this information in physical models for drug discovery or other applications requires the ability to evaluate the energetic interactions within and between biopolymers. Among the various components of molecular energetics, solvation properties and electrostatic interactions are of special importance due to the long range of these interactions and the substantial charges of typical biopolymer components.

APBS solves the equations of continuum electrostatics for large biomolecular assemblages. This software was designed “from the ground up” using modern design principles to ensure its ability to interface with other computational packages and evolve as methods and applications change over time. The APBS code is accompanied by extensive documentation for both users and programmers and is supported by a variety of utilities for preparing calculations and analyzing results. Finally, the free, open-source APBS license ensures its accessibility to the entire biomedical community.

2.1 Getting APBS

Note: *Before you begin!* APBS funding is dependent on your help for continued development and support. Please [register](#) before using the software so we can accurately report the number of users to our funding agencies.

2.1.1 Web servers

Most functionality is available through our online web servers.

The web server offers a simple way to use both APBS and PDB2PQR without the need to download and install additional programs.

After [registering](#), please visit <http://server.poissonboltzmann.org/> to access the web server.

2.1.2 Installing from pre-compiled binaries

The best way to install APBS is via download of a pre-compiled binary from [SourceForge](#) or [GitHub releases](#) (after [registering](#), of course).

Caution: On Windows 10, if you get a popup error when running the APBS binaries, you will need to install the [Visual C++ Redistributable Package](#)

Caution: On Linux and MacOS, you may need to set your LD_LIBRARY_PATH and PATH environment variables: For example, in bash with APBS installed in \$HOME/apbs, you would set the environment variables in your .bashrc like: .. code-block:: bash

<pre>export LD_LIBRARY_PATH=\$HOME/apbs/lib:\${LD_LIBRARY_PATH} PATH=\$HOME/apbs/bin:\${PATH}</pre>	<pre>export</pre>
----------------------------------------------------------------------------------------------------------	-------------------

What's in the box?

The binary distributions typically provide the following contents:

bin contains the main APBS executable

share/apbs contains additional APBS-related files

doc the APBS programmer guide

examples APBS examples

tests the APBS test suite

tools useful programs to help process APBS input and output

include header files for building software that calls APBS

lib libraries for building software that calls APBS

2.1.3 Installing from source code

Those who enjoy an adventure can download the source code from [GitHub releases](#) and install from source code following the directions at the link below:

How to build APBS from source

These instructions assume that you have downloaded the source code from [GitHub releases](#).

<p>Caution: We do not recommend cloning directly from the head of the master branch because it is typically under development and could be unstable. Unless you really know what you are doing, we advise you to skip the next step.</p>

Get source directly from Github

Here are the commands to get the source directly from our [GitHub repository](#),

<pre>git clone https://github.com/Electrostatics/apbs cd apbs</pre>

Shortcut to build

There is a script that is used to build APBS in the Github Actions. You may want to use the file, `.build.sh`, as a template for building APBS.

Caution: When using make, there can be race conditions with CMake, autoconf, downloading dependencies, and make. It is best to run

```
VERBOSE=1 make -j 1
```

Import submodules

We are using Git submodules to manage various pieces of code. To build the master branch, after cloning it, you will need to do the following from within the top of the source directory:

```
git submodule init
git submodule update
```

Set up CMake

From the top of the source directory, the basic commands for configuring the APBS build for CMake are

```
mkdir build
cd build
# NOTE: This will be you $APBS_BUILD_DIR
export APBS_BUILD_DIR=`echo $(PWD)`
cmake ..
```

To see all the options you can run:

```
cd $APBS_BUILD_DIR
ccmake ..
```

Additional features can be built using the flags described below.

Geometric flow

If you want to use the geometric flow implementation, when invoking CMake, set **ENABLE_GEOFLOW** to ON; e.g.,

```
cd $APBS_BUILD_DIR
cmake -DENABLE_GEOFLOW=ON ..
```

Using PB-AM

If you want to use the Poisson-Boltzmann Analytical Method developed by the Teresa Head-Gordon lab, set the CMake variable **ENABLE_PBAM** to ON.

Warning: PB-AM currently runs on OS X or Linux only.

```
cd $APBS_BUILD_DIR
cmake -DENABLE_PBAM=ON ..
```

Using TABI-PB

If you want to use the Treecode-Accelerated Boundary Integral method (TABI-PB) developed by Robert Krasny and Weihua Geng, set the CMake variable **ENABLE_BEM** to ON.

TABI-PB requires the use of a molecular surface mesh generation software to create a surface representation of the molecule. By default, TABI-PB uses NanoShaper to generate an SES or Skin surface. See [TABI-PB documentation](#) for details on choosing NanoShaper. When TABI-PB runs, it will attempt to generate a surface mesh by looking in your path for the mesh generation executable. A user can obtain the appropriate executable using the steps described below. The user then must place these executables in their path.

```
cd $APBS_BUILD_DIR
cmake -DENABLE_BEM=ON ..
```

Getting NanoShaper executable

Surface meshing software executables are currently pre-built for OS X, Linux, and Windows and can be installed via CMake. The executables will be placed in the `bin` directory of your build.

NanoShaper is a molecular surface mesh generation software package developed by W. Rocchia and S. Decherchi.

```
cd $APBS_BUILD_DIR
cmake -DGET_NanoShaper=ON ..
```

Using finite element support

Warning: Finite element methods are currently only supported on POSIX-like operating systems such as OS X or Linux.

To enable finite element support, set the CMake **ENABLE_FETK** variable to ON.

On Linux, the FETK shared libraries need to be locatable by the shared library loader. One way to do this is to update **LD_LIBRARY_PATH** to point at `<build-dir>/fetk/lib`, where `<build-dir>` is the location where APBS was built. In base, this can be accomplished with the command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<build-dir>/fetk/lib:<install-dir>/fetk/lib
cd $APBS_BUILD_DIR
cmake -DENABLE_FETK=ON ..
```

Enabling APBS Python support

APBS Python support requires a local installation of [SWIG](#).

Assuming SWIG is installed, APBS Python support can be enabled by setting the CMake variable **ENABLE_PYTHON** to ON. If you are on Linux you will also need to set the CMake variable **BUILD_SHARED_LIBS** to OFF.

```
cd $APBS_BUILD_DIR
cmake -DENABLE_PYTHON=ON ..
```

Building the code - minimal

Assuming the Cmake command completed successfully, APBS can be built with

```
cd $APBS_BUILD_DIR
# Run cmake with the options you prefer:
VERBOSE=1 make -j 1
```

Building the code - advanced

```
export INSTALL_DIR=$SOME_DIR/apbs
export PATH=$INSTALL_DIR/bin:$PATH
# NOTE: In case you need to debug the source code:
# export RELEASE_TYPE=Debug
export RELEASE_TYPE=Release
# NOTE: If cmake or make fail, save yourself and make sure you remove
#       everything including the build directory. This code base uses
#       many older autoconf based projects that do not know how to save
#       state or recover from partial builds. If cmake or make fail, you
#       should figure out how to fix it and then remove everything and
#       try again.
rmdir $APBS_BUILD_DIR
mkdir -p $APBS_BUILD_DIR
cd $APBS_BUILD_DIR
# NOTE: In case you need to debug cmake, use verbose debug/trace mode:
# cmake -S .. -B $BUILD_DIR --trace-source=../CMakeLists.txt --trace-expand \
cmake \
  -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR \
  -DCMAKE_BUILD_TYPE=$RELEASE_TYPE \
  -DENABLE_GEOFLOW=ON \
  -DENABLE_BEM=ON \
  -DENABLE_FETK=ON \
  -DENABLE_OPENMP=ON \
  -DENABLE_PBAM=ON \
  -DENABLE_PBSAM=ON \
  -DENABLE_PYTHON=ON \
  -DENABLE_TESTS=ON \
  -DENABLE_TINKER=OFF \
  -DBUILD_SHARED_LIBS=ON \
  ..
VERBOSE=1 make -j 1
```

Testing APBS

```
cd $APBS_BUILD_DIR
# NOTE: Assuming you have already built APBS
# NOTE: So that the apbs and optional NanoShaper binaries are in the path:
export PATH="$APBS_BUILD_DIR/bin:$PATH"
ctest -C Release --output-on-failure
```

Installing APBS

```
export INSTALL_DIR="Some directory - default is /usr/local"
cd $APBS_BUILD_DIR
cmake \
  -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR \
  # NOTE: Add cmake options that you used during the Build APBS section
..
VERBOSE=1 make -j 1 install
```

2.1.4 Current platform support

OS	PYTHON VERSION	GE- OFLOW	BEM, NanoShaper	FETK	PB- SAM	PBAM	PYTHON SUPPORT	SHARED_LIBS
Ubuntu	3.7+	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ma- cOSX	3.7+	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Win- dows 10	3.7+	Yes	Yes	No	No	Yes	No	No

2.2 Using APBS

Note: *Before you begin!* PDB2PQR funding is dependent on your help for continued development and support. Please [register](#) before using the software so we can accurately report the number of users to our funding agencies.

APBS is often used together with the [PDB2PQR software](#); e.g., in the following type of workflow

1. Start with a [PDB ID](#) or locally generated PDB file (see [PDB molecular structure format](#)).
2. Assign titration states and parameters with **pdb2pqr** to convert the protein and ligands to PQR format (see [PQR molecular structure format](#)).
3. Perform electrostatics calculations with **apbs** (can be done from within the [PDB2PQR web server](#)).
4. Visualize results from within PDB2PQR web server or with [Other software](#).

2.2.1 Web server use

Most users will use PDB2PQR through [the web server](#) (after [registering](#), of course). However, it is also possible to install local versions of PDB2PQR and run these through the command line.

2.2.2 Command line use

```
apbs [options] input-file
```

where the list of [options] can be obtained by running APBS with the `--help` option. The input file format is described below.

2.2.3 Input file syntax

APBS has a *new input file format* that accepts **YAML**- or **JSON**- format input. The *old APBS input format* has been deprecated but will continue to be supported for the next few releases.

YAML- and JSON-format input files

In its new input format, APBS accepts either **JSON**- or **YAML**-format input files.

These input files consist of the following keywords and objects:

Data loading input file section (required)

This required section is denoted by the keyword `read` and is described in `Read`. The section includes objects indexed by the following keywords:

- `molecules`: a list of molecule input objects of class `Molecule`
 - `potential maps`: a list of electrostatic potential map input objects of class `Map`
 - `charge density maps`: a list of charge density map input objects of class `Map`
 - `ion accessibility maps`: a list of ion accessibility map input objects of class `Map`
 - `dielectric maps`: a list of dielectric map input objects of class `DielectricMapGroup`
 - `parameters`: a list of parameter files of class `Parameter`
-

Data loading input file API

Calculation input file section (required)

This required section is denoted by the keyword `calculate` and includes a list of objects of the type `apbs.input_file.calculate.Calculate`.

Todo: improve documentation with outline and/or example.

Calculation types

Finite-difference Poisson-Boltzmann calculations

Todo: Make this more user-friendly by:

- Adding introductory text with contents
 - Only showing inherited members in API documentation
 - Only showing undocumented members in API documentation
-

Nonpolar grid-based calculations

Todo: Provide overview and/or separate API documentation from main docs.

Calculate API

Results processing section (optional)

This optional section is denoted by the keyword `process` and includes lists of arithmetic operation objects (see `apbs.input_file.process.Operation`) indexed by the following keywords:

- `sums`: a list weighted sum operations
- `products`: a list of weighted product operations
- `exps`: a list of element-wise exponentiation operations

The syntax for these objects is described in `apbs.input_file.process.Process`.

See also:

Data processing input file API

Todo: finish other required and optional sections

Input file class structure

The input file object parsing and validation follows the basic pattern implemented in `apbs.input_file.InputFile` (see below). This class should serve as a template for adding new input file sections.

Old APBS input format

In the old format, APBS input files are loosely-formatted files which contain information about the input, parameters, and output for each calculation.

These files are whitespace- or linefeed-delimited. Comments can be added to the input files via the `#` character; all text between the `#` and the end of the line is not parsed by APBS. If pathnames used in the input file contain spaces, then the entire pathname must be enclosed in quotes. For example, if you wanted to refer to the file `foo` which resides in a directory with spaces in its name, then you should refer to `foo` as `"/path with spaces/foo"`. Specific examples of APBS input are provided in [Examples](#).

APBS input files contain three basic sections which can be repeated any number of times:

- *READ input file section*: Section for specifying data-reading input. For the *new APBS syntax*, see [Data loading input file section \(required\)](#).
- *ELEC input file section*: Section for specifying polar solvation (electrostatics) calculation parameters. For the *new APBS syntax*, see [Calculation input file section \(required\)](#).

- *APOLAR input file section*: Section for specifying apolar solvation calculation parameters. For the *new APBS syntax*, see *Nonpolar grid-based calculations*.
- *PRINT input file section*: Section for specifying summary output. For the *new APBS syntax*, see *Results processing section (optional)*.

The APBS input file is constructed from these sections in the following format:

```

READ
...
END

ELEC
...
END

APOLAR
...
END

PRINT
...
END

QUIT

```

These sections can occur in any order and can be repeated any number of times. However, the sections are inter-dependent. For example, PRINT requires ELEC and/or APOLAR while ELEC requires one or more READ sections. Sections can also be repeated; several READ statements may be used to load molecules and multiple ELEC or APOLAR sections would specify various electrostatics calculations on one or more molecules.

Each section has the following syntax:

```
SECTION [name <id>]
```

where the optional `name` argument allows the user to include a string to identify the section. In the absence of this argument, sections are assigned numerical IDs.

READ input file section

Note: This section has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*). See *Data loading input file section (required)* for more information.

The READ block of an APBS input file has the following general format:

```

READ
    [ keywords... ]
END

```

where `keywords` is or more of the keywords described below (the line breaks and indentation are for clarity; only whitespace is necessary).

Note: One of these sections must be present for every molecule involved in the APBS calculation. Molecule and “map” IDs are assigned implicitly assigned for each molecule/map read, based on order and starting at 1 and incre-

mented independently for each input type. In other words, each input PQR file is assigned an ID 1, 2, 3, ...; each input dielectric map is assigned an independent ID 1, 2, 3, ...; etc.

charge

This command allows APBS to read the fixed (molecular) charge density function mapped to a mesh. The inputs are maps of charge densities; these values have units of $e_c \text{ \AA}^{-3}$, where e_c is the electron charge. In general, this command will read charge-maps written by *ELEC input file section write* commands. The syntax of this command is:

```
READ charge {format} {path} END
```

format Specify the format of the charge map. Acceptable values include:

dx *OpenDX scalar data format*

gz gzipped (zlib) compressed *OpenDX scalar data format*. Files can be read directly in compressed form.

path The location of the charge map file.

diel

This command allows APBS to read the dielectric function mapped to 3 meshes shifted by one-half grid spacing in the x, y, and z directions. The inputs are maps of dielectric variables between the solvent and biomolecular dielectric constants; these values are unitless. In general, this command will read dielectric maps written by *ELEC input file section write* commands. The syntax of this command is:

```
READ diel {format} {path-x} {path-y} {path-z} END
```

format The format of the dielectric map.

dx *OpenDX scalar data format*

gz gzipped (zlib) compressed *OpenDX scalar data format*. Files can be read directly in compressed form.

path-x The location of the x-shifted dielectric map file.

path-y The location of the y-shifted dielectric map file.

path-z The location of the z-shifted dielectric map file.

Note: If you choose this option and have a non-zero ionic strength, you must also include a *READ kappa* statement.

kappa

This command allows APBS to read the ion-accessibility function mapped to a mesh. The inputs are maps of ion accessibility values which range between 0 and the build Debye-Hückel screening parameter; these values have units of \AA^{-2} . In general, this command will read kappa-maps written by *ELEC input file section write* commands. The syntax of this command is:

```
READ kappa {format} {path} END
```

format Specify the format of the charge map. Acceptable values include:

dx *OpenDX scalar data format*

gz gzipped (zlib) compressed *OpenDX scalar data format*. Files can be read directly in compressed form.

path The location of the map file.

Note: If you choose this option, you must also include a read diel statement.

mol

This command specifies the molecular data to be read into APBS. The syntax is

```
READ mol {format} {path} END
```

format The format of the input data.

pqr Specify that molecular data is in *PQR format*.

pdb Specify that molecular data is in pseudo-PDB format. If this type of structure file is used, then a parameter file must also be specified with a **READ** *parm* statement to provide charge and radius parameters for the biomolecule's atoms.

path The location of the molecular data file.

parm

This command specifies the charge and radius data to be used with pseudo-PDB-format molecule files. The syntax is:

```
READ parm {format} {path} END
```

format The format of the parameter file.

flat Specify that the parameter file is in *APBS flat-file parameter format*.

xml Specify that the parameter file is in *APBS XML parameter format*

path The location of the parameter data file.

Note: APBS provides a few example files as part of the source code distribution. Currently, example files only contain the polar parameters that can also be assigned more easily through the PDB2PQR software.

pot

This command allows APBS to read the electrostatic potential mapped to a mesh. The inputs are maps of the electrostatic potential from a previous calculation. In general, this command will read potential-maps written by by *ELEC input file section write* commands. The syntax of this command is:

```
READ pot {format} {path} END
```

format Specify the format of the charge map. Acceptable values include:

dx *OpenDX scalar data format*

gz gzipped (zlib) compressed *OpenDX scalar data format*. Files can be read directly in compressed form.

path The location of the map file.

Note: To use this functionality you must set the *bcfl* keyword to *map*. See also: *usemap*.

ELEC input file section

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- *Finite-difference Poisson-Boltzmann calculations*
-

Todo: port other versions of command

The ELEC block of an APBS input file is used for polar solvation (electrostatics) calculations and has the following syntax:

```
ELEC [ name {id} ]
      {type}
      {keywords...}
END
```

The optional *id* variable is a simple string that allows ELEC statements to be named. Since numerous ELEC blocks may appear in an APBS input file, it can be difficult to keep track of them all. It is possible to assign an optional name (string) to each ELEC block to simplify the organizational process.

The *type* command defines the types of ELEC calculation to be performed and includes:

- Finite difference multigrid calculations with **PMG**.
 - *mg-auto*
 - *mg-para*
 - *mg-manual*
- **Geometric flow solvation** finite difference calculations
 - *geoflow-auto*
- Boundary element method calculations with **TABI-PB**.
 - *tabi*
- Analytic and semi-analytic Poisson-Boltzmann approximations
 - *pbam-auto*
 - *pbsam-auto*
- Finite element calculations with **FETk**.
 - *fe-manual*
- No-op modes for generating coefficient maps
 - *mg-dummy*

Finally, the `keywords` are calculation-specific commands that customize the particular type of calculation. This section is the main component for polar solvation calculations in APBS runs. There may be several ELEC sections, operating on different molecules or using different parameters for multiple runs on the same molecule. The order of the ELEC statement can matter since certain types of boundary conditions (*bcbf*) can require information about previous calculations.

tabi

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

This mode uses the TABI-PB integral equation software from Geng and Krasny to solve the linearized Poisson-Boltzmann equation. Boundary element methods offer the ability to focus numerical effort on a much smaller region of the problem domain: the interface between the molecule and the solvent. In this method, two coupled integral equations defined on the solute-solvent boundary define a mathematical relationship between the electrostatic surface potential and its normal derivative with a set of integral kernels consisting of Coulomb and screened Coulomb potentials with their normal derivatives. The boundary element method requires a surface triangulation, generated by a program such as [NanoShaper](#), on which to discretize the integral equations.

For more information, see the Geng & Krasny 2013 [J Comput Phys](#) paper.

ion

Note: This command has been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)); see `MobileIons` for more information.

Specify the bulk concentrations of mobile ion species present in the system. This command can be repeated as necessary to specify multiple types of ions; however, only the largest ionic radius is used to determine the ion-accessibility function. The total bulk system of ions must be electroneutral which means the charge densities/concentrations of positive and negative ions must be equal. The syntax is:

```
ion charge {charge} conc {conc} radius {radius}
```

where

charge Mobile ion species charge (floating point number in ec)

conc Mobile ion species concentration (floating point number in M)

radius Mobile ion species radius (floating point number in Å)

mac

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

TABI-PB parameter, multipole acceptance criterion (MAC), that controls distance ratio at which the method uses direct summation or Taylor approximation (a particle-cluster interaction) to calculate the integral kernels. The syntax is:

```
mac {theta}
```

where `theta` is a floating-point number from 0 to 1 controlling the distance ratio. This multipole acceptance criterion (MAC) is $\frac{r_c}{R} \leq \theta$, where r_c is the cluster radius, and R is the distance of the particle to the cluster center. If the above relationship is satisfied, the Taylor approximation will be used instead of direct summation. A typical value for this parameter is 0.8.

mesh

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

TABI-PB parameter that specifies the meshing software used to generate surface mesh. The syntax is:

```
mesh {flag}
```

where `flag` is an integer indicating the meshing software to be used:

0 Formerly used for msms, no longer supported.

1 SES implementation in [NanoShaper](#)

2 Skin surface implementation in [NanoShaper](#)

Note that the executable [NanoShaper](#) must be included in your path to use them.

Todo: The integer flag values for `mesh` should be replaced by human-readable strings. Documented in <https://github.com/Electrostatics/apbs/issues/496>

mol

Note: Some versions of this command have been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)):

- Nonpolar calculations: see `nonpolar.Nonpolar.molecule()` for more information.
 - Finite difference Poisson-Boltzmann calculations: see `finite_difference.FiniteDifference.molecule()` for more information.
-

Todo: port for other calculation types

This term specifies the molecule for which the calculation is to be performed. The syntax is:

```
mol {id}
```

where `id` is the integer ID of the molecule for which the apolar calculation is to be performed. The molecule IDs are based on the order in which molecules are read by `READ mol` statements (see [READ input file section](#)), starting from 1.

outdata

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

TABI-PB parameter that specifies the file type for printing the output data. The syntax is:

```
outdata {flag}
```

where `flag` is an integer indicating the output file types:

0 .dat format

1 Both the .dat format and a VTK polygonal data file that can be visualized in the ParaView software. The VTK file contains color mappable potentials and normal derivatives of potentials on the faces and vertices of the mesh.

Todo: The integer flag values for `mesh` should really be replaced by human-readable strings.

pdie

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference Poisson-Boltzmann calculations: see `FiniteDifference.solute_dielectric()` for more information.

Todo: port for other calculation types

Specify the dielectric constant of the solute molecule. The syntax is:

```
pdie {diel}
```

where `diel` is the floating point value of the unitless biomolecular dielectric constant. This is usually a value between 2 to 20, where lower values consider only electronic polarization and higher values consider additional polarization due to intramolecular motion. The dielectric value must be ≥ 1 .

sdens

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- **Nonpolar calculations:** See `Nonpolar.surface_density()` for more information.

This keyword specifies the number of quadrature points per \AA^2 to use in calculation surface terms (e.g., molecular surface, solvent accessible surface). This keyword is ignored when *srad* is 0.0 (e.g., for van der Waals surfaces) or when *srfn (elec)* is `sp12` (e.g., for spline surfaces). The syntax is:

```
sdens {density}
```

where `density` is a floating point number indicating the number of grid points per \AA^{-2} . A typical value is 10.0.

Note: There is a strong correlation between the value used for the sphere density, the accuracy of the results, and the APBS calculation time.

sdie

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference Poisson-Boltzmann calculations: see `FiniteDifference.solvent_dielectric()` for more information.
-

Todo: port for other calculation types

Specify the dielectric constant of the solvent. The syntax is:

```
sdie {diel}
```

where `diel` is a floating point number representing the solvent dielectric constant (unitless). This number must be ≥ 1 . Bulk water at biologically-relevant temperatures is usually modeled with a dielectric constant of 78-80.

srad

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Nonpolar calculations: see `nonpolar.Nonpolar.solvent_radius()` for more information.
 - Finite-difference Poisson-Boltzmann calculations: see `finite_difference.FiniteDifference.solvent_radius()` for more information.
-

This keyword specifies the radius of the solvent molecules; this parameter is used to define various solvent-related surfaces and volumes (see *srfm (elec)*). This value is usually set to 1.4 \AA for a water-like molecular surface and set to 0 \AA for a van der Waals surface. The syntax is:

```
srad {radius}
```

where `radius` is the floating point value of the solvent radius (in \AA). This keyword is ignored for `srfm spl2` (see *srfm (elec)*).

temp

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference calculations: See `finite_difference.FiniteDifference.temperature()`.
- Nonpolar calculations: See `nonpolar.Nonpolar.temperature()`.

Todo: add other uses to new syntax

This keyword specifies the temperature for the calculation. The syntax is:

```
temp {T}
```

where T is the floating point value of the temperature for calculation (in K).

tree_n0

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

TABI-PB parameter that specifies the maximum number of particles in a treecode leaf. This controls leaf size in the process of building the tree structure. The syntax is:

```
tree_n0 {max_number}
```

where `max_number` is an integer. A typical value for this parameter is 500.

tree_order

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

TABI-PB parameter that specifies the order of the treecode multipole expansion. The syntax is:

```
tree_order {order}
```

where `order` is an integer that indicates the Taylor expansion order. Users can adjust the order for different accuracy. A typical choice for this parameter is 3.

Background information

The Treecode-Accelerated Boundary Integral Poisson-Boltzmann solver (TABI-PB; [Geng, 2013](#) calculates electrostatics of solvated biomolecules governed by the linearized Poisson-Boltzmann equation. It uses a well-posed boundary integral Poisson-Boltzmann formulation to ensure rapid convergence. In addition, a fast treecode algorithm for the screened Coulomb potential ([Li, 2009](#)) is applied to speed up the matrix-vector products in each GMRES iteration. The molecular surfaces, which divide the entire domain into solute region and solvent region, are generated by [NanoShaper](#).

TABI-PB algorithm

The coupled integral equations derived from the linearized Poisson-Boltzmann equation are

$$\begin{aligned} \frac{1}{2}(1 + \epsilon)\phi(x) - \int_{\Gamma} (K_1(x, y) \frac{\partial \phi(y)}{\partial v} + K_2(x, y) \phi(y)) dS_y &= S_1(x) \\ \frac{1}{2}(1 + \frac{1}{\epsilon}) \frac{\partial \phi(x)}{\partial v} - \int_{\Gamma} (K_3(x, y) \frac{\partial \phi(y)}{\partial v} + K_4(x, y) \phi(y)) dS_y &= S_2(x), x \in \Gamma \end{aligned}$$

for the surface potential ϕ , and its normal derivative $\frac{\partial \phi}{\partial v}$ on the surface Γ . The kernels $K_{1,2,3,4}$ are linear combinations of the Coulomb and screened Coulomb potentials:

$$\begin{aligned} G_0(x, y) &= \frac{1}{4\pi|x - y|} \\ G_{\kappa}(x, y) &= \frac{e^{-\kappa|x - y|}}{4\pi|x - y|} \end{aligned}$$

and their first and second derivatives.

The sums in the discretized form of the integral equations above have the form of N -body interactions,

$$V_i = \sum_{j=1, j \neq i}^N q_j G(x_i, x_j), i = 1, \dots, N$$

where G is the screened Coulomb potential kernel, x_i, x_j are the centroids of the triangles, and q_j is the charge at x_j . The particles (centroids of the triangles) are divided into a hierarchy of clusters having a tree structure. The treecode replaces the $\mathcal{O}(N^2)$ particle-particle interactions by $\mathcal{O}(N \log N)$ particle-cluster interactions and TABI-PB utilizes this feature efficiently.

Output

The TABI-PB code produces an output file called `surface_potential.dat` containing:

- number of nodes, number of triangles
- node index, vertices, normal vector, surface potential ($\text{kJ mol}^{-1} \text{ e}_c^{-1}$), surface potential normal derivatives ($\text{kJ mol}^{-1} \text{ e}_c^{-1} \text{ \AA}^{-1}$)
- connectivity data for surface triangulation

The format is given below:

```
num_node num_triangle
node_index x y z norm_x norm_y norm_z phi norm_phi
(et cetera)
node_index1 node_index2 node_index3
```

The TABI-PB code prints the free energy of solvation and Coulombic free energy in kJ/mol, along with some other information such as CPU time and the GMRES residuals at each step.

Additionally, TABI-PB can optionally output a VTK polygonal data file containing color mappable potentials and normal derivatives of potentials on the faces and vertices of the mesh. The VTK file can be visualized using [ParaView](#).

fe-manual

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Manually-configured adaptive finite element Poisson-Boltzmann calculations.

This is a single-point PBE calculation performed by our adaptive finite element PBE solver. It requires that APBS be linked to the Michael Holst group [FETk finite element library](#) during compilation. The finite element solver uses a “solve-estimate-refine” cycle. Specifically, starting from an initial mesh, it performs the following iteration:

1. solve the problem with the current mesh
2. estimate the error in the solution
3. adaptively refine the mesh to reduce the error

This iteration is repeated until a global error tolerance is reached.

Keywords for this calculation type include:

akeyPRE

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specifies how the initial finite element mesh should be constructed (from refinement of a very coarse 8-tetrahedron mesh prior to the solve-estimate-refine iteration in [fe-manual](#) finite element calculations. The syntax is:

```
akeyPRE {key}
```

where `key` is a text string that specifies the method used to guide initial refinement and takes one of the values:

unif Uniform refinement

geom Geometry-based refinement at molecular surfaces and charges

akeySOLVE

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specifies how the the finite element mesh should be adaptively subdivided during the solve-estimate-refine iterations of a [fe-manual](#) finite element calculation. The syntax is:

```
akeySOLVE {key}
```

where `key` is a text string that specifies the method used to guide adaptive refinement:

resi Residual-based *a posteriori* refinement.

bcfl

Note: Some versions of this command have been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)):

- Finite difference boundary conditions; see `apbs.input_file.calculate.finite_difference.FiniteDifference.boundary_condition()`.
-

Specifies the type of boundary conditions used to solve the Poisson-Boltzmann equation. The syntax is:

```
bcfl {flag}
```

where `flag` is a text string that identifies the type of conditions to be used.

zero “Zero” boundary condition. Dirichlet conditions where the potential at the boundary is set to zero. This condition is not commonly used and can result in large errors if used inappropriately.

sdh “Single Debye-Hückel” boundary condition. Dirichlet condition where the potential at the boundary is set to the values prescribed by a Debye-Hückel model for a single sphere with a point charge, dipole, and quadrupole. The sphere radius in this model is set to the radius of the biomolecule and the sphere charge, dipole, and quadrupole are set to the total moments of the protein. This condition works best when the boundary is sufficiently far from the biomolecule.

mdh “Multiple Debye-Hückel” boundary condition. Dirichlet condition where the potential at the boundary is set to the values prescribed by a Debye-Hückel model for a multiple, non-interacting spheres with a point charges. The radii of the non-interacting spheres are set to the atomic radii of and the sphere charges are set to the atomic charges. This condition works better than `sdh` for closer boundaries but can be very slow for large biomolecules.

focus “Focusing” boundary condition. Dirichlet condition where the potential at the boundary is set to the values computed by the previous (usually lower-resolution) PB calculation. This is **only** used in sequential focusing performed manually in [mg-manual](#) calculations. All of the boundary points should lie within the domain of the previous calculation for best accuracy; if any boundary points lie outside, their values are computed using single Debye-Hückel boundary conditions (see above).

map Specifying `map` allows a previously calculated potential map to be used in a new focusing calculation. A typical scenario is using the same coarse grid for multiple focusing calculations. A potential map can be written once from a coarse grid calculation, then used in subsequent runs to bypass the need to recalculate the coarse grid. See the `READ` keyword pot (see [READ input file section](#)) and the attached example files for its use.

calcenergy

Note: Some versions of this command have been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)):

- **Nonpolar calculations:** See `Nonpolar.calculate_energy()` for more information.
-

This optional keyword controls energy output from an apolar solvation calculation. The syntax is:

```
calcenergy <flag>
```

where `flag` is a string denoting what type of energy to calculate:

no (Deprecated) Don’t calculate any energies.

total Calculate and return total apolar energy for the entire molecule.

comps Calculate and return total apolar energy for the entire molecule as well as the energy components for each atom.

Note: This option must be used consistently (with the same `flag` value) for all calculations that will appear in subsequent *PRINT input file section* statements.

calcforce

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- **Nonpolar calculations:** See `Nonpolar.calculate_forces()` for more information.

This optional keyword controls energy output from an apolar solvation calculation. The syntax is:

```
calcforce {flag}
```

where `flag` is a text string that specifies the types of force values to be returned:

no (Deprecated) don't calculate any forces.

total Calculate and return total electrostatic and apolar forces for the entire molecule.

comps Calculate and return total electrostatic and apolar forces for the entire molecule as well as force components for each atom.

The possible outputs from `calcforce` are:

tot {n} total force for atom *n*

qf {n} fixed charge force for atom *n*

db {n} dielectric boundary force for atom *n*

ib {n} ionic boundary force for atom *n*

The values will be printed in three columns which correspond to the x, y, and z components of the force vector.

Note: This option must be used consistently (with the same `flag` value) for all calculations that will appear in subsequent *PRINT input file section* statements.

chgm

```
. currentmodule:: apbs.input_file.calculate.finite_difference
```

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Focus.charge_discretization()` for more information.

Specify the method by which the biomolecular point charges (i.e., Dirac delta functions) by which charges are mapped to the grid for a multigrid (*mg-manual*, *mg-auto*, *mg-para*) Poisson-Boltzmann calculation. As we are attempting to model delta functions, the support (domain) of these discretized charge distributions is always strongly dependent on the grid spacing. The syntax is:

```
chgm {flag}
```

`flag` is a text string that specifies the type of discretization:

sp10 Traditional trilinear interpolation (linear splines). The charge is mapped onto the nearest-neighbor grid points. Resulting potentials are very sensitive to grid spacing, length, and position.

sp12 Cubic B-spline discretization. The charge is mapped onto the nearest- and next-nearest-neighbor grid points. Resulting potentials are somewhat less sensitive (than `sp10`) to grid spacing, length, and position.

sp14 Quintic B-spline discretization. Similar to `sp12`, except the charge/multipole is additionally mapped to include next-next-nearest neighbors (125 grid points receive charge density).

domainLength

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the rectangular finite element mesh domain lengths for *fe-manual* finite element calculations. This length may be different in each direction. If the *usemesh* keyword is included, then this command is ignored. The syntax is:

```
domainLength {xlen ylen zlen}
```

where the parameters `xlen` `ylen` `zlen` are floating point numbers that specify the mesh lengths in the x-, y-, and z-directions (respectively) in units of Å.

ekey

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the method used to determine the error tolerance in the solve-estimate-refine iterations of the finite element solver (*fe-manual*). The syntax is:

```
ekey { flag }
```

where `flag` is a text string that determines the method for error calculation.

simp Per-simplex error limit

global Global (whole domain) error limit

frac Fraction of simplices you'd like to see refined at each iteration

etol

Note: Some versions of this command have been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)):

- For a finite difference calculation, see `Focus.coarse_grid_dimensions()` for more information.

Todo: add documentation links for other instances.

Specifies the tolerance for iterations of the partial differential equation solvers: The syntax is:

```
etol { tol }
```

where `tol` is the (floating point) numerical value for the error tolerance.

For finite difference solvers, this keyword is optional and is intended for *mg-manual*, *mg-auto*, and *mg-para* calculation types.

For finite element solvers, this keyword specifies the tolerance for error-based adaptive refinement during the solve-estimate-refine iterations of the finite element solver (*fe-manual*), where `tol` is the (floating point) numerical value for the error tolerance.

lpbe

Note: Some aspects of this command have been moved to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference:

See `FiniteDifference.equation()` for more information.

Todo: port for other types of calculations.

Specifies that the linearized Poisson-Boltzmann equation should be solved.

Note: The options *lpbe*, *npbe*, *lrpbe*, *nrpbe* are mutually exclusive.

lrpbe

Note: Some aspects of this command have been moved to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference:

See `FiniteDifference.equation()` for more information.

Todo: port for other types of calculations.

Specifies that the linear form of the regularized Poisson-Boltzmann equation (RPBE) should be solved. The regularized PBE equation replaces the point charge distribution with the corresponding Green's function. As a result of this replacement, the solution corresponds to the reaction field instead of the total potential; the total potential can be

recovered by adding the appropriate Coulombic terms to the solution. Likewise, this equation immediately yields the solvation energy without the need for reference calculations.

Note: The options *lpbe*, *npbe*, *lrpbe*, *nrbpe* are mutually exclusive.

maxsolve

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the number of times to perform the solve-estimate-refine iteration of the finite element solver (*fe-manual*). The syntax is:

```
maxsolve { num }
```

where *num* is an integer indicating the desired maximum number of iterations.

maxvert

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the maximum number of vertices to allow during solve-estimate-refine cycle of finite element solver (*fe-manual*). This places a limit on the memory that can be used by the solver. The syntax is:

```
maxvert { num }
```

where *num* is an integer indicating the maximum number of vertices.

npbe

Note: Some aspects of this command have been moved to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference:

See `FiniteDifference.equation()` for more information.

Todo: port for other types of calculations.

Specifies that the nonlinear (full) Poisson-Boltzmann equation should be solved.

Note: The options *lpbe*, *npbe*, *lrpbe*, *nrbpe* are mutually exclusive.

nrpbe

Note: Some aspects of this command have been moved to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference:

See `FiniteDifference.equation()` for more information.

Todo: port for other types of calculations.

Specifies that the nonlinear form of the regularized Poisson-Boltzmann equation (RPBE) should be solved. The regularized PBE equation replaces the point charge distribution with the corresponding Green's function. As a result of this replacement, the solution corresponds to the reaction field instead of the total potential; the total potential can be recovered by adding the appropriate Coulombic terms to the solution. Likewise, this equation immediately yields the solvation energy without the need for reference calculations.

Note: The options *lpbe*, *npbe*, *lrpbe*, *nrpbe* are mutually exclusive.

Note: This functionality is only available with FEM-based solvers.

srfm (elec)

Note: Some uses of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite differences; see `finite_difference.FiniteDifference()` for more information
-

Todo: port other versions of `srfm` to new syntax.

Specify the model used to construct the dielectric and ion-accessibility coefficients. The syntax for this command is:

```
srfm {flag}
```

where `flag` is a string describing the coefficient model:

mo1 The dielectric coefficient is defined based on a molecular surface definition. The problem domain is divided into two spaces. The “free volume” space is defined by the union of solvent-sized spheres (see *srad*) which do not overlap with biomolecular atoms. This free volume is assigned bulk solvent dielectric values. The complement of this space is assigned biomolecular dielectric values. With a non-zero solvent radius (*srad*), this choice of coefficient corresponds to the traditional definition used for PB calculations. When the solvent radius is set to zero, this corresponds to a van der Waals surface definition. The ion-accessibility coefficient is defined by an “inflated” van der Waals model. Specifically, the radius of each biomolecular atom is increased by the radius of the ion species (as specified with the *ion* keyword). The problem domain is then divided into two spaces. The space inside the union of these inflated atomic spheres is assigned an ion-accessibility value of 0; the complement space is assigned bulk ion accessibility values.

smo1 The dielectric and ion-accessibility coefficients are defined as for mol (see above). However, they are then “smoothed” by a 9-point harmonic averaging to somewhat reduce sensitivity to the grid setup as described by Bruccoleri et al. J Comput Chem 18 268-276, 1997 ([10.1007/s00214-007-0397-0](https://doi.org/10.1007/s00214-007-0397-0)).

sp12 The dielectric and ion-accessibility coefficients are defined by a cubic-spline surface as described by Im et al, Comp Phys Commun 111 (1-3) 59-75, 1998 ([10.1016/S0010-4655\(98\)00016-2](https://doi.org/10.1016/S0010-4655(98)00016-2)). The width of the dielectric interface is controlled by the *swin* parameter. These spline-based surface definitions are very stable with respect to grid parameters and therefore ideal for calculating forces. However, they require substantial reparameterization of the force field; interested users should consult Nina et al, Biophys Chem 78 (1-2) 89-96, 1999 ([10.1016/S0301-4622\(98\)00236-1](https://doi.org/10.1016/S0301-4622(98)00236-1)). Additionally, these surfaces can generate unphysical results with non-zero ionic strengths; this is an on-going area of development.

sp14 The dielectric and ion-accessibility coefficients are defined by a 7th order polynomial. This surface definition has characteristics similar to spl2, but provides higher order continuity necessary for stable force calculations with atomic multipole force fields (up to quadrupole).

swin

Note: Some instances of this keyword have been moved to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- For finite difference calculations, see `FiniteDifference.surface_spline_window()`

Todo: move other instances of this keyword to the new syntax

Specify the size of the support (i.e., the rate of change) for spline-based surface definitions (see *srfm (elec)*). The syntax is:

```
swin {win}
```

where *win* is a floating point number for the spline window width (in Å). Usually 0.3 Å.

Note that, per the analysis of Nina, Im, and Roux ([article](#)), the force field parameters (radii) generally need to be adjusted if the spline window is changed.

targetNum

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the target number of vertices in the initial finite element mesh for *fe-manual* calculations. The syntax is:

```
targetNum { num }
```

where *num* is an integer denoting the target number of vertices in initial mesh. Initial refinement will continue until this number is reached or the the longest edge of every simplex is below *targetRes*.

targetRes

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the target resolution of the simplices in a finite element mesh ([fe-manual](#)). The syntax is:

```
targetRes { res }
```

where `res` is a floating point number denoting the target resolution for longest edges of simplices in mesh (in Å). Refinement will continue until the longest edge of every simplex is below this value or the number of vertices reaches *targetNum*.

usemesh

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the external finite element mesh to be used in the finite element Poisson-Boltzmann calculation ([fe-manual](#)). These must have been input via an earlier READ mesh statement (see [READ input file section](#)). The syntax is:

```
usemesh {id}
```

where `id` is an integer ID specifying the particular map read in with [READ input file section](#). These IDs are assigned sequentially, starting from 1, and incremented independently for each mesh read by APBS.

write

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

This controls the output of scalar data calculated during the Poisson-Boltzmann run. This keyword can be repeated several times to provide various types of data output from APBS. The syntax is:

```
write {type} {format} {stem}
```

type A string indicating what type of data to output:

charge Write out the biomolecular charge distribution in units of e_c (electron charge) per Å³ (multigrid only).

pot Write out the electrostatic potential over the entire problem domain in units of $k_b T e_c^{-1}$ (multigrid and finite element), where

k_b Boltzmann's constant: $1.3806504 \times 10^{23} \text{ J K}^{-1}$

T The temperature of your calculation in K

e_c is the charge of an electron: $1.60217646 \times 10^{-19} \text{ C}$

As an example, if you ran your calculation at 300 K, then the potential would be written out as multiples of $k_b T e_c^{-1} = (1.3806504 \times 10^{23} \text{ J K}^{-1}) \times (300 \text{ K}) \times (1.60217646 \times 10^{-19} \text{ C})^{-1} = (4.1419512 \times 10^{-21} \text{ J}) \times (6.241509752 \times 10^{18} \text{ C}^{-1}) = 25.85202 \text{ mV}$

atompot Write out the electrostatic potential at each atom location in units of $k_b T e_c^{-1}$ (multigrid and finite element).

- smol** Write out the solvent accessibility defined by the molecular surface definition (see *srfm (elec) smol*). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)
- sspl** Write out the spline-based solvent accessibility (see *srfm (elec) spl2*). Values are unitless and range from 0 (inaccessible) to 1 (accessible) (multigrid and finite element)
- vdw** Write out the van der Waals-based solvent accessibility (see *srfm (elec) smol* with *srad* 0.0). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)
- ivdw** Write out the inflated van der Waals-based ion accessibility (see *srfm (elec) smol*). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)
- lap** Write out the Laplacian of the potential $\nabla^2\phi$ in units of $k_B T e_c^{-1} \text{\AA}^{-2}$ (multigrid only).
- edens** Write out the “energy density” $-\nabla \cdot \epsilon \nabla \phi$ in units of $k_B T e_c^{-1} \text{\AA}^{-2}$ (multigrid only).
- ndens** Write out the total mobile ion number density for all ion species in units of M (multigrid only). The output is calculated according to the formula (for nonlinear PB calculations): $\rho(x) = \sum_i^N \bar{\rho}_i e^{-q_i \phi(x) - V_i(x)}$, where N is the number of ion species, $\bar{\rho}_i$ is the bulk density of ion species i , q_i is the charge of ion species i , $\phi(x)$ is the electrostatic potential, and V_i is the solute-ion interaction potential for species i .
- qdens** Write out the total mobile ion charge density for all ion species in units of $e_c M$ (multigrid only). The output is calculated according to the formula (for nonlinear PB calculations): $\rho(x) = \sum_i^N \bar{\rho}_i q_i e^{-q_i \phi(x) - V_i(x)}$, where N is the number of ion species, $\bar{\rho}_i$ is the bulk density of ion species i , q_i is the charge of ion species i , $\phi(x)$ is the electrostatic potential, and V_i is the solute-ion interaction potential for species i .
- dielx or diely or dielz** Write out the dielectric map shifted by 1/2 grid spacing in the {x, y, z}-direction (see *READ input file section diel*). The values are unitless (multigrid only).
- format** A string that specifies the format for writing out the data:
- dx** Write out data in *OpenDX scalar data format*. This is the preferred format for APBS I/O. (multigrid and finite element).
 - avs** Write out data in AVS UCD format. (finite element only).
 - uhbd** Write out data in *UHBD scalar data format*. (multigrid only).
 - gz** Write out *OpenDX scalar data format* in gzipped (zlib) compatible format. Appends .dx.gz to the filename.
 - flat** Write out data as a plain text file. (multigrid and finite element).
- stem** A string that specifies the path for the output; files are written to *stem.XYZ*, where XYZ is determined by the file format (and processor rank for parallel calculations). If the pathname contains spaces, then it must be surrounded by double quotes.

Note: The finite element methods are currently most useful for a select set of problems which can benefit from adaptive refinement of the solution. Furthermore, this implementation is experimental. In general, the sequential and parallel focusing multigrid methods offer the most efficient solution of the PBE for most systems.

geoflow-auto

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

To increase the accuracy of our implicit solvent modeling, we have implemented a differential geometry based geometric flow solvation model (Thomas, 2013). In this model, polar and nonpolar solvation free energies are coupled and the

solvent-solute boundary is determined in a self-consistent manner. Relevant references are provided in [Recommended reading](#). This section provides a brief overview of the method.

The solutions for the electrostatic potential ϕ and the characteristic function S (related to the solvent density) are obtained by minimizing a free energy functional that includes both polar and nonpolar solvation energy terms. Minimization of the functional with respect to ϕ gives the Poisson-Boltzmann equation with a dielectric coefficient ϵ has the solute value ϵ_m where $S = 1$ and the solvent value ϵ_s where $S = 0$. Minimization of the free energy functional with respect to S gives

$$-\nabla \cdot \left(\gamma \frac{\nabla S}{\|\nabla S\|} \right) + p - \rho_0 U^{att} + \rho_m \phi - \frac{1}{2} \epsilon_m |\nabla \phi|^2 + \frac{1}{2} \epsilon_s |\nabla \phi|^2 = 0$$

where γ is the microscopic surface tension, p is the hydrostatic pressure, and U^{att} is the attractive portion of the van der Waals dispersion interaction between the solute and the solvent.

Keywords for this calculation type include:

bconc

Note: Some versions of this command have been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)):

- **Nonpolar calculations:** See `Nonpolar.solvent_density()` for more information.

This keyword specifies the bulk solvent density. This coefficient multiplies the integral term of the apolar model discussed above and can be set to zero to eliminate integral contributions to the apolar solvation calculation. The syntax is:

```
bconc <density>
```

where `density` is a floating point number giving the bulk solvent density in \AA^{-3} .

gamma

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

This keyword specifies the surface tension coefficient for apolar solvation models.

```
gamma { value }
```

where `value` is a floating point number designating the surface tension in units of $\text{kcal mol}^{-1} \text{\AA}^{-2}$. This term can be set to zero to eliminate the SASA (solvent-accessible surface area) contributions to the apolar solvation calculations.

Warning: *Either this documentation is incorrect or the implementation needs to be changed to use $\text{kJ mol}^{-1} \text{\AA}^{-2}$ instead of kcal.*

Todo: Resolve unit confusion with geometric flow *gamma* keyword. <https://github.com/Electrostatics/apbs/issues/490>

press

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This term specifies the solvent pressure in $\text{kJ mol}^{-1} \text{\AA}^{-3}$. This coefficient multiplies the volume term of the apolar model and can be set to zero to eliminate volume contributions to the apolar solvation calculation. The syntax is:

```
press {value}
```

where `value` is the floating point value of the pressure coefficient in $\text{kJ mol}^{-1} \text{\AA}^{-3}$.

Warning: *Either* this documentation is incorrect *or* the implementation needs to be changed to use $\text{kJ mol}^{-1} \text{\AA}^{-3}$ instead of kcal.

Todo: Resolve unit confusion with geometric flow `press` keyword and the apolar *press* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/499>

vdwdisp

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify whether the attractive van der Waals contribution to the geometric flow potential is on or off.

```
vdwdisp { flag }
```

where `flag` is 0 (vdw off) or 1 (vdw on).

Warning: Although the `ion` and `lpbe` keywords will be accepted in the `geoflow-auto` calculation, the treatment of salt is not currently implemented in APBS geometric flow.

Todo: Add LPBE/NPBE support to geometric flow or remove the `ion` and `lpbe` keywords. Documented in <https://github.com/Electrostatics/apbs/issues/491>

Todo: If there's only one mode, then we can change the keyword from `geoflow-auto` to just `geoflow`. Documented in <https://github.com/Electrostatics/apbs/issues/492>

mg-auto

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*). See `FiniteDifference` and `Focus` for more information.

Automatically configured finite difference Poisson-Boltzmann calculations.

This multigrid calculation automatically sets up and performs a string of single-point PBE calculations to “focus” on a region of interest (binding site, etc.) in a system. It is basically an automated version of *mg-manual* designed for easier use. Most users should use this version of ELEC.

Focusing is a method for solving the Poisson-Boltzmann equation in a finite difference setting. Some of the earliest references to this method are from Gilson and Honig¹. The method starts by solving the equation on a coarse grid (i.e., few grid points) with large dimensions (i.e., grid lengths). The solution on this coarse grid is then used to set the Dirichlet boundary condition values for a smaller problem domain – and therefore a finer grid – surrounding the region of interest. The finer grid spacing in the smaller problem domain often provides greater accuracy in the solution.

The following keywords are present in mg-auto ELEC blocks; all keywords are required unless otherwise noted.

Note: During focusing calculations, you may encounter the message “WARNING! Unusually large potential values detected on the focusing boundary!” for some highly charged systems based on location of the focusing boundary. First, you should determine if you received any other warning or error messages as part of this calculation, particularly those referring to exceeded number of iterations or error tolerance (*etol*). Next, you should check if the calculation converged to a reasonable answer. In particular, you should check sensitivity to the grid spacing by making small changes to the grid lengths (via the *fglen* parameter) and see if the changes in energies are correspondingly small. If so, then this warning can be safely ignored.

cgcent

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Focus`. `coarse_grid_center()` for more information.

This keyword controls electrostatic energy output from a Poisson-Boltzmann calculation. The syntax is:

```
cgcent { mol id | xcent ycent zcent }
```

The arguments for this keyword are **either**

mol id Center the grid on molecule with integer ID *id*; as assigned in the `READ` section with a `READ mol` command (see *READ input file section*)

or

xcent ycent zcent Center the grid on the (floating point) coordinates (in Å) at which the grid is centered. Based on the PDB coordinate frame.

cglen

¹ Gilson MK and Honig BH, Calculation of electrostatic potentials in an enzyme active site. *Nature*, 1987. 330(6143): p. 84-6. DOI:10.1038/330084a0

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Focus.coarse_grid_dimensions()` for more information.

Specify the length of the coarse grid (in a focusing calculation) for an automatic multigrid (*mg-auto*, *mg-para*) Poisson-Boltzmann calculation. This may be different in each direction.

```
cglen {xlen ylen zlen}
```

This is the starting mesh, so it should be large enough to completely enclose the biomolecule and ensure that the chosen boundary condition (see *bcbf*) is appropriate.

xlen ylen zlen Grid lengths (floating point numbers) in the x-, y-, and z-directions in Å.

dime

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- For a focused finite difference calculation, see `Focus.coarse_grid_dimensions()` for more information.
-

Todo: add manual finite difference documentation.

Specifies the number of grid points per processor for grid-based discretization. The syntax is:

```
dime {nx ny nz}
```

For *mg-manual* calculations, the arguments are dependent on the choice of *nlev* by the formula: $n = c2^{l+1} + 1$ where n is the *dime* argument, c is a non-zero integer, l is the *nlev* value. The most common values for grid dimensions are 65, 97, 129, and 161 (they can be different in each direction); these are all compatible with a *nlev* value of 4. If you happen to pick a “bad” value for the dimensions (i.e., mismatch with *nlev*), the APBS code will adjust the specified *dime* downwards to more appropriate values. This means that “bad” values will typically result in lower resolution/accuracy calculations! The arguments for this keyword are:

nx ny nz The (integer) number of grid points in the x-, y-, and z-directions, respectively.

Note: *dime* should be interpreted as the number of grid points per processor for all calculations, including *mg-para*. This interpretation helps manage the amount of memory per-processor - generally the limiting resource for most calculations.

fgcent

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Focus.fine_grid_center()` for more information.

Specify the center of the fine grid (in a focusing calculation) based on a molecule’s center or absolute coordinates for *mg-para* and *mg-auto* multigrid calculations. The syntax is:

where a user can specify **either**

mol {id} Center the grid on molecule with integer ID *id*; as assigned in the READ section (see *READ input file section*) of the input file. Molecule IDs are assigned in the order they are read, starting at 1.

or the user can specify

xcent ycent zcent Center the grids on the coordinates (floating point numbers in Å) at which the grid is centered. Based on the input molecule PDB coordinate frame.

fglen

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Focus.fine_grid_dimensions()` for more information.

Specifies the fine mesh domain lengths in a multigrid focusing calculation (*mg-para* or *mg-auto*); this may be different in each direction. The syntax is:

```
fglen {xlen ylen zlen}
```

This should enclose the region of interest in the molecule. The arguments to this command are:

xlen ylen zlen Grid lengths (floating point numbers) in the x-, y-, and z-directions in Å.

usemap

Note:

Some instances of this keyword have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- Finite difference calculations: see `UseMap`

Todo: Port other uses to new syntax.

Specify pre-calculated coefficient maps to be used in the Poisson-Boltzmann calculation. These must have been input via an earlier READ statement (see *READ input file section*).

The syntax for this command is:

```
usemap {type} {id}
```

where the mandatory keywords are:

type A string that specifies the type of pre-calculated map to be read in:

diel Dielectric function map (as read by *READ input file section* *diel*); this causes the *pdie*, *sdie*, *srad*, *swin*, and *srfm (elec)* parameters and the radii of the biomolecular atoms to be ignored when computing dielectric maps for the Poisson-Boltzmann equation. Note that the *pdie* and *sdie* values are still used for some boundary condition calculations as specified by *bcfl*.

kappa Mobile ion-accessibility function map (as read by *READ input file section* kappa); this causes the *swin* and *srfm (elec)* parameters and the radii of the biomolecular atoms to be ignored when computing mobile ion values for the Poisson-Boltzmann equation. The *ion* parameter is not ignored and will still be used.

charge Charge distribution map (as read by *READ input file section* charge); this causes the *chgm* parameter and the charges of the biomolecular atoms to be ignored when assembling the fixed charge distribution for the Poisson-Boltzmann equation.

pot Potential map (as read by *READ input file section* pot); this option requires setting *bconf* to map.

id As described in the READ command documentation (see *READ input file section*), this integer ID specifies the particular map read in with READ. These IDs are assigned sequentially, starting from 1, and incremented independently for each map type read by APBS. In other words, a calculation that uses two PQR files, one parameter file, three charge maps, and four dielectric maps would have PQR files with IDs 1-2, a parameter file with ID 1, charge maps with IDs 1-3, and dielectric maps with IDs 1-4.

writemat

Note: This command is deprecated and will not be ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This controls the output of the mathematical operators in the Poisson-Boltzmann equation as matrices in Harwell-Boeing matrix format (multigrid only). The syntax is:

```
writemat {type} {stem}
```

where

type A string that indicates what type of operator to output.

poisson Write out the Poisson operator $-\nabla \cdot \epsilon \nabla$.

stem A string that specifies the path for the output.

mg-manual

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*); see Manual and `FiniteDifference.boundary_condition()`.

Manually-configured finite difference multigrid Poisson-Boltzmann calculations.

This is a standard single-point multigrid PBE calculation without focusing or additional refinement. The `mg-manual` calculation offers the most control of parameters to the user. Several of these calculations can be strung together to perform focusing calculations by judicious choice of the *bconf* flag; however, the setup of the focusing is not automated as it is in *mg-auto* and *mg-para* calculations and therefore this command should primarily be used by more experienced users.

gcent

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the center of the grid based on a molecule's center or absolute coordinates *mg-manual* multigrid calculations. The syntax is:

```
fgcent { mol id | xcent ycent zcent }
```

where a user can specify **either**

mol {id} Center the grid on molecule with integer ID *id*; as assigned in the READ section (see *READ input file section*) of the input file. Molecule IDs are assigned in the order they are read, starting at 1.

or the user can specify

xcent ycent zcent Center the grids on the coordinates (floating point numbers in Å) at which the grid is centered. Based on the input molecule PDB coordinate frame.

glen

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the mesh domain lengths for multigrid *mg-manual* calculations. These lengths may be different in each direction. The syntax is:

```
glen {xlen ylen zlen}
```

where `xlen ylen zlen` are the (floating point) grid lengths in the x-, y-, and z-directions (respectively) in Å.

grid

Note: Some versions of this command have been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*):

- **Nonpolar calculations:** See `Nonpolar.grid_spacings()` for more information.
-

Specify the grid spacings for multigrid and volume integral calculations. This value may be different in each direction. The syntax is:

```
grid {hx hy hz}
```

where `hx hy hz` are the (floating point) grid spacings in the x-, y-, and z-directions (respectively) in Å.

nlev

Note: `..currentmodule:: apbs.input_file.calculate.finite_difference`

This command has been eliminated in the *new APBS syntax* (see *YAML- and JSON-format input files*); see `GridDimensions` for more information.

Specify the depth of the multilevel hierarchy used in the *mg-manual* multigrid solver. See *dime* for a discussion of how *nlev* relates to grid dimensions. The syntax is:

```
nlev {lev}
```

where `lev` is an integer indicating the desired depth of the multigrid hierarchy.

mg-para

Note: This command has been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)); see `Focus.parallel()` and `FiniteDifference.calculation_type()`.

Automatically-configured parallel focusing multigrid Poisson-Boltzmann calculations.

This calculation closely resembles [mg-auto](#) in syntax. However, it is designed to perform electrostatics calculations on systems in a parallel focusing fashion.

async

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

An optional keyword to perform an asynchronous parallel focusing Poisson-Boltzmann equation. The syntax is

```
async {rank}
```

where `rank` is the integer ID of the particular processor to masquerade as. Processor IDs range from 0 to $N-1$, where N is the total number of processors in the run (see [pdime](#)). Processor IDs are related to their position in the overall grid by $p = nxnyk + nxj + i$ where nx is the number of processors in the x-direction, ny is the number of processors in the y-direction, nz is the number of processors in the z-direction, i is the index of the processor in the x-direction, j is the index of the processor in the y-direction, k is the index of the processor in the z-direction, and p is the overall rank of the processor.

ofrac

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the amount of overlap to include between the individual processors meshes in a parallel focusing calculation ([mg-para](#)). The syntax is:

```
ofrac {frac}
```

where `frac` is a floating point value between 0.0 and 1.0 denoting the amount of overlap between processors. Empirical evidence suggests that an value of 0.1 is sufficient to generate stable energies. However, this value may not be sufficient to generate stable forces and/or good quality isocontours. For example, the following table illustrates the change in energies and visual artifacts in isocontours as a function of `ofrac` values for a small peptide (2PHK:B).

Table 1: Sensitivity of 2PHK:B solvation energy calculations to ofrac values.

ofrac value	Energy (kJ/mol)	Visual artifact in isocontour?
0.05	342.79	No
0.06	342.00	No
0.07	341.12	Yes
0.08	341.14	Yes
0.09	342.02	Yes
0.10	340.84	Yes
0.11	339.67	No
0.12	341.10	No
0.13	341.10	No
0.14	341.32	No
0.15	341.54	No

In general, larger ofrac values will reduce the parallel efficiency but will improve the accuracy.

For broad spatial support of the splines, every charge included in partition needs to be at least 1 grid space (*chgm* sp10), 2 grid spaces (*chgm* sp12), or 3 grid spaces (*chgm* sp14) away from the partition boundary.

pdime

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the processor array to be used in a parallel focusing (*mg-para*) calculation. The syntax is:

```
pdime {npx npy npz}
```

where np_x np_y np_z are the integer number of processors to be used in the x-, y- and z-directions of the system. The product np_x × np_y × np_z should be less than or equal to the total number of processors with which APBS was invoked (usually via mpirun). If more processors are provided at invocation than actually used during the run, the extra processors are not used in the calculation. The processors are tiled across the domain in a Cartesian fashion with a specified amount of overlap (see *ofrac*) between each processor to ensure continuity of the solution. Each processor's subdomain will contain the number of grid points specified by the dime keyword. For broad spatial support of the splines, every charge included in partition needs to be at least 1 grid space (*chgm* sp10), 2 grid spaces (*chgm* sp12), or 3 grid spaces (*chgm* sp14) away from the partition boundary.

mg-dummy

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*); see `FiniteDifference.noop()` for more information.

Not a Poisson-Boltzmann calculation. Many calculations of surface and charge distribution properties which do not require solution of the PBE.

This type of calculation allows users to write out dielectric, ion-accessibility, and charge distribution, and other types of maps that depend solely on biomolecular geometry. Since these maps depend only on geometry, they can be written out without actually solving the PB equation.

pbam-auto

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

PB-AM is an analytical solution to the linearized Poisson-Boltzmann equation for multiple spherical objects of arbitrary charge distribution in an ionic solution. More details on the method are available in [Lotan, Head-Gordon \(2006\)](#). The physical calculations are used to perform various actions on a system of molecules such as calculation of energies, forces, torques, electrostatic potentials, and Brownian dynamics schemes. This fast method coarse-grains all molecules of the system into single spheres large enough to contain all molecule atoms.

Todo: If there's only one mode to PBAM, let's call it `pbam` instead of `pbam-auto`. Documented in <https://github.com/Electrostatics/apbs/issues/498>

The current implementation of PB-AM in APBS includes:

- Calculation of energies, forces and torques
- Calculation of electrostatic potentials
- Brownian dynamics simulations

Keywords for this calculation type include:

3dmap

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the name of the file into which the potential surface on the coarse-grain molecule surface will be printed.

```
3dmap {filename}
```

where `filename` is a string for the name of the file where a 3D grid will be printed out.

Todo: The PB-(S)AM `3dmap` keyword should not exist; please replace it ASAP with the *write* command. Documented this todo as <https://github.com/Electrostatics/apbs/issues/482>

diff

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the diffusion coefficients for each molecule in the system for a PB-(S)AM Brownian dynamics calculation.

```
diff {type} {dTrans} {dRot}
```

type a string indicating the molecule dynamics type

stat Stationary.

rot Object is fixed but rotates

move Object moves and rotates.

dTrans Translational diffusion coefficient in units of $\text{\AA}^2 \text{ps}^{-1}$. Used only with the `move` keyword.

dRot Rotational diffusion coefficient. Used with the `move` and `rot` keywords.

Todo: What are the units for `dRot`? Documented as <https://github.com/Electrostatics/apbs/issues/486>

Note: The order of these keywords is expected to be identical to the order of the molecules in the `READ` section.

Todo: Add a `mol id` flag rather than have an implicit ordering of the `diff` keywords. Documented in <https://github.com/Electrostatics/apbs/issues/487>

dx

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the name of the file into which the potential will be printed.

```
dx {filename}
```

where `filename` is a string for the name of the file where an OpenDX file will be printed out.

Todo: The PB-(S)AM `dx` keyword should not exist; please replace it ASAP with the *write* command. Documented in <https://github.com/Electrostatics/apbs/issues/488>

grid2d

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the filename and location of a 2D cross sectional potential to be written to.

```
grid2d {filename} {axis} {axis_value}
```

filename String for the name of the 2D grid to be printed out

axis String of either x, y, or z, for which cartesian axis the grid will be computed along

axis_value A floating point number of the position along `axis` that will be used.

Note: Multiple 2D files can be printed out with 1 PB-AM run. Just specify them with more `grid2d` flags.

Todo: The PB-(S)AM `grid2d` keyword should not exist; please replace it ASAP with the *write* command. Documented in <https://github.com/Electrostatics/apbs/issues/493>

gridpts

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the number of gridpoints in each cartesian dimension.

```
gridpts {pts}
```

where `pts` is a integer number indicating the number of grid points.

Todo: The PB-(S)AM `gridpts` keyword should not exist; it's duplicative of the existing *dime* keyword! Documented in <https://github.com/Electrostatics/apbs/issues/494>

ntraj

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the number of Brownian Dynamic trajectories desired for the PB-(S)AM run.

```
ntraj {traj}
```

where `traj` is an integer of the number of desired trajectories.

pbc

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This keyword is used to indicate if 3D periodic boundary conditions (PBCs) will be used in a PB-(S)AM calculation. If used, a box length must also be specified, in Ångstroms.

```
pbc {boxlength}
```

where `boxlength` is the floating point value of the box length in Ångstroms.

Note: The box is centered at the origin (0, 0, 0). The code assumes a minimum image convention, so it only includes the closest image of the neighboring molecules. For this convention to always be preserved, the periodic box is assumed to be large enough such that the electrostatic forces are sufficiently attenuated beyond one boxlength. Generally, the program assumes a mutual polarization cutoff of 100 Å for the mutual polarization, so if the boxlength is shorter, the cutoff will be reduced to boxlength/2.

randorient

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Flag to indicate that the molecules should have a random orientation in subsequent PB-(S)AM calculations.

runname

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Specify the output name for the PB-(S)AM calculation.

```
..code-block:: bash
```

```
    runname {name}
```

where `name` is a string indicating the prefix for all PB-(S)AM output files.

runtype

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

Indicate what type of calculation you would like to run with the PB-(S)AM model.

```
runtype {type}
```

where `type` is the type of calculation to be performed:

energyforce Compute and print out the interaction energies, forces and torques on each molecule.

electrostatics Print the electrostatic potential of points in the system.

dynamics Perform a Brownian Dynamics simulation, using forces and torques generated from the PB-(S)AM model. The calculation of force and torque has been integrated into a Brownian dynamics scheme that is detailed in [Yap EH, Head-Gordon TL \(2013\)](#) This option will generate a series of files of the form

dyn_toy.pqr The starting configuration of the system for the first trajectory

dyn_toy.stat A file that prints how each trajectory was terminated and the time that this occurred at.

dyn_toy_traj.xyz A VMD-readable xyz file for the trajectory of `traj`.

dyn_toy_traj.dat A file with positions, forces and torques for the system.

Todo: The dynamics part of the PB-(S)AM code should be moved out of the `ELEC` section. Documented in <https://github.com/Electrostatics/apbs/issues/500>

salt

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the monovalent salt concentration of the system, in molar. This is usually a value between 0.00 to 0.15.

```
salt {saltConc}
```

where `saltConc` is the floating point value of the monovalent salt concentration in molar.

Todo: The PB-(S)AM `salt` keyword should be eradicated and replaced with the *ion* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/501>

term

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify a termination condition for a PB-(S)AM Brownian dynamics trajectory. The syntax is:

```
term {type} {options}
```

where the `options` are determined by the `type` as follows:

contact {file} Termination based on molecular contact conditions. `file` is a string for the contact file filename. The contact file has a list formatted as follows: `moltype1 at1 moltype2 at2 dist` where `moltype1` and `moltype2` are indices of the molecular types, `at1` is the index of an atom from the first molecular type, `at2` is the index of an atom from the second molecular type, and `dist` is the maximum distance between the two atoms that defines the contact. `pad` is distance criterion that will be checked in the case that the true atom contact distance may not be fulfilled.

Note: Sometimes these distances cannot be reached due to the assumption in this model that the molecule is spherical. If this is the case, the atom positions are transformed to the molecule surface and surface points are compared to the `pad` distance.

{pos} {val} {molecule} Specify a position termination condition for a given molecule. where `pos` is one of the following options: `x<=`, `x>=`, `y<=`, `y>=`, `z<=`, `z>=`, `r<=`, `r>=`. `val` is the value along the given axis to check against. `molecule` is the molecule index (1 based) according to the order of molecules listed in the `READ` section that this condition applies to. This command can be understood as: “Terminate the simulation when molecule `molecule` fulfills the condition `pos val`”.

Todo: Add a constant keyword (e.g., like `position`) before the `{pos}` argument of `term`. Documented in <https://github.com/Electrostatics/apbs/issues/503>

time {val} Specify a time termination condition where `val` is a floating point number for the trajectory time limit (in picoseconds).

termcombine

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Combine multiple PB-(S)AM Brownian dynamics trajectory termination conditions (see [term](#)) via logic operators

```
termcombine {op}
```

where `op` is either the string `or` or `and`. If `and` is selected, all listed termination conditions must be fulfilled before the simulation ends. If `or` is selected, only one needs to be satisfied to complete the simulation.

units

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

Specify the units for energy/force/potential output in PB-(S)AM calculations:

```
units {flag}
```

where `flag` specifies the unit system:

kcalmol kcal/mol

jmol J/mol

kT kT

Force units will be energy units/Angstrom and potential units will be energy units/electron.

Todo: It would be great to use the same units everywhere in APBS. Documented in <https://github.com/Electrostatics/apbs/issues/485>

xyz

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

For each molecule in the system and for each trajectory, specify a xyz file for the starting position of that molecule. The syntax is:

```
xyz {molecule_id} {filename}
```

molecule_id An integer (starting at 1) of the molecule index from the READ section

filename The name of the file for the xyz coordinates of the molecule center for a given trajectory. The trajectories for a given molecule should be ordered sequentially in the ELEC section.

Todo: It would be nice to incorporate the `xyz` functionality into the *READ input file section* block. Documented in <https://github.com/Electrostatics/apbs/issues/505>

Background information

PB-AM is an analytical solution to the linearized Poisson-Boltzmann equation for multiple spherical objects of arbitrary charge distribution in an ionic solution. The solution can be reduced to a simple system of equations as follows:

$$A = \Gamma \cdot (\Delta \cdot T \cdot A + E)$$

Where $A^{(i)}$ represents the effective multipole expansion of the charge distributions of molecule i . $E^{(i)}$ is the free charge distribution of molecule i . Γ is a dielectric boundary-crossing operator, Δ is a cavity polarization operator, T an operator that transforms the multipole expansion to a local coordinate frame. $A^{(i)}$ is solved for through an iterative SCF method.

From the above formulation, computation of the interaction energy $\Omega^{(i)}$ for molecule i , is given as follows:

$$\Omega^{(i)} = \frac{1}{\epsilon_s} \left\langle \sum_{j \neq i}^N T \cdot A^{(j)}, A^{(i)} \right\rangle$$

where $\langle M, N \rangle$ denotes the inner product. Forces can be obtained from

$$\mathbf{F}^{(i)} = \nabla_i \Omega^{(i)} = \frac{1}{\epsilon_s} \left[\langle \nabla_i T \cdot A^{(i)}, A^{(i)} \rangle + \langle T \cdot A^{(i)}, \nabla_i A^{(i)} \rangle \right]$$

pbsam-auto

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

PB-SAM is a semi-analytical solution to the linearized Poisson-Boltzmann equation for multiple molecules of arbitrary charge distribution in an ionic solution. The solution is an extension of the [analytical method](#), leveraging fast-multipole methods as well as boundary elements. Each molecule is coarse-grained as a system of overlapping spheres, whose surface charges are represented by multipole expansions. For details on the method, please see [Yap, Head-Gordon \(2010\)](#) and [Yap, Head-Gordon \(2013\)](#).

Todo: If there's only one mode to PBAM, let's call it `pbsam` instead of `pbsam-auto`.

The current implementation of PB-SAM in APBS includes:

- Calculation of energies, forces and torques
- Calculation of electrostatic potentials
- Brownian dynamics simulations

Keywords for this calculation type include:

exp

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

This keyword can be used to load in the expansion matrices from files. They will have been previously generated, and will be named `mol.m.H, F.[s].exp` (see [pbam-auto](#) for more information). The syntax is:

```
exp {prefix}
```

where `prefix` is the filename prefix `molmsph`. The `H` or `F` and `s.bin` will be appended during the program run.

Todo: It would be better to generalize the *READ input file section* of the input file rather than use the `exp` command. This command also needs to be cleaned up – it’s too fragile. Documented at <https://github.com/Electrostatics/apbs/issues/489>

imat

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This keyword can be used to load in the surface integral matrices previously generated by PB-SAM named as `molmsphs.bin` for molecule ID `s` and coarse-grained sphere `s` (see *pbam-auto* for more information). The syntax is:

```
imat {prefix}
```

where `prefix` is the filename prefix `molmsph`. The `s.bin` will be appended during the program run.

Todo: It would be better to generalize the *READ input file section* of the input file rather than use the `imat` command. This command also needs to be cleaned up – it’s too fragile. Documented in <https://github.com/Electrostatics/apbs/issues/495>

surf

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This keyword can be used to load in the MSMS vertex file for coarse-graining (see *pbsam-auto*) The syntax is:

```
surf {prefix}
```

where `prefix` refers to the filename `:file:{prefix}.vert`.

Todo: The PB-SAM `surf` command is redundant with and should be replaced by the existing *usemesh* command. Documented in <https://github.com/Electrostatics/apbs/issues/502>

tolsp

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

This is an undocumented parameter from the PB-SAM code that does something with the coarseness of the molecule representation. The PB-SAM authors recommend a value of 2.5.

Todo: We need documentation for `tolsp`. Documented in <https://github.com/Electrostatics/apbs/issues/504>

Background information

PB-SAM is a semi-analytical solution to the linearized Poisson-Boltzmann equation for multiple molecules of arbitrary charge distribution in an ionic solution. The solution is an extension of the analytical method, leveraging Fast-Multipole methods as well as boundary elements. Each molecule is coarse-grained as a system of overlapping spheres, whose surface charges are represented by the multipole expansions $H^{(i)}$ and $F^{(i)}$. To solve for the potential, the following interactions are considered:

- Intra-molecular interactions between overlapping spheres are treated numerically
- Intra-molecular interactions between non-overlapping spheres are treated analytically
- Inter-molecular interactions between spheres on different molecules

With these interactions, the multipole expansions are solved with an iterative SCF method, briefly given as

$$\begin{aligned} H^{(i,k)} &= I_E^{(i,k)} \cdot \left(H^{(i,k)} + F^{(i,k)} + T \cdot H^{(j,l)} \right) \\ F^{(i,k)} &= I_E^{(i,k)} \cdot \left(H^{(i,k)} + F^{(i,k)} + T \cdot F^{(j,l)} \right) \end{aligned}$$

Where $H^{(i)}$ and $F^{(i)}$ are multipole expansions, $I_E^{(i,k)}$ is the exposed surface integral matrix for sphere k of molecule i , and T is an operator that transforms the multipole expansion to a local coordinate frame.

From the above formulation, computation of the interaction energy $\Omega^{(i)}$ for molecule i , is given as a sum of all the interactions of spheres k within it with all external spheres (in a simplified form) as follows:

$$\Omega^{(i)} = \frac{1}{\epsilon_s} \left\langle \sum_{k \text{ in } i} \sum_{j \neq i}^N \sum_{l \text{ in } j} T \cdot H^{(j,l)}, H^{(i,k)} \right\rangle$$

where $\langle M, N \rangle$ denotes the inner product.

When energy is computed, forces follow as:

$$\mathbf{F}^{(i)} = \nabla_i \Omega^{(i)} = \frac{1}{\epsilon_s} [\langle \nabla_i T \cdot H^{(j,l)}, H^{(i,k)} \rangle + \langle T \cdot H^{(j,l)}, \nabla_i H^{(i,k)} \rangle]$$

The method to calculate the torque is discussed in [Yap, Head-Gordon \(2013\)](#).

PB-SAM files

Vertex/surface file

As part of the coarse-graining process a definition of the molecular surface is necessary.

Coarse-grained PQR file

The coarse-graining process will produce a new PQR file `mol #_cg.pqr` that contains the original PQR concatenated with coarse-graining spherical centers. The number `#` refers to the order the file was read during the *READ input file section* statements.

IMAT: surface integral file

The surface integrals are computed for the boundary element part of PB-SAM. Their calculation can be quite time-consuming, so the first time they are computed for a system, they are saved to the working directory with the name `molmsphs.bin``. The *m* in `molmsphs.bin`` is the ordered ID of the molecule from the PQR section. The *s* in `molmsphs.bin`` refers to coarse-grained sphere *s* of the molecule.

Multipole expansion files

Much like the IMAT files, the expansion files are files generated from self-polarization that are useful and time-saving methods for running a system of full-mutual polarization on many molecules. If no expansion path is provided, the program will calculate self-polarization for each type of molecule in the system and save files of the form `molmH, F.s.exp`, where *m* is the molecule ID, *H* and *F* refer to the respective expansion (see above), and *s* is the coarse-grained sphere number.

APOLAR input file section

Note: This section has been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)). See [Nonpolar grid-based calculations](#) for more information.

This section is the main component for apolar solvation calculations in APBS runs. There may be several APOLAR sections, operating on different molecules or using different parameters for multiple runs on the same molecule. The syntax of this section is:

```
APOLAR [name id]
  <keywords...>
END
```

The first (optional) argument is:

```
name <id>
```

where *id* is a unique string which can be assigned to the calculation to facilitate later operations (particularly in the [PRINT input file section](#) statements). The `keywords...` describing the parameters of the apolar calculation are discussed in more detail below:

dpos

Note: This command has been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)): see `Nonpolar.displacement()` for more information.

This is the displacement used for finite-difference-based calculations of surface area derivatives. I know, this is a terrible way to calculate surface area derivatives – we’re working on replacing it with an analytic version. In the meantime, please use this parameter with caution. If anyone has code for a better method, please share!

The syntax is:

```
dpos {displacement}
```

where `displacement` is a floating point number indicating the finite difference displacement for force (surface area derivative) calculations in units of Å.

Warning: This parameter is very dependent on `sdens` (see *sdens*); e.g., smaller values of `dpos` require larger values of `sdens`.

gamma

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Nonpolar.surface_tension()` for more information.

This keyword specifies the surface tension coefficient for apolar solvation models.

```
gamma { value }
```

where `value` is a floating point number designating the surface tension in units of $\text{kJ mol}^{-1} \text{Å}^{-2}$. This term can be set to zero to eliminate the SASA contributions to the apolar solvation calculations.

Todo: Resolve unit confusion with geometric flow *gamma* keyword. <https://github.com/Electrostatics/apbs/issues/490>

press

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Nonpolar.pressure()` for more information.

This term specifies the solvent pressure in $\text{kJ mol}^{-1} \text{Å}^{-3}$. This coefficient multiplies the volume term of the apolar model and can be set to zero to eliminate volume contributions to the apolar solvation calculation. The syntax is:

```
press {value}
```

where `value` is the floating point value of the pressure coefficient in $\text{kJ mol}^{-1} \text{Å}^{-3}$.

Todo: Resolve unit confusion with geometric flow *press* keyword and the apolar *press* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/499>

srfm (apolar)

Note: This command has been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*): see `Nonpolar.surface_method()` for more information.

This keyword specifies the model used to construct the solvent-related surface and volume. The syntax is:


```
srfm {flag}
```

where `flag` is a string that specifies the model used for surface and volume. Acceptable values of `flag` include:

sacc Solvent-accessible (also called “probe-inflated”) surface and volume.

APBS apolar calculations follow the very generic framework described in Wagoner JA, Baker NA. Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms. *Proc Natl Acad Sci USA*, 103, 8331-8336, 2006. doi:[10.1073/pnas.0600118103](https://doi.org/10.1073/pnas.0600118103).

Nonpolar solvation potentials of mean force (energies) are calculated according to:

$$W^{(\text{np})}(x) = \gamma A(x) + pV(x) + \bar{\rho} \sum_{i=1}^N \int_{\Omega} u_i^{(\text{att})}(x_i, y) \theta(x, y) dy$$

and mean nonpolar solvation forces are calculated according to:

$$\mathbf{F}_i^{(\text{np})}(x) = -\gamma \frac{\partial A(x)}{\partial x_i} - p \int_{\Gamma_i(x)} \frac{y - x_i}{\|y - x_i\|} dy - \bar{\rho} \sum_{i=1}^N \int_{\Omega} \frac{\partial u_i^{(\text{att})}(x_i, y)}{\partial x_i} \theta(x, y) dy$$

In these equations, γ is the repulsive (hard sphere) solvent surface tension (see [gamma](#)), A is the conformation-dependent solute surface area (see [srad](#) and [srfm \(apolar\)](#) keywords), p is the repulsive (hard sphere) solvent pressure (see [press](#) keyword), V is the conformation-dependent solute volume (see [srad](#) and [srfm \(apolar\)](#) keywords), ρ (see [bconc](#) keywords) is the bulk solvent density, and the integral involves the attractive portion (defined in a Weeks-Chandler-Andersen sense) of the Lennard-Jones interactions between the solute and the solvent integrated over the region of the problem domain outside the solute volume V . Lennard-Jones parameters are taken from APBS parameter files as read in through an APBS input file `READ` statement (see [READ input file section](#)).

Note: The above expressions can easily be reduced to simpler apolar solvation formalisms by setting one or more of the coefficients to zero through the keywords.

Warning: All APOLAR calculations require a parameter file which contains Lennard-Jones radius and well-depth parameters for all the atoms in the solute PDB. This parameter file must also contain radius and well-depth parameters for water (specifically: residue “WAT” and atom “OW”). Complete parameter files for protein and nucleic acid parameters are not currently available; we prefer geometric flow calculations (coupled polar and apolar components) rather than this model.

PRINT input file section

Note: This section has been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)). See [Results processing section \(optional\)](#) for more information.

This is a very simple section that allows linear combinations of calculated properties to be written to standard output. The syntax of this section is:

```
PRINT {what} [id op id op...] END
```

The first mandatory argument is `what`, the quantity to manipulate or print. This variable is a string that can assume the following values:

elecEnergy Print electrostatic energies as calculated with an earlier *ELEC input file section calcenergy* command.

elecForce Print electrostatic forces as calculated with an earlier *ELEC input file section calcforce* command.

apolEnergy Print apolar energies as calculated with an earlier *APOLAR input file section calcenergy* command.

apolForce Print electrostatic forces as calculated with an earlier *APOLAR input file section calcforce* command.

The next arguments are a series of `id op id op id op ... id` commands where every `id` is immediately followed by an `op` and another `id`.

id This is a variable string or integer denoting the ID of a particular *ELEC input file section* or *APOLAR input file section* calculations. String values of `id` correspond to the optional “names” that can be assigned to *ELEC input file section* or *APOLAR input file section* calculations. Integer values of `id` are assumed to correspond to the sequentially-assigned integer IDs for *ELEC input file section* or *APOLAR input file section* calculations. These IDs start at 1 and are incremented (independently) for each new *ELEC input file section* or *APOLAR input file section* calculation.

op Specify the arithmetic operation (+ for addition and – for subtraction) to be performed on the calculated quantities

For example:

```
# Energy change due to binding
print energy complex - ligand - protein end
# Energy change due to solvation
print energy solvated - reference end
# Solvation energy change due to binding
print energy complex_solv - complex_ref - ligand_solv + ligand_ref - protein_solv +
↪protein_ref end
```

2.2.4 Examples

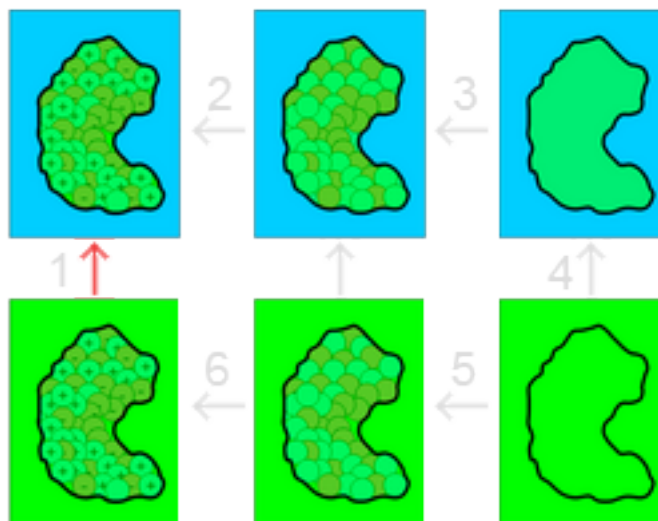
APBS examples start with a PQR file (e.g., generated by [PDB2PQR](#)). Some of these examples can be performed through the [APBS-PDB2PQR web interface](#) but most require a command-line APBS program.

Solvation energies

Solvation energies are usually decomposed into a free energy cycle as shown in the free energy cycle below. Note that such solvation energies often performed on fixed conformations; as such, they are more correctly called “potentials of mean force”. More details on using APBS for the polar and nonpolar portions of such a cycle are given in the following sections.

Our model solvation free energy cycle illustrating several steps:

1. The solvation energy to be calculated.
2. Charging of the solute in solution (e.g., inhomogeneous dielectric, ions present).
3. Introduction of attractive solute-solvent dispersive interaction interactions (e.g., an integral of Weeks-Chandler-Andersen interactions over the solvent-accessible volume).
4. Introduction of repulsive solute-solvent interaction (e.g., cavity formation).
5. Basically a null step although it could be used to offset unwanted energies added in Steps 3 and 4 above.
6. Charging of the solute in a vacuum or homogeneous dielectric environment in the absence of mobile ions.



Polar solvation

The full free energy cycle is usually decomposed into polar and nonpolar parts. The polar portion is usually represented by the charging energies in Steps 2 and 6:

$$\Delta_p G = \Delta_2 G - \Delta_6 G$$

Energies returned from APBS electrostatics calculations are charging free energies. Therefore, to calculate the polar contribution to the solvation free energy, we simply need to setup two calculations corresponding to Steps 2 and 6 in the free energy cycle. Note that the electrostatic charging free energies returned by APBS include self-interaction terms. These are the energies of a charge distribution interacting with itself. Such self-interaction energies are typically very large and extremely sensitive to the problem discretization (grid spacing, location, etc.). Therefore, it is very important that the two calculations in Steps 2 and 6 are performed with identical grid spacings, lengths, and centers, in order to ensure appropriate matching (or “cancellation”) of self-energy terms.

Born ion

One of the canonical examples for polar solvation is the Born ion: a nonpolarizable sphere with a single charge at its center surrounded by an aqueous medium. Consider the transfer of a non-polarizable ion between two dielectrics. In the initial state, the dielectric coefficient inside and outside the ion is ϵ_{in} , and in the final state, the dielectric coefficient inside the ion is ϵ_{in} and the dielectric coefficient outside the ion is ϵ_{out} . In the absence of external ions, the polar solvation energy of this transfer for this system is given by:

$$\Delta_p G_{Born} = \frac{q^2}{8\pi\epsilon_0 a} \left(\frac{1}{\epsilon_{out}} - \frac{1}{\epsilon_{in}} \right)$$

where q is the ion charge, a is the ion radius, and the two ϵ variables denote the two dielectric coefficients. This model assumes zero ionic strength.

Note that, in the case of transferring an ion from vacuum, where $\epsilon_{in} = 1$, the expression becomes

$$\Delta_p G_{Born} = \frac{q^2}{8\pi\epsilon_0 a} \left(\frac{1}{\epsilon_{out}} - 1 \right)$$

We can setup a PQR file for the Born ion for use with APBS with the contents:

```
REMARK  This is an ion with a 3 Å radius and a +1 e charge
ATOM      1      I      ION      1 0.000    0.000    0.000  1.00 3.00
```

We're interested in performing two APBS calculations for the charging free energies in homogeneous and heterogeneous dielectric coefficients. We'll assume the internal dielectric coefficient is 1 (e.g., a vacuum) and the external dielectric coefficient is 78.54 (e.g., water). For these settings, the polar Born ion solvation energy expression has the form

$$\Delta_p G_{\text{Born}} = -691.85 \left(\frac{z^2}{R} \right) \text{kJ A/mol}$$

where z is the ion charge in electrons and R is the ion size in Å.

This solvation energy calculation can be setup in APBS with the following input file:

```
# READ IN MOLECULES
read
  mol pqr born.pqr
end
elec name solv # Electrostatics calculation on the solvated state
  mg-manual # Specify the mode for APBS to run
  dime 97 97 97 # The grid dimensions
  nlev 4 # Multigrid level parameter
  grid 0.33 0.33 0.33 # Grid spacing
  gcent mol 1 # Center the grid on molecule 1
  mol 1 # Perform the calculation on molecule 1
  lpbe # Solve the linearized Poisson-Boltzmann equation
  bcfl mdh # Use all multipole moments when calculating the potential
  pdie 1.0 # Solute dielectric
  sdie 78.54 # Solvent dielectric
  chgm spl2 # Spline-based discretization of the delta functions
  srfm mol # Molecular surface definition
  srads 1.4 # Solvent probe radius (for molecular surface)
  swin 0.3 # Solvent surface spline window (not used here)
  sdens 10.0 # Sphere density for accessibility object
  temp 298.15 # Temperature
  calcenergy total # Calculate energies
  calcforce no # Do not calculate forces
end
elec name ref # Calculate potential for reference (vacuum) state
  mg-manual
  dime 97 97 97
  nlev 4
  grid 0.33 0.33 0.33
  gcent mol 1
  mol 1
  lpbe
  bcfl mdh
  pdie 1.0
  sdie 1.0
  chgm spl2
  srfm mol
  srads 1.4
  swin 0.3
  sdens 10.0
  temp 298.15
  calcenergy total
  calcforce no
```

(continues on next page)

(continued from previous page)

```

end
# Calculate solvation energy
print energy solv - ref end
quit

```

Running this example with a recent version of APBS should give an answer of -229.59 kJ/mol which is in good agreement with the -230.62 kJ/mol predicted by the analytic formula above.

Note: The Born example above can be easily generalized to other polar solvation energy calculations. For example, ions could be added to the solv ELEC, dielectric constants could be modified, surface definitions could be changed (in both ELEC sections!), or more complicated molecules could be examined. Many of the examples included with APBS also demonstrate solvation energy calculations.

Note: As molecules get larger, it is important to examine the sensitivity of the calculated polar solvation energies with respect to grid spacings and dimensions.

Apolar solvation

Referring back to the solvation free energy cycle, the nonpolar solvation free energy is usually represented by the energy changes in Steps 3 through 5:

$$\Delta_n G = (\Delta_3 G - \Delta_5 G) + \Delta_4 G$$

where Step 4 represents the energy of creating a cavity in solution and Steps 3-5 is the energy associated with dispersive interactions between the solute and solvent. There are many possible choices for modeling this nonpolar solvation process. APBS implements a relatively general model described by [Wagoner and Baker \(2006\)](#) and references therein. The implementation and invocation of this model is described in more in the *APOLAR input file section* documentation. Our basic model for the cavity creation term (Step 4) is motivated by scaled particle theory and has the form

$$\Delta_4 G = pV + \gamma A$$

where p is the solvent pressure (*press* keyword), V is the solute volume, γ is the solvent surface tension (*gamma* keyword), and A is the solute surface area.

Our basic model for the dispersion terms (Steps 3 and 5) follow a Weeks-Chandler-Anderson framework as proposed by [Levy et al \(2002\)](#):

$$\Delta_3 G - \Delta_5 G = \bar{\rho} \int_{\Omega} u^{(att)}(y) \theta(y) dy$$

where $\bar{\rho}$ is the bulk solvent density (*bconc* keyword), Ω is the problem domain, $u^{(att)}(y)$ is the attractive dispersion interaction between the solute and the solvent at point y with dispersive Lennard-Jones parameters specified in APBS parameter files, and $\theta(y)$ describes the solvent accessibility of point y .

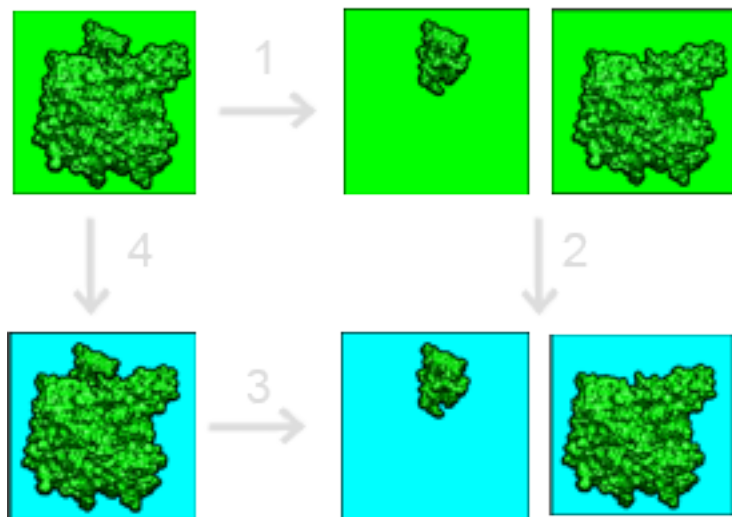
The ability to independently adjust *press*, *gamma*, and *bconc* means that the general nonpolar solvation model presented above can be easily adapted to other popular nonpolar solvation models. For example, setting *press* and *bconc* to zero yields a typical solvent-accessible surface area model.

Binding energies

In general, implicit solvent models are used to calculate the contribution of solvation to binding free energies. Additional binding free energy contributions (molecular mechanics energies, entropic changes, etc.) must be calculated separately and are not discussed in this tutorial.

Free energy cycle

Our framework for calculating solvation contributions to binding free energies is shown in the figure below:



This binding free energy cycle illustrates binding in terms of transfer free energies from a homogeneous dielectric environment (where interactions are described by Coulomb's law) to an inhomogeneous dielectric environment with differing internal (green) and external (cyan) dielectric constants. The binding (dissociation) free energy is depicted in Step 3. The binding free energy is given by

$$\Delta_b G = -\Delta_3 G = \Delta_4 G - \Delta_1 G - \Delta_2 G.$$

The following sections provide more detail on calculating individual terms of this equation.

Binding energy calculations

The most general method for calculating binding free energies divides the binding process up into solvation $\Delta\Delta_s G$ and Coulombic $\Delta\Delta_c G$ components:

$$\Delta\Delta_b G = \Delta\Delta_s G + \Delta\Delta_c G.$$

As mentioned above, this framework neglects the numerous other mechanical and entropic components actually involved in the binding process.

Solvation contribution to binding

If we're just interested in calculating the solvation contributions to binding (steps 4 and 2 in the binding free energy cycle), then we simply need to follow the instructions from the [Solvation energies](#) section for the complex and isolated components. The solvation energy contribution to the binding is then given by

$$\Delta\Delta_s G = \Delta_4 G - \Delta_2 G = \Delta_s G_{\text{cmpx}} - \Delta_s G_{\text{mol1}} - \Delta_s G_{\text{mol2}}$$

Coulombic contribution to binding

To complete the binding free energy cycle, we need to add intermolecular Coulombic contributions to the solvation energy change upon binding to get the total electrostatic/solvent contribution to the binding free energy. In particular, we're interested in the change in Coulombic electrostatic energy upon binding, as given by

$$\Delta\Delta_c G = -\Delta_1 G = \Delta_c G_{\text{complex}} - \Delta_c G_{\text{mol1}} - \Delta_c G_{\text{mol2}}$$

Each of the quantities in this equation is the sum of pairwise Coulombic interactions between all atoms in the molecule (or complex) for a particular uniform dielectric. In order to combine these Coulombic binding energies with the solvation energies described above, we need to make sure consistent dielectric constants are used. In particular, Coulombic interactions should be calculated using the same uniform dielectric constant as the reference state of the solvation energy above. For example, if solvation energies are calculated for transferring a protein from a homogeneous medium with uniform dielectric of to an inhomogeneous medium with internal dielectric ϵ_u and external dielectric ϵ_v , then Coulombic energies should be calculated using a dielectric of ϵ_u . The APBS accessory program `tools/manip/coulomb` was created to help with the calculation of these analytic individual per-molecule Coulombic energies. Given a PQR file as input, the `tools/manip/coulomb` program calculates Coulombic energies for a vacuum dielectric (e.g., a uniform dielectric of 1). If the reference dielectric is ϵ_u , then all energies returned by `tools/manip/coulomb` need to be divided by ϵ_u .

Other examples

Several binding energy examples are distributed in the `examples` directory with APBS.

Protein-RNA binding linked equilibria

Before reading this example, please review *Caveats and sources of error* for relevant caveats.

Introduction

This example is taken from a paper by García-García and Draper. Special thanks to David Draper who provided the PDB files. This example explores the electrostatic contributions to the binding interaction between a 22-residue α -helical peptide of protein λ with the “box B” RNA hairpin structure. In particular, this example uses nonlinear Poisson-Boltzmann equation calculations to look at the non-specific screening effects of monovalent salt on the peptide-RNA complex. García-García and Draper isolated the contribution of KCl concentration to the binding of the folded peptide with the folded RNA hairpin and determined a fairly linear relationship between the binding free energy $\Delta_b G$ and the logarithm of the KCl concentration which yields

$$\frac{\partial \Delta_b G}{\partial \log_{10}[\text{KCl}]} = 6.0 \pm 0.2 \text{ kcal/mol}$$

This slope can be used to determine the number of KCl ions linked to the binding equilibrium through the expression

$$n = -\frac{\partial \Delta_b G}{RT \partial \log_{10}[\text{KCl}]} = -4.52 \pm 0.08 \text{ kcal/mol}$$

where RT is the thermal energy, to determine $n = -4.4 \pm 0.2$ for the RNA-peptide binding equilibrium. RT is equal to $kT * N_a$ where kT is the product of the Boltzmann constant k (equal to the gas constant R/N_a), and the temperature T (at STP it is 298.15 K) and N_a is Avogadro's constant. Thus RT is equal to

$$R \text{ (Joules/Kelvin)} * T \text{ (Kelvin)} * N_a \text{ (mols)} * 1 \text{ kJ/1000 J}$$

which roughly equals

$$(1.38 \times 10^{-23}) \times (6.022 \times 10^{23}) \times (298.15)/(1000)$$

which is approximately 2.479 kJ/mol or 0.593 kcal/mol.

García-García and Draper used nonlinear Poisson-Boltzmann equation calculations to estimate the electrostatic contributions to the binding free energy as a function of the monovalent salt concentration. As *discussed elsewhere*, the Poisson-Boltzmann equation is only able to describe non-specific interactions of ions with solutes, including the effects of ion size and charge but otherwise ignoring the important differences between ionic species. Interestingly (and perhaps surprisingly), they find excellent agreement between the experimental binding energy dependence on KCl and their Poisson-Boltzmann calculations with equivalent concentrations of monovalent ions. This agreement strongly suggests that the binding of RNA and the peptide is primarily determined by electrostatic interactions. It also suggests that the primary interaction of the KCl with this system is through non-specific screening interactions. The García-García and Draper nonlinear Poisson-Boltzmann equation calculations gave:

$$\frac{\partial \Delta_b G}{\partial \log_{10}[\text{KCl}]} = 5.9 \pm 0.2 \text{ kcal/mol}$$

and $n = -4.3 \pm 0.2$ for KCl linkage to the RNA-peptide binding equilibrium.

APBS implementation

This example follows the calculations from their paper.

The PQR files are included in the `examples/protein-rna/` directory of the apbs repository. This directory also includes a `template.txt` file that serves as a template for the APBS input files with `IONSTR` as a placeholder for the ionic strength. This file is also shown here:

```
read
  mol pqr model_outNB.pqr
  mol pqr model_outNpep.pqr
  mol pqr model_outBoxB19.pqr
end
elec name complex
  mg-auto
  dime 65 97 129
  cglen 45.3322 54.9498 82.2633
  fglen 45.3322 52.3234 68.3902
  cgcent mol 1
  fgcent mol 1
  mol 1
  npbe
  bcfl sdh
  pdie 4.0
  ion charge 1 conc IONSTR radius 2.0
  ion charge -1 conc IONSTR radius 2.0
  sdie 80.0
  srfm mol
  chgm spl2
  sdens 10.00
  sradi 1.40
  swin 0.30
  temp 298.15
  calcenergy total
  calcforce no
  write qdens dx qdens-complex-IONSTR
```

(continues on next page)

(continued from previous page)

```

    write ndens dx ndens-complex-IONSTR
end
elec name peptide
  mg-auto
  dime 65 97 129
  cglen 45.3322 54.9498 82.2633
  fglen 45.3322 52.3234 68.3902
  cgcent mol 1
  fgcent mol 1
  mol 2
  npbe
  bcfl sdh
  pdie 4.0
  sdie 80.0
  ion charge 1 conc IONSTR radius 2.0
  ion charge -1 conc IONSTR radius 2.0
  srfm mol
  chgm spl2
  sdens 10.00
  srاد 1.40
  swin 0.30
  temp 298.15
  calcenergy total
  calcforce no
  write qdens dx qdens-peptide-IONSTR
  write ndens dx ndens-peptide-IONSTR
end
elec name rna
  mg-auto
  dime 65 97 129
  cglen 45.3322 54.9498 82.2633
  fglen 45.3322 52.3234 68.3902
  cgcent mol 1
  fgcent mol 1
  mol 3
  npbe
  bcfl sdh
  pdie 4.0
  sdie 80.0
  ion charge 1 conc IONSTR radius 2.0
  ion charge -1 conc IONSTR radius 2.0
  srfm mol
  chgm spl2
  sdens 10.00
  srاد 1.40
  swin 0.30
  temp 298.15
  calcenergy total
  calcforce no
  write qdens dx qdens-rna-IONSTR
  write ndens dx ndens-rna-IONSTR
end
print elecEnergy complex - peptide - rna end
quit

```

As used in the template file, the READ command, our calculation will have three parts:

- Calculation of the total electrostatic energy (including self-interaction energies) of the peptide-RNA complex.

This calculation is named `complex` in the input file.

- Calculation of the total electrostatic energy (including self-interaction energies) of the peptide. This calculation is named `peptide` in the input file.
- Calculation of the total electrostatic energy (including self-interaction energies) of the RNA. This calculation is named `rna` in the input file.

The calculations themselves will not be overly demanding, since we will use relatively coarse grids. This grid coarseness has a significant impact on the absolute electrostatic binding energy we obtain from this particular calculation: the calculated energy isn't converged with respect to grid spacing. However, the overall slope of binding energy with respect to monovalent ion concentration is rather insensitive with respect to the grid spacing, allowing us to save computational time and effort during the calculations. The calculation will conclude with a `/using/input/print` command which will combine the total energies from the three parts to obtain our approximate absolute electrostatic binding energy for the complex at 0.225 M monovalent salt concentration. It is very important to note that this absolute energy no meaning in isolation for several reasons:

- It is not converged with respect to grid spacing
- It does not contain other very important non-electrostatic aspects of the binding energy which are important for the measured affinity

`IONSTR` is a placeholder that represents the ion concentration for the APBS calculation.

You will also have to create a `dxmath.txt` file which contains the following.

```
qdens-complex-IONSTR.dx
qdens-pep-IONSTR.dx -
qdens-rna-IONSTR.dx -
qdens-diff-IONSTR.dx =
```

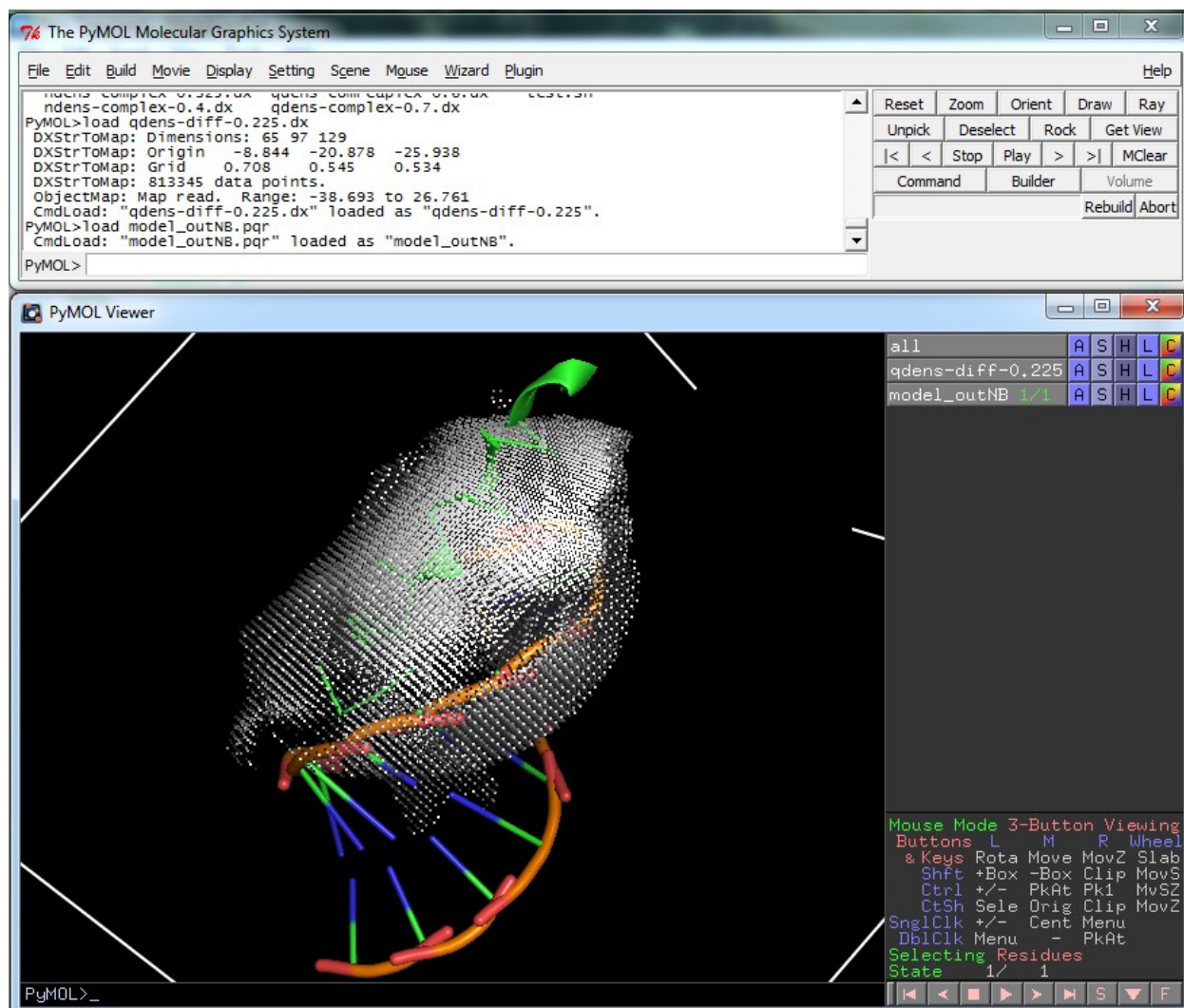
`dxmath` will subtract the dx maps of the individual peptide and RNA from the overall structure (and prints to the `qdens-diff-IONSTR.dx` file.

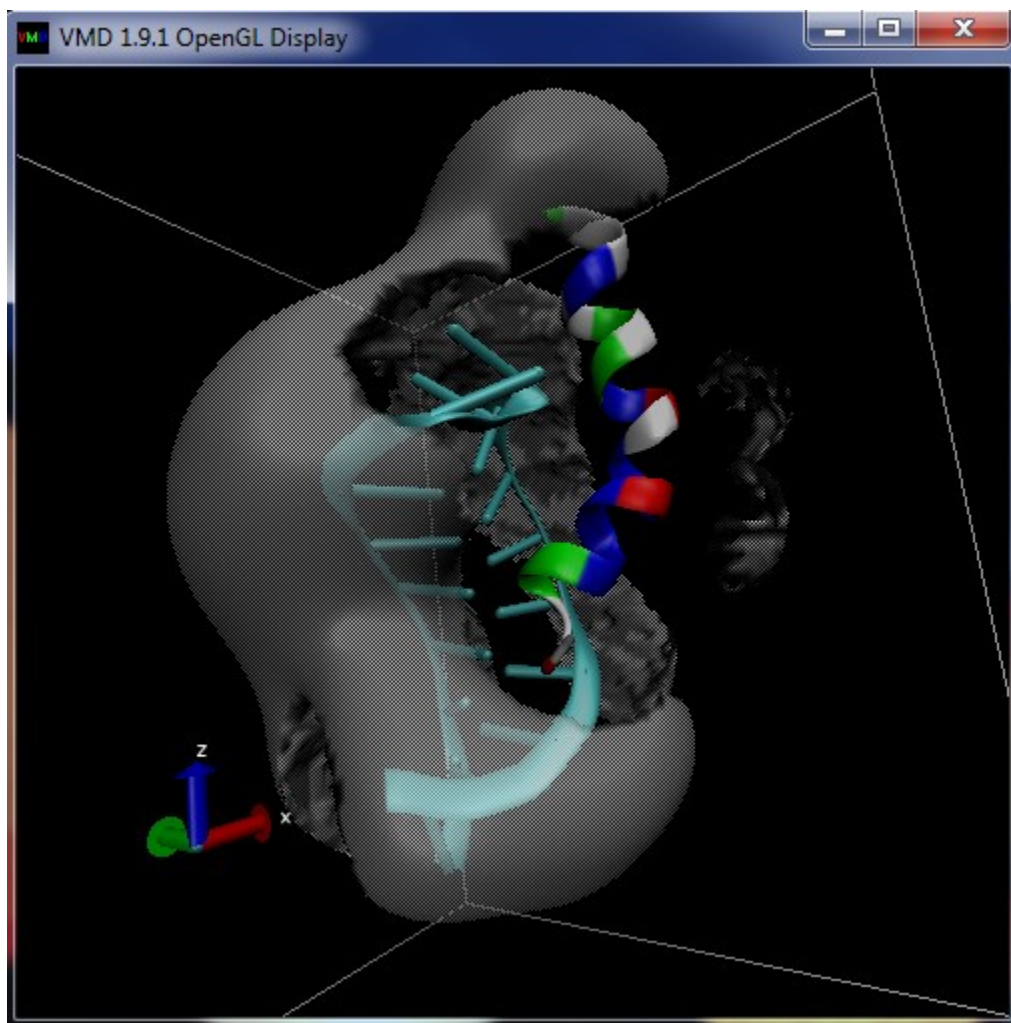
Automation with Python

We have provided Python scripts `apbs_win`, `unix_dx.py` that run the necessary APBS calculations and analyze the results. When you run these programs, you need to be in the same directory as `template.txt` and `dxmath.txt`. This script will create all the input files for the tests as well as run `apbs` and `dxmath` on your `template.txt` and `dxmath.txt` files. Most of the syntax fills in the ion concentrations in the template file, and the call commands actually run the calculations on each input.

Visualization

The `qdens-diff-0.225.dx` file produced by the script can be viewed in PyMOL or another visualization program to give something similar to the following imaged which show the difference in charge density before and after binding.





Visualization with PyMOL

The [PyMOL](#) molecular graphics software package can both run APBS and visualize resulting electrostatic potentials. Below are instructions for performing a basic demonstration of how to go from a PDB entry to a plot of structure and potential in PyMOL using APBS.

Run the APBS calculation

- Load your PQR file you created into PyMOL (*File* → *Open...*) and choose your favorite graphical representation of the molecular structure.
- Go to *Plugin* → *APBS Tools...* to open the APBS calculation plugin.
- Under the *Main* tab of the PyMOL APBS Tools window, select *Use another PQR* and either browse to (via the *Choose Externally Generated PQR* button) or input the path to your PQR file. This step is necessary to ensure you use the radii and charges assigned by PDB2PQR.
- Under the *APBS Location* tab of the PyMOL APBS Tools window, either browse to (via the *APBS binary location: button*) or input the path to your local APBS binary. It is not necessary to provide a path to the APBS `psize.py` binary for most biomolecules.

- Under the *Temporary File Locations* tab of the PyMOL APBS Tools window, customize the locations of the various temporary files created during the run. This can be useful if you want to save the generated files for later use.
- Under the *Configuration* tab of the PyMOL APBS Tools window, press *Set grid* to set the grid spacings. The default values are usually sufficient for all but the most highly charged biomolecules.
- Under the *Configuration* tab of the PyMOL APBS Tools window, customize the remaining parameters; the defaults are usually OK.

Note: 0.150 M concentrations for the +1 and 1 ion species are often useful to ensure that electrostatic properties are not overly exaggerated.

- Under the *Configuration* tab of the PyMOL APBS Tools window, press the *Run APBS button* to start the APBS calculation. Depending on the speed of your computer, this could take a few minutes. The *Run APBS button* will become unselected when the calculation is finished.

Visualize the results

Before proceeding, you must load the electrostatic potential data into PyMOL. Under the *Visualization* tab of the PyMOL APBS Tools window, press the *Update* button.

Electrostatic isocontours

PyMOL makes this step very easy: adjust the positive and negative “Contour” fields to the desired values (usually ± 1 , ± 5 , or ± 10 kT/e) and press the *Positive Isosurface*, *Negative Isosurface*, and *Show buttons*.

At this point, you probably have a figure that looks something like the image below.

If the colors are not as you expect, you can change the colors of the objects *iso_neg* and *iso_pos* in the main menu. By convention (for electrostatics in chemistry), red is negative (think oxygen atoms in carboxyl groups) and blue positive (think nitrogen atoms in amines).

Surface potentials

If you haven’t already, hide the isocontours by pressing the *Positive Isosurface*, *Negative Isosurface*, and *Hide* buttons. The surface potential is also straightforward to visualize. Set the “Low” and “High” values to the desired values (usually ± 1 , ± 5 , or ± 10 kT/e) at which the surface colors are clamped at red (-) or blue (+). Check the “Solvent accessible surface” and “Color by potential on sol. acc. surf.” buttons to plot the potential on the solvent-accessible (probe-inflated or Lee-Richards) surface. Press the *Molecular Surface Show* button to load the surface potential.

The solvent-accessible surface tends to reveal more global features of the surface potential. Tighter surfaces (e.g., van der Waals and molecular or Connolly surfaces) provide more information about the shape of the biomolecule but otherwise tend to simply map atomic surface charges onto the biomolecular surface. PyMOL can simultaneously provide geometric information (from the molecular surface) and useful electrostatic potential information (from the solvent-accessible surface). To visualize the molecule in this way, simply uncheck the “Solvent accessible surface” box and check the “Color by potential on sol. acc. surf.” box on the *Visualization* tab.

Virtual reality with UnityMol

Molecular visualization software packages provide the ability for users to explore the 3D representations molecular structures and properties. Typical user interaction is limited to panning, zooming, and rotating the molecule

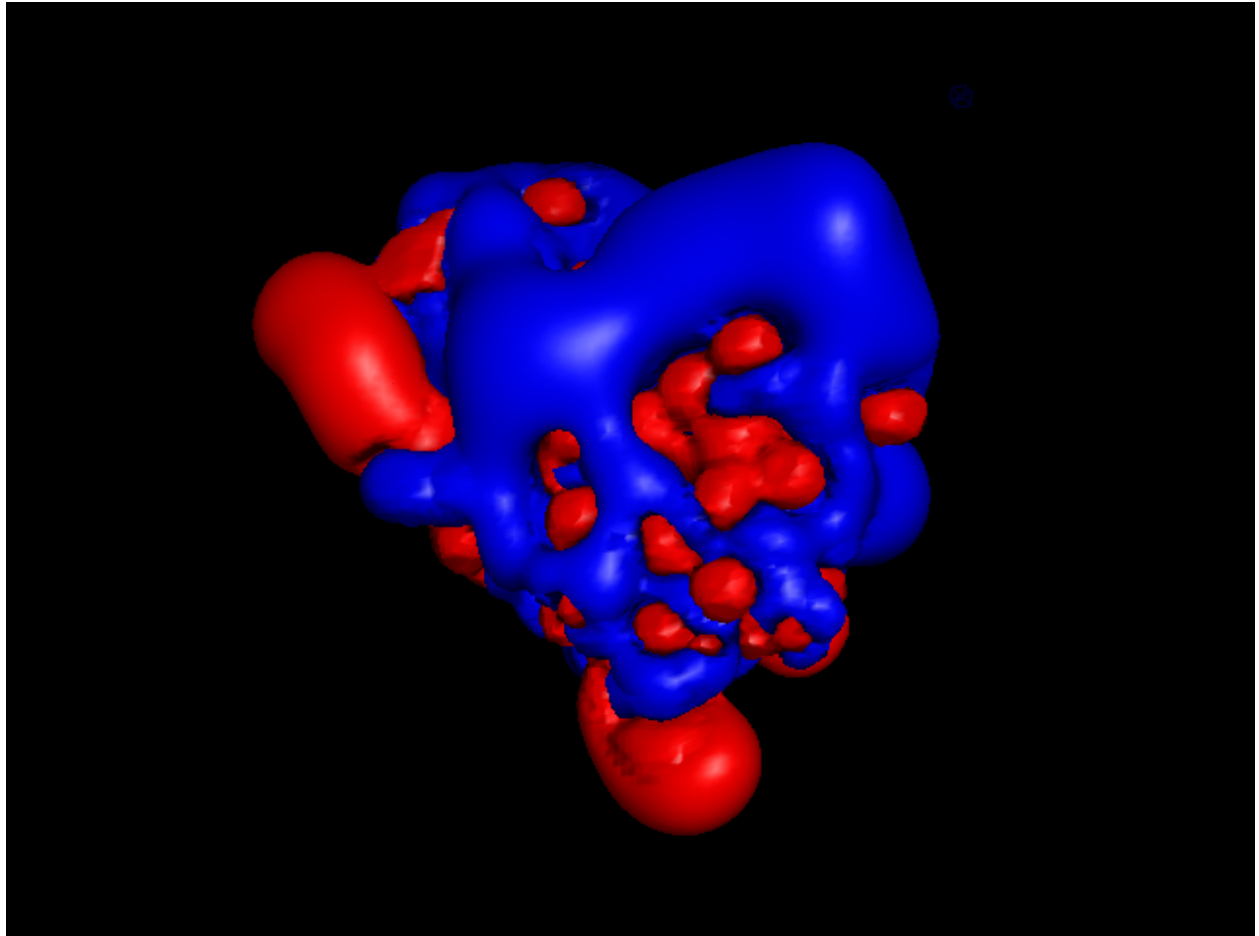


Fig. 1: ± 1 kT/e electrostatic potential isocontours of FAS2 in PyMOL

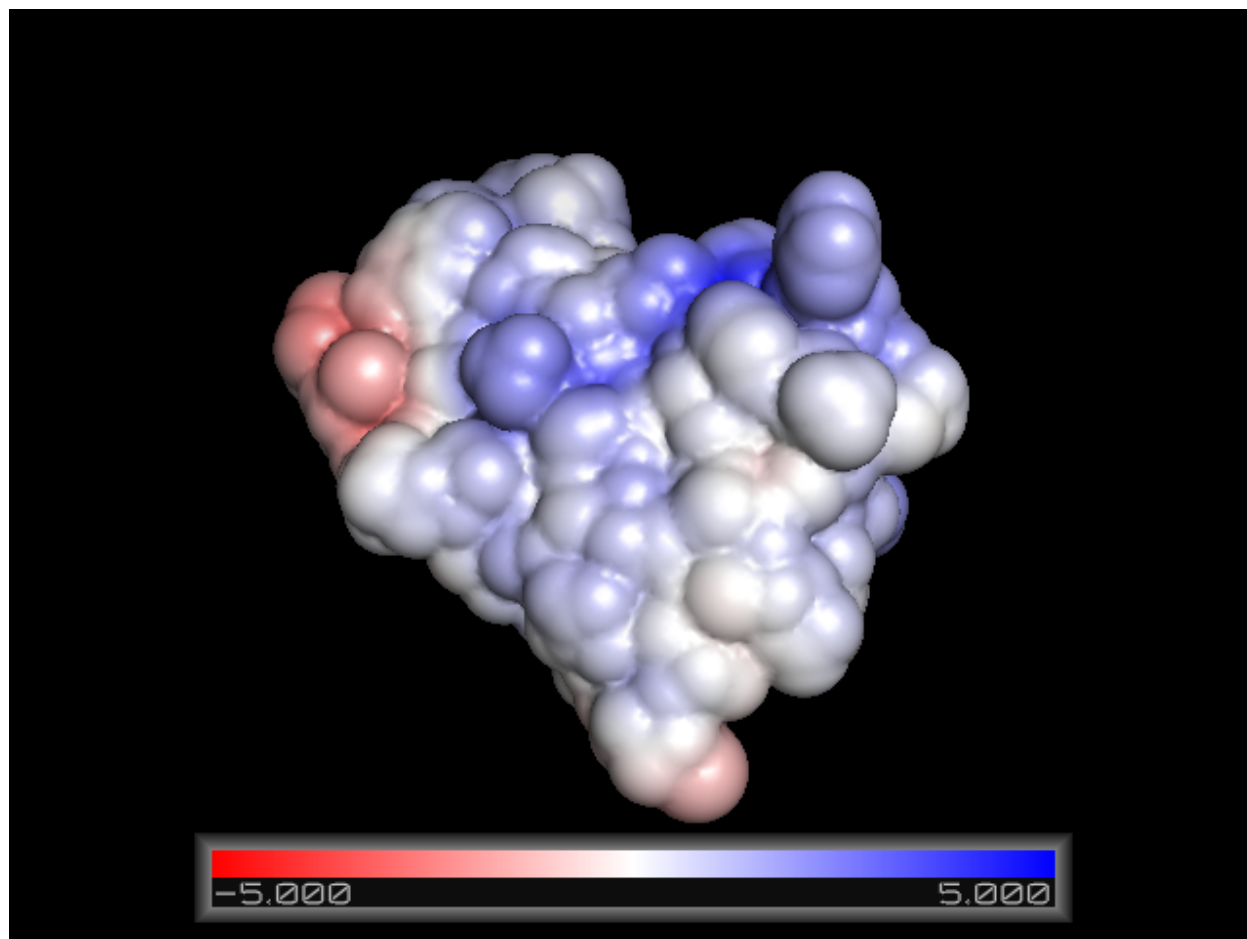


Fig. 2: ± 5 kT/e electrostatic potential of FAS2 in PyMOL plotted on the solvent-accessible surface.

using a mouse and keyboard while viewing on a standard computing monitor. These techniques support a pseudo 3-dimensional view of a molecule to understand its structure but lack the true depth perception people are used to with stereoscopic vision in the real world.

New advancements in virtual reality (VR) technologies has resulted in lower costs and systems that are easier to use to many consumers. Compared to past VR hardware, these new systems have several key advancements including lower latency, higher frame rates, and improved resolution. Additionally, these systems are equipped with better optics and motion tracking and a more robust software ecosystem.

We are extending the visualization capabilities for APBS through the incorporation of a VR device with molecular rendering software. We are currently experimenting with the HTC Vive, which allows a person to walk around a 15' by 15' physical space while wearing a head mounted display. Precise head movements are matched in virtual reality with no noticeable latency. Additionally, the HTC Vive controllers are motion tracked with millimeter precision and provide a valuable method for interacting with virtual objects. We have enabled VR using the HTC Vive in the [UnityMol molecular visualization software](#) (created by Baaden, et al.) and incorporated electrostatic surface data (see figure below and a [YouTube video](#)). New viewing capabilities now include walking around, grabbing (using the motion controllers), and scaling (gestures) of molecules. We are actively working with Dr. Baaden and his group to determine the best use of interaction techniques for users to interact with molecular models through his software.

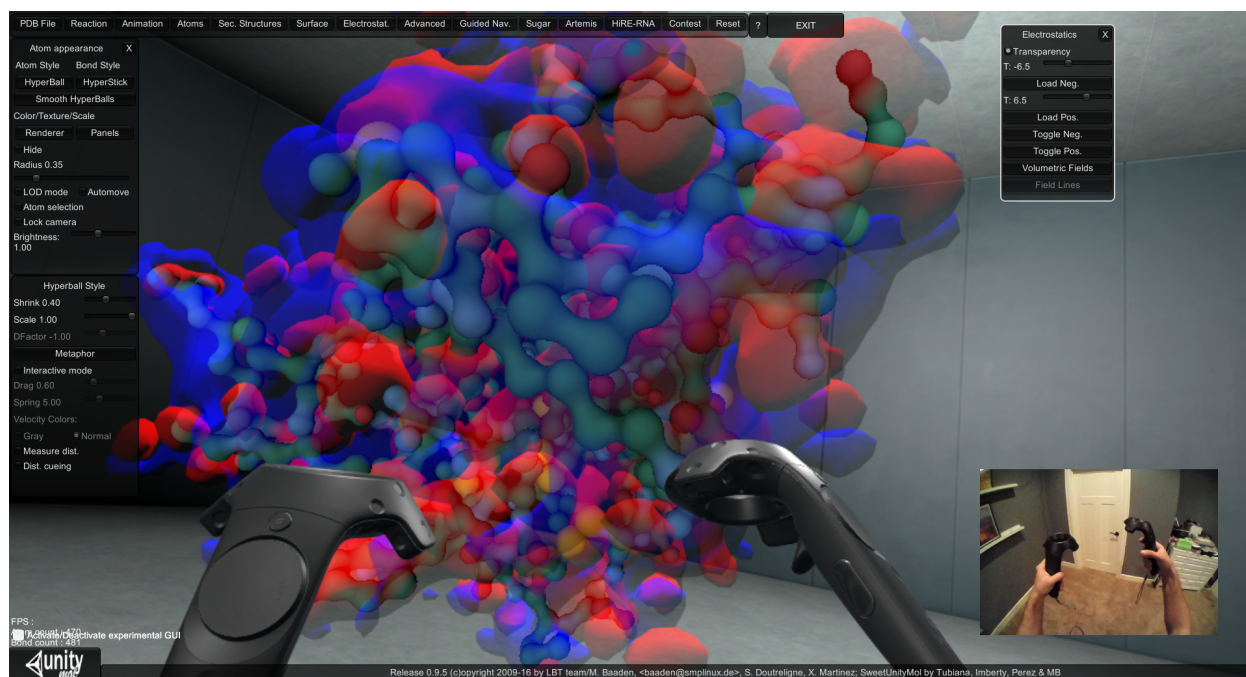


Fig. 3: View of UnityMol from the monitor as it is being used in VR with controllers.

For future work, we would like to further extend UnityMol in the HTC Vive to include natural user interactions for viewing multiple molecules, vary the electrostatic results from APBS, and change molecular attributes. We envision this tool will also enable virtual collaboration for participant in different locations. Each participant will be able to view, gesture and interact with the same data in the same VR space. Finally, we would like to explore the use of VR for research related to docking of different molecules.

Getting the software

1. Download `UnityMol-APBS-PS.zip` from [SourceForge](#).
2. Unzip `UnityMol-APBS-PS.zip`; the resulting folder contains `UnityMol-APBS.zip` and `APBS-PDB2PQR.zip`.

3. Unzip UnityMol-APBS.zip; the resulting folder contains **UnityMol.exe**.
4. Optionally unzip APBS-PDB2PQR.zip into C:\ to generate three directories :file:\apbs (containing **apbs** executable), pdb2pqr (containing **pdb2pqr** executable), and OutputFiles. Alternatively, these executables can be downloaded and installed separately.

Using the software

Launch UnityMol.exe **UnityMol.exe** to start the VR visualization. The user interface is illustrated below.



Fig. 4: UnityMol-APBS user interface for PDB2PQR and APBS. (A) The main UnityMolAPBS menu; orange box highlights the two buttons used to open the APBS and PDB2PQR tools. (B) The main menu for interactions with APBS and PDB2PQR. Blue boxes show the buttons to launch PDB2PQR and APBS executables, green boxes show the location of the options used for producing the image in below, and the purple boxes highlight the two input fields required to use custom force fields and custom residue names.

Acetylcholinesterase example

The example illustrates the VR vizualization of the electrostatic surface potentials and electrostatic field lines of *Torpedo californica* acetylcholinesterase (AChE).

1. Download `5ei5.pdb` from <https://www.rcsb.org/structure/5EI5>
2. Open UnityMol-APBS (VR or desktop)
3. Load `5ei5.pdb` file
4. Open the *PDB2PQR panel*
5. Choose *options* (examples below) or run the default (default force field is AMBER)
 - *apbs-input* generates input file necessary for APBS
 - *drop-water* removes explicit water molecules from structure
 - *summary* writes atom names and sequence to a new file
 - *salt* writes salt bridge donor and acceptor atoms to a new file
 - *hbond* writes hydrogen bonding donors and acceptors to a new file. The resulting `.hbond` and `.salt` files can be loaded as a new selection in UnityMol-APBS
6. Select *all(5EI5)* and run PDB2PQR
7. `5ei5X.pqr` is written to a file and is immediately loaded for the user.
8. Select *all(5EI5)* and run APBS
9. `5ei5X.dx` is created and loaded into the selection *all(5EI5X)* automatically
10. Select the + button on the *all(5EI5X)* selection tab, then select *surface*
11. Select *color by charge*
12. Select the + button on the *all(5EI5X)* selection tab, then select *field lines*

As a result of these steps, you should see a figure similar to the following.

2.2.5 Tests and validation

APBS is distributed with testing tools and validation examples.

APBS validation and test cases

This directory serves as the root directory for the APBS test suite. This directory contains python source files used for testing an input file containing the input files used by the `apbs` executable and the expected results for each test case.

The default input file is called *test_cases.cfg*, and the main testing program is called *apbs_tester.py*.

Usage

Note: It is important that you run the tests from the command line and that you run them from within the *INSTALL_DIR/share/apbs/tests* directory.

A usage description for *apbs_tester.py* can be obtained by running:

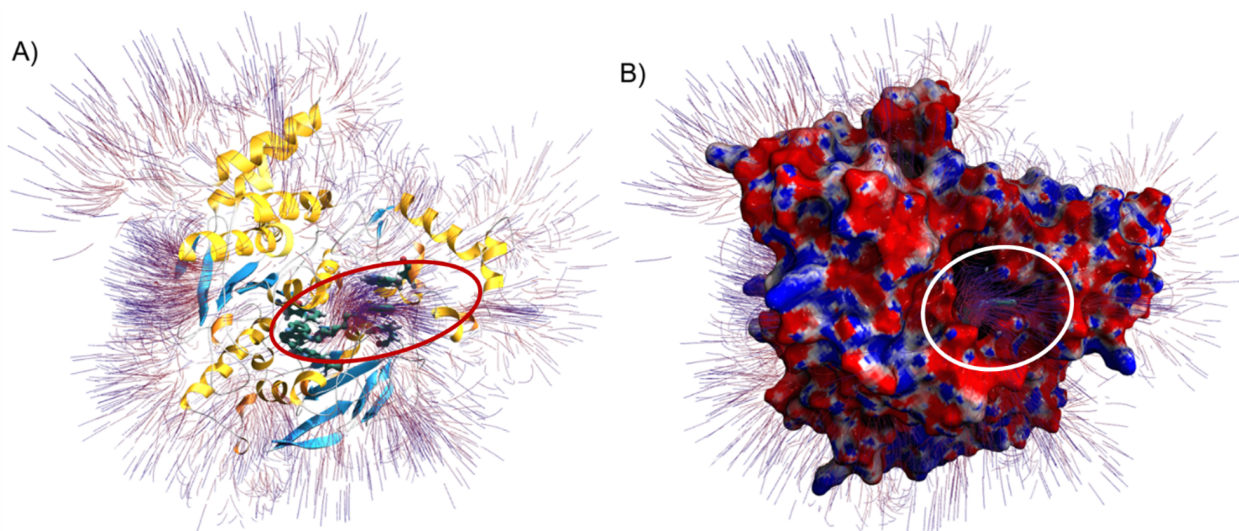


Fig. 5: Electrostatic surface potential and field lines of *Torpedo californica* AChE (PDB ID 5EI5) with bound alkylene-linked bis-tacrine. (A) Electrostatic field lines and protein secondary structure shown with alpha helices (yellow), beta sheets (blue), and random coils (white). Residues Tyr70, Trp84, Trp279, and Phe330 are shown interacting with alkylene-linked bis-tacrine via hydrogen bonding and π - π stacking interactions. The red oval highlights the potential gradient. (B) AChE surface model with field lines and mapped electrostatic surface potentials shown with neutral, negative, and positive charges in white, red, and blue, respectively. Field lines are calculated from a gradient (value of 0.2) and depicted with the starting points in red and the ending points in blue. The orientation is the same in Figures A and B, where the alkylene-linked bis-tacrine can be seen occupying the catalytic gorge. The white circle highlights the potential gradient exiting the catalytic gorge.

```
python3 apbs_tester.py -h
```

Test Sections

The sections of the test file, `test_cases.cfg`, follow the following format:

```
[Some-Target_Test]
input_dir      : ../path/to/some-example
some-forces    : forces
some-input     : * * 1.0E+01 2.0E+02
```

where:

- The first element in brackets `[Some-Target_Test]` describes the name of the *target_test* section.
- After the first element, the remaining elements are *property/value* pairs
- The first property is the **input_dir**. This is the location of all input files referenced in other properties.
- A property has a *name* that is also the basename of the input file concatenated with the file extension, `input-file.in`.
- The property *name* will also be used for the output from `apbs some-input.in` to create the output file, `some-input.out`
- If the *value* of the property is **forces** the test will calculate forces.
- If the *value* of the property is a list of floating point numbers, the values are expected outputs.

- If a \star is used in place of a floating point number, the output will be ignored. Some test cases have multiple outputs. The test function parses each of these, but if a \star is used, the output will be ignored in testing. Most often, the first outputs are intermediate values followed by a final output, and the test case is only concerned with the final output.

Examples

The following will run the *apbs_tester.py* specifying the path to the apbs executable and using the geoflow section of the test_cases.cfg file:

```
python3 apbs_tester.py -e ${INSTALL_DIR}/bin/apbs -c test_cases.cfg -t geoflow
```

where **INSTALL_DIR** is the path to your installation directory.

2.2.6 Tools and utilities

APBS is distributed with utilities designed to simplify typical tasks associated with electrostatics calculations.

Conversion utilities

amber2charmm.sh

A bash script for converting AMBER atom names to CHARMM names. Found in `tools/conversion`

del2dx

Converts DelPhi-format map files (electrostatic potential, etc.) to APBS OpenDX format. Found in `tools/mesh`

dx2mol

Converts OpenDX format map files to MolMol format. Found in `tools/mesh`

dx2uhbd

Converts OpenDX format map files to UHBD format. Found in `tools/mesh`

qcd2pqr.awk

An awk script for converting from UHBD QCD format to PQR format.

Benchmarking utilities

benchmark

Benchmark file I/O for reading/writing scalar data. Found in `tools/mesh`

uhbd_asc2bin

Converts UHBD ASCII-format files to binary format. Found in `tools/mesh`

WHATIF2AMBER.sed

A sed script for converting WHATIF atoms names to the AMBER naming scheme. Found in `tools/conversion`

Setup and analysis utilities**analysis**

Calculates various metrics from input scalar data. Found in `tools/mesh`

born

Calculate generalized Born forces and energies. Found in `tools/manip`

coulomb

Calculate Coulomb forces and energies. Found in `tools/manip`

dxmath

Performs simple arithmetic operations with Cartesian grid data. This program takes as input a file with operations specified in a stack-based (RPN) manner. For example, a command file which adds grid1 and grid2, multiplies the result by 5.3, adds grid4, subtracts 99.3 from the whole thing, and writes the result on grid5 would have the form:

```
grid1
grid2 +
5.3 *
grid4 +
99.3 -
grid5 =
```

The file names, scalar values, and operations must be separated by tabs, line breaks, or white space. Comments can be included between the character # and a new line (in the usual shell script fashion). Found in `tools/mesh`

inputgen.py

Create an APBS input file using *psize.py* data. Found in `tools/manip`

mergedx and mergedx2

Combine multiple OpenDX files into a single resampled file. **mergedx2** can perform a number of grid manipulation operations, including:

- Combining multiple OpenDX map files

- Resampling of one or more OpenDX map files (for example to alter the grid spacing of separate OpenDX files for further manipulation)
- Extracting a subregion of an existing OpenDX map file.

Found in `tools/mesh`

mgmesh

Prints out acceptable combinations of `input/elec/nlev` and `input/elec/dime` for multigrd calculations. Found in `tools/mesh`

multivalue

This program evaluates OpenDX scalar data at a series of user-specified points and returns the value of the data at each point. Found in `tools/mesh`

psize.py

Suggest grid sizes and spacings for APBS given an input molecule. Found in `tools/manip`

similarity

Computes similarity between two scalar grid datasets. Found in `tools/mesh`

smooth

Convolve grid data with various filters. Found in `tools/mesh`

2.2.7 Other software

A variety of other software can be used to visualize and process the results of PDB2PQR and APBS calculations.

Visualization software

Examples of visualization software that work with output from PDB2PQR and APBS:

- [PyMOL](#)
- [VMD](#)
- [Chimera](#)
- [PMV](#)

Dynamics simulations

As an example of PDB2PQR and APBS integration with molecular mechanics software, the **iAPBS** library was developed to facilitate the integration of APBS with other molecular simulation packages. This library has enabled the integration of APBS with several molecular dynamics packages, including **NAMD**, **AMBER**, and **CHARMM**.

APBS is also used directly by Brownian dynamics software such as **SDA** and **BrownDye**.

2.2.8 Notes on units

APBS and PDB2PQR use a few different sets of units, explained in the following sections: The following are preferred APBS units.

Numbers

Many quantities are scaled by Avogadro's number, the number of atoms in a mole (mol), $N_A = 6.02214076 \times 10^{23} \text{ mol}^{-1}$.

Length

The preferred unit of length is the ångström (Å), equal to 10^{-10} meters.

Volume

The preferred unit of volume is Å³, equal to 10^{-27} liters.

Density and concentration

The preferred unit of density is number per Å³, corresponding to a concentration of approximately 1660.5391 mol L⁻¹ or molar (M).

Temperature

The preferred unit of temperature is Kelvin (K).

Charge

The preferred unit of charge is $e_c = 1.602176634 \times 10^{-19} \text{ C}$. The following number is often useful: $N_A e_c = 9.64853321233 \times 10^4 \text{ C mol}^{-1}$.

Energy

The preferred unit of energy is $k_B T$ or RT where

- Boltzmann's constant: $k_B = 1.38064852 \times 10^{-23} \text{ J K}^{-1}$
- Gas constant: $R = N_A k_B = 8.31446261815324 \text{ J K}^{-1} \text{ mol}^{-1}$
- Temperature: T

If $T \approx 298 \text{ K}$, then $RT \approx 2.49 \text{ kJ}$.

Surface tension

The preferred unit for surface tension is $\text{kJ mol}^{-1} \text{\AA}^{-2}$. Values for the surface tension of water in these models often range from 0.105 to 0.301 $\text{kJ mol}^{-1} \text{\AA}^{-2}$.¹ However, these values can vary significantly depending on the model used.²³

Pressure

The preferred unit for pressure is $\text{kJ mol}^{-1} \text{\AA}^{-3}$. Values for the surface tension of water in these models vary significantly depending on the model used (e.g., between 0.0004 and 0.146 $\text{kJ mol}^{-1} \text{\AA}^{-3}$).²³

Electrostatic potential

The preferred unit of electrostatic potential is $k_B T e_c^{-1}$ or $RT e_c^{-1}$. If $T \approx 298 \text{ K}$, then $k_B T e_c^{-1} = RT e_c^{-1} \approx 0.0256 \text{ J C}^{-1} = 25.6 \text{ mV}$.

2.3 Solvation model background

2.3.1 Solvation models

Electrostatic and solvation models can be roughly divided into two classes ([Warshe12006], [Roux1999], [Ren2012]) explicit solvent models that treat the solvent in atomic detail and implicit solvent models that generally replace the explicit solvent with a dielectric continuum. Each method has its strengths and weaknesses. While explicit solvent models offer some of the highest levels of detail, they generally require extensive sampling to converge properties of interest. On the other hand, implicit solvent models trade detail and some accuracy for the “pre-equilibration” of solvent degrees of freedom and elimination of sampling for these degrees of freedom. Implicit solvent methods are popular for a variety of biomedical research problems.

The polar solvation energy is generally associated with a difference in charging free energies in vacuum and solvent. A variety of implicit solvent models are available to biomedical researchers to describe polar solvation; however, the most widely-used methods are currently the Generalized Born (GB) and Poisson-Boltzmann (PB) models. GB and related methods are very fast heuristic models for estimating the polar solvation energies of biomolecular structures and therefore are often used in high-throughput applications such as molecular dynamics simulations. PB methods can be formally derived from more detailed theories and offer a somewhat slower, but often more accurate, method for evaluating polar solvation properties and often serve as the basis for parameterization and testing of GB methods. Finally, unlike most GB methods, PB models provide a global solution for the electrostatic potential and field within and around a biomolecule, therefore making them uniquely suited to visualization and other structural analyses, diffusion simulations, and a number of other methods which require global electrostatic properties.

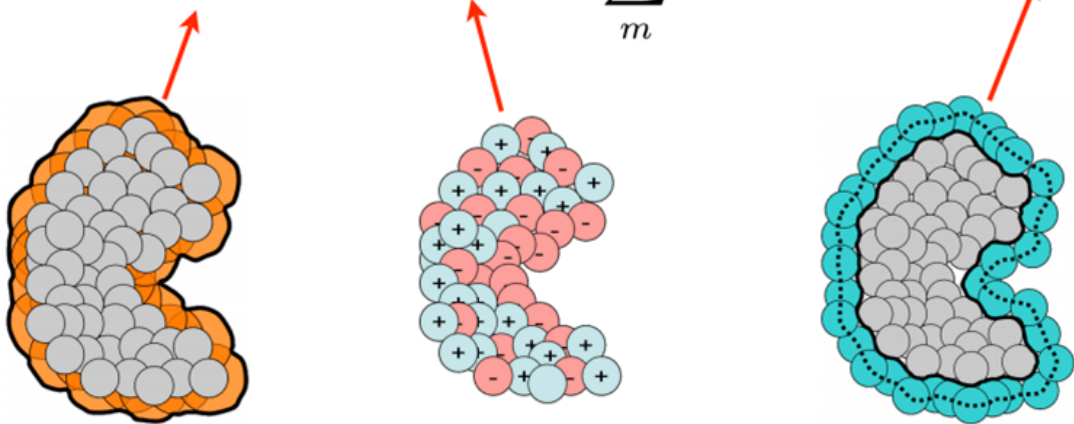
The PB equation ([Fogolari2002], [Lamm2003], [Grochowski2007], [Baker2005]) is a nonlinear elliptic partial differential equation of the form shown in the figure below which is solved for the electrostatic potential. The coefficients of this equation are directly related to the molecular structure of the system under consideration. PB theory is approximate and, as a result, has several well-known limitations which can affect its accuracy ([Grochowski2007], [Netz2000]), particularly for strongly charged systems or high salt concentrations. However, despite these limitations, PB methods are still very important for biomolecular structural analysis, modeling, and simulation. Furthermore, these limitations

¹ Sharp KA, Nicholls A, Fine RF, Honig B. Reconciling the magnitude of the microscopic and macroscopic hydrophobic effects. *Science*, 252, 106-109, 1991. DOI:10.1126/science.2011744.

² Thomas DG, Chun J, Zhen C, Wei GW, Baker NA. Parameterization of a geometric flow implicit solvation model. *J Comput Chem*, 34, 687-695, 2013. DOI:10.1002/jcc.23181.

³ Wagoner JA and Baker NA. Assessing implicit models for nonpolar mean solvation forces: The importance of dispersion and volume terms. *Proc Natl Acad Sci USA*, 103, 8331-9336, 2006. DOI:10.1073/pnas.0600118103.

are currently being addressed through new implicit solvent models and hybrid treatments which extend the applicability of PB theory while preserving some of its computational efficiency. There are currently examples of both types of treatments which leverage APBS ([Azura2006], [Chu2007], [Vitalis2004]).

$$-\nabla \cdot \epsilon(\mathbf{x}) \nabla \phi(\mathbf{x}) = \rho_f(\mathbf{x}) + \sum_m q_m c_m e^{-\beta[q_m \phi(\mathbf{x}) + V_m(\mathbf{x})]}$$


$$G[\phi] = \frac{1}{4\pi} \int_{\Omega} \left\{ \rho_f(\mathbf{x}) \phi(\mathbf{x}) - \frac{\epsilon(\mathbf{x})}{2} [\nabla \phi(\mathbf{x})]^2 + \sum_m c_m e^{-\beta V_m(\mathbf{x})} [e^{-\beta q_m \phi(\mathbf{x})} - 1] \right\} d\mathbf{x}$$

$$\mathbf{F}_i[\phi] = -\frac{\partial G[\phi]}{\partial \mathbf{r}_i} = -\frac{1}{4\pi} \int_{\Omega} \left\{ \frac{\partial \rho_f(\mathbf{x})}{\partial \mathbf{r}_i} \phi(\mathbf{x}) - \frac{1}{2} \frac{\partial \epsilon(\mathbf{x})}{\partial \mathbf{r}_i} [\nabla \phi(\mathbf{x})]^2 + \sum_m c_m \frac{\partial e^{-\beta V_m(\mathbf{x})}}{\partial \mathbf{r}_i} [e^{-\beta q_m \phi(\mathbf{x})} - 1] \right\} d\mathbf{x}$$

Reaction
field

Dielectric
boundary

"Osmotic"

PB methods provide polar solvation energies and therefore must be complemented by non-polar solvation models to provide a complete view of biomolecular solvent-solute interactions. non-polar solvation is generally associated with the insertion of the uncharged solute into solvent. There are many non-polar solvation models available; however, work by Levy et al. [Levy2003] as well as our own research [Wagoner2006] has demonstrated the importance of non-polar implicit solvent models which include treatment of attractive solute-solvent dispersion terms. This model has been implemented in APBS and can also be easily transformed into simpler popular non-polar models (e.g., solvent-accessible surface area). While this model can be used separately from PB to analyze non-polar contributions to solvation energy, its preferred use is coupled to the PB equation through a geometric flow model [Chen2010] which treats polar and non-polar interactions in the same framework and reduces the number of user-specified empirical parameters.

2.3.2 Caveats and sources of error

Model error

When performing solvation calculations using APBS, it is important to keep in mind that you are using an approximate model for solvation. Therefore, your answers may contain errors related to approximations in the model. Many review articles have covered the nature of these approximations, we will stress the highlights below.

Linear dielectric response

The Poisson-Boltzmann equation models the solvent as a dielectric continuum that responds linearly to all applied fields. In particular, under this model, very strong fields can induce unrealistically strong polarization in the dielectric

media representing the solvent and/or the solute interior. However, molecular solvents or solutes cannot support an infinite amount of polarization: they are limited by their density, their finite dipole moments, and their finite degree of electronic polarizability. Therefore, the continuum model assumption of linear dielectric response can break down in situations with strong electric fields; e.g., around nucleic acids or very highly-charged proteins.

Local dielectric response

The Poisson-Boltzmann equation models the solvent as a dielectric continuum that also responds locally to all applied fields. In other words, under this model, the local polarization at a point x is only dependent on the field at point x . However, molecular solvents and solutes clearly don't obey this assumption: the variety of covalent, steric, and other non-bonded intra- and inter-molecular interactions ensures that the polarization at point x is dependent on solute-field interactions in a non-vanishing neighborhood around x . One way to limit the impact of this flawed assumption, is to model solute response as “explicitly” as possible in your continuum electrostatics problems. In other words, rather than relying upon the continuum model to reproduce conformational relaxation or response in your solute, model such response in detail through molecular simulations or other conformational sampling.

Ambiguity of dielectric interfaces and coefficient values

Violation of the assumptions of linear and local dielectric response in real molecular systems leads to serious ambiguity in the definition of the dielectric coefficient in the Poisson-Boltzmann equation. In particular, while the values for bulk solvent (i.e., far away from the solute) response are well-defined, all other values of the dielectric coefficient are ambiguous. In general, continuum models assume a constant low-dielectric value inside the solute and the bulk solvent value outside the solute. This assumption creates tremendous sensitivity of calculation results on the placement of the dielectric interface (usually determined by solute atomic radii) and the specific value of the internal solute dielectric. In general, errors arising from this assumption can be minimized by using internal dielectric values that are consistent with the solute atomic radii parameterization.

No specific ion-solvent or ion-solute interactions

Most Poisson-Boltzmann models assume that ions do not interact directly with the solvent: they are charges embedded in the same dielectric material as the bulk solvent. This assumption implies that ions experience no “desolvation” penalty as they interact with the solute surface. Additionally, most Poisson-Boltzmann models assume that ions interact with the solute only through electrostatic and hard-sphere steric potentials. However, this assumption neglects some of the subtlety of ion-protein interactions; in particular, dispersive interactions that can possibly lead to some degree of ion specificity.

Mean field ion behavior

Finally, the Poisson-Boltzmann model is a “mean field” description of ionic solutions. This means that ions only experience the average influence of other ions in the system; the model neglects fluctuations in the ionic atmosphere and correlations between the ions in solution. Such correlations and fluctuations can be very important at high ionic charge densities; e.g., for multivalent ions, high ion concentrations, or the high-density ionic regions near highly-charged biomolecules.

Parameter set errors

Todo: Under construction; please see <https://arxiv.org/abs/1705.10035> for an initial discussion. Saved as issue <https://github.com/Electrostatics/apbs/issues/481>

Structure-based errors

Electrostatics calculations can be very sensitive to errors in the structure, including:

- Misplaced atoms or sidechains
- Missing regions of biomolecular structure
- Incorrect titration state assignments

Of these errors, incorrect titration states are the most common and, often, the most problematic. The software package PDB2PQR was created to minimize all of the above problems and we recommend its use to “pre-process” structures before electrostatics calculations.

Discretization error

The Poisson-Boltzmann partial differential equation must be discretized in order to be solved on a computer. APBS discretizes the equation in spacing by evaluating the problem coefficients and solving for the electrostatic potential on a set of grid (finite difference) or mesh (finite element) points. However, this discretization is an approximation to the actual, continuously-specified problem coefficients. Coarser discretization of coefficients and the solution reduce the overall accuracy and introduce errors into the final potential and calculated energies.

It is very important to evaluate the sensitivity of your calculated energies to the grid spacings and lengths. In general, it is a good idea to scan a range of grid spacings and lengths before starting a problem and choose the largest problem domain with the smallest grid spacing that gives consistent results (e.g., results that don’t change as you further reduce the grid spacing).

Solver and round-off error

APBS uses iterative solvers to solve the nonlinear algebraic equations resulting from the discretized Poisson-Boltzmann equation. Iterative solvers obtain solutions to algebraic equations which are accurate within a specified error tolerance. Current versions of APBS use a fixed error tolerance of 10^{-6} which implies approximately 1 part per million root-mean-squared error in calculated potentials. Such error tolerances have been empirically observed to give good accuracy in the calculated energies obtained with APBS.

However, it is important to note that the error in potential does not necessarily directly relate to the error in the energies calculated by APBS. In particular, most meaningful energies are calculated as differences between energies from several calculations. While the accuracy of each separate energy can be related to the solver error tolerance, the energy difference can only be loosely bounded by the error tolerance.

This issue is illustrated in the protein kinase ligand binding example provided with APBS as `pka-lig` and analyzed below. This example demonstrates that, while the errors for each calculation remain small, the overall error in the computed energy can be very large; particularly when two different methods are compared.

Table 2: Sensitivity of PB energies to iterative solver error tolerance (APBS 1.2)

Error tolerance	Protein energy	Protein energy relative error (with respect to 10^{-12} tolerance)	Ligand energy	Ligand energy relative error (with respect to 10^{-12} tolerance)	Complex energy	Complex energy relative error (with respect to 10^{-12} tolerance)	Binding energy	Binding energy relative error (with respect to 10^{-12} tolerance)
1.00E-06	3.01E+05	1.7E-08	1.05E+04	1.2E-08	3.11E+05	5.4E-08	8.08E+00	7.5E-06
1.00E-09	3.01E+05	1.9E-11	1.05E+04	1.1E-11	3.11E+05	5.4E-08	8.08E+00	8E-09
1.00E-12	3.01E+05	0E+00	1.05E+04	0E+00	3.11E+05	0E+00	8.08E+00	0E+00

2.3.3 Further reading

2.4 Supporting APBS

2.4.1 Please register as a user!

Please help ensure continued support for APBS-PDB2PQR by [registering your use of our software](#).

2.4.2 Citing our software

If you use APBS in your research, please cite one or more of the following papers:

- Jurrus E, Engel D, Star K, Monson K, Brandi J, Felberg LE, Brookes DH, Wilson L, Chen J, Liles K, Chun M, Li P, Gohara DW, Dolinsky T, Konecny R, Koes DR, Nielsen JE, Head-Gordon T, Geng W, Krasny R, Wei G-W, Holst MJ, McCammon JA, Baker NA. Improvements to the APBS biomolecular solvation software suite. *Protein Sci*, 27 (1), 112-128, 2018. <https://doi.org/10.1002/pro.3280>
- R. Bank and M. Holst, A New Paradigm for Parallel Adaptive Meshing Algorithms. *SIAM Review* 45, 291-323, 2003. <http://epubs.siam.org/doi/abs/10.1137/S003614450342061>
- Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. USA* 98, 10037-10041 2001. <http://www.pnas.org/content/98/18/10037>
- M. Holst, Adaptive numerical treatment of elliptic systems on manifolds. *Advances in Computational Mathematics* 15, 139-191, 2001 <http://dx.doi.org/10.1023/A:1014246117321>
- M. Holst and F. Saied, Numerical solution of the nonlinear Poisson-Boltzmann equation: Developing more robust and efficient methods. *J. Comput. Chem.* 16, 337-364, 1995.
- M. Holst and F. Saied, Multigrid solution of the Poisson-Boltzmann equation. *J. Comput. Chem.* 14, 105-113, 1993.

2.4.3 Supporting organizations

The PDB2PQR authors would like to give special thanks to the supporting organizations behind the APBS and PDB2PQR software:

National Institutes of Health Primary source of funding for APBS via grant GM069702

National Biomedical Computation Resource Deployment and computational resources support from 2002 to 2020

National Partnership for Advanced Computational Infrastructure Funding and computational resources

Washington University in St. Louis Start-up funding

2.5 Getting help

2.5.1 GitHub issues

Our preferred mechanism for user questions and feedback is via [GitHub issues](#). We monitor these issues daily and usually respond within a few days.

2.5.2 Announcements

Announcements about updates to the APBS-PDB2PQR software and related news are available through our [mailing list](#); please [register for updates](#).

2.5.3 Contacting the authors

If all else fails, feel free to contact nathanandrewbaker@gmail.com.

2.6 Further reading

2.6.1 General solvation reviews

- Baker NA. Poisson-Boltzmann methods for biomolecular electrostatics. *Methods in Enzymology*, 383, 94-118, 2004. <http://www.sciencedirect.com/science/article/pii/S0076687904830052>
- Baker NA, McCammon JA. Electrostatic interactions. *Structural Bioinformatics*. Weissig H, Bourne PE, eds., 2005. <http://dx.doi.org/10.1002/0471721204.ch21>
- Baker NA. Biomolecular applications of Poisson-Boltzmann methods. *Reviews in Computational Chemistry*. Lipkowitz KB, Larter R, Cundari TR., 21, 2005. <http://dx.doi.org/10.1002/0471720895.ch5>
- Baker NA. Improving implicit solvent simulations: a Poisson-centric view. *Curr Opin Struct Biol*, 15, 137-43, 2005. <http://dx.doi.org/10.1016/j.sbi.2005.02.001>
- Baker NA, Bashford D, Case DA. Implicit solvent electrostatics in biomolecular simulation. *New Algorithms for Macromolecular Simulation*. Leimkuhler B, Chipot C, Elber R, Laaksonen A, Mark A, Schlick T, Schutte C, Skeel R, eds., 2006. http://dx.doi.org/10.1007/3-540-31618-3_15
- Dong F, Olsen B, Baker NA. Computational Methods for Biomolecular Electrostatics. *Methods in Cell Biology: Biophysical Tools for Biologists*, 84, 843-870, 2008. <http://www.sciencedirect.com/science/article/pii/S0091679X0784026X>
- Ren P, Chun J, Thomas DG, Schnieders MJ, Marucho M, Zhang J, Baker NA. Biomolecular electrostatics and solvation: a computational perspective. *Quart Rev Biophys*, 45 (4), 427-491, 2012. <http://dx.doi.org/10.1017/S003358351200011X>

2.6.2 APBS parallel solvers

- Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc Natl Acad Sci USA*, 98, 10037-41, 2001. <http://dx.doi.org/10.1073/pnas.181342398>
- Baker NA, Sept D, Holst MJ, McCammon JA. The adaptive multilevel finite element solution of the Poisson-Boltzmann equation on massively parallel computers. *IBM J Res Devel*, 45, 427-38, 2001. <http://dx.doi.org/10.1147/rd.453.0427>

2.6.3 APBS multigrid solver

- M. Holst, Adaptive numerical treatment of elliptic systems on manifolds. *Advances in Computational Mathematics* 15, 139-191, 2001 <http://dx.doi.org/10.1023/A:1014246117321>
- M. Holst and F. Saied, Numerical solution of the nonlinear Poisson-Boltzmann equation: Developing more robust and efficient methods. *J. Comput. Chem.* 16, 337-364, 1995.
- M. Holst and F. Saied, Multigrid solution of the Poisson-Boltzmann equation. *J. Comput. Chem.* 14, 105-113, 1993.

2.6.4 APBS finite element solver

- Holst M, Baker NA, Wang F. Adaptive multilevel finite element solution of the Poisson-Boltzmann equation I: algorithms and examples. *J Comput Chem*, 21, 1319-42, 2000. <http://bit.ly/1goFAFE>
- Baker N, Holst M, Wang F. Adaptive multilevel finite element solution of the Poisson-Boltzmann equation II: refinement schemes based on solvent accessible surfaces. *J Comput Chem*, 21, 1343-52, 2000. <http://bit.ly/1dNSP8l>

2.6.5 APBS geometric flow solver

- Chen Z, Baker NA, Wei GW. Differential geometry based solvation model I: Eulerian formulation, *J Comput Phys*, 229, 8231-58, 2010. <http://dx.doi.org/10.1016/j.jcp.2010.06.036>
- Chen Z, Baker NA, Wei GW. Differential geometry based solvation model II: Lagrangian formulation. *J Math Biol*, 63, 1139-1200, 2011. <http://dx.doi.org/10.1007/s00285-011-0402-z>
- Chen Z, Zhao S, Chun J, Thomas DG, Baker NA, Wei GW. Variational approach for nonpolar solvation analysis. *Journal of Chemical Physics*, 137, 084101, 2012. <http://dx.doi.org/10.1063/1.4745084>
- Thomas DG, Chun J, Chen Z, Wei G, Baker NA. Parameterization of a Geometric flow implicit solvation model. *Journal of Computational Chemistry*, 34, 687-95, 2013. <http://dx.doi.org/10.1002/jcc.23181>
- Daily M, Chun J, Heredia-Langner A, Baker NA. Origin of parameter degeneracy and molecular shape relationships in geometric-flow calculations of solvation free energies. *J Chem Phys*, 139, 204108, 2013. <http://dx.doi.org/10.1063/1.4832900>

2.6.6 TABI-PB boundary element solver

- Geng W, Krasny R. A treecode-accelerated boundary integral Poisson-Boltzmann solver for electrostatics of solvated biomolecules, *J Comput Phys*, 247, 62-78, 2013. <https://doi.org/10.1016/j.jcp.2013.03.056>

2.6.7 Structural bioinformatics based on electrostatic properties

- Zhang X, Bajaj CL, Kwon B, Dolinsky TJ, Nielsen JE, Baker NA. Application of new multi-resolution methods for the comparison of biomolecular electrostatic properties in the absence of global structural similarity. *Multiscale Model Simul*, 5, 1196-213, 2006. <http://dx.doi.org/10.1137/050647670>
- Chakraborty S, Rao BJ, Baker N, Ásgeirsson B. Structural phylogeny by profile extraction and multiple superimposition using electrostatic congruence as a discriminator. *Intrinsically Disordered Proteins*, 1 (1), e25463, 2013. <https://www.landesbioscience.com/journals/idp/article/25463/>

2.6.8 Other fun with APBS

- Wagoner JA, Baker NA. Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms. *Proc Natl Acad Sci USA*, 103, 8331-6, 2006. <http://dx.doi.org/10.1073/pnas.0600118103>
- Swanson JMJ, Wagoner JA, Baker NA, McCammon JA. Optimizing the Poisson dielectric boundary with explicit solvent forces and energies: lessons learned with atom-centered dielectric functions. *J Chem Theory Comput*, 3, 170-83, 2007. <http://dx.doi.org/10.1021/ct600216k>
- Schnieders MJ, Baker NA, Ren P, Ponder JW. Polarizable Atomic Multipole Solutes in a Poisson-Boltzmann Continuum. *J Chem Phys*, 126, 124114, 2007. <http://dx.doi.org/10.1063/1.2714528>
- Callenberg KM, Choudhary OP, de Forest GL, Gohara DW, Baker NA, Grabe M. APBSmem: A graphical interface for electrostatic calculations at the membrane. *PLoS ONE*, 5, e12722, 2010. <http://dx.doi.org/10.1371/journal.pone.0012722>
- Unni S, Huang Y, Hanson RM, Tobias M, Krishnan S, Li WW, Nielsen JE, Baker NA. Web servers and services for electrostatics calculations with APBS and PDB2PQR. *J Comput Chem*, 32 (7), 1488-1491, 2011. <http://dx.doi.org/10.1002/jcc.21720>
- Konecny R, Baker NA, McCammon JA. iAPBS: a programming interface to the adaptive Poisson-Boltzmann solver. *Computational Science and Discovery*, 5, 015005, 2012. <http://dx.doi.org/10.1088/1749-4699/5/1/015005>
- Jurrus E, Engel D, Star K, Monson K, Brandi J, Felberg LE, Brookes DH, Wilson L, Chen J, Liles K, Chun M, Li P, Gohara DW, Dolinsky T, Konecny R, Koes DR, Nielsen JE, Head-Gordon T, Geng W, Krasny R, Wei G-W, Holst MJ, McCammon JA, Baker NA. Improvements to the APBS biomolecular solvation software suite. *Protein Sci*, 27 (1), 112-128, 2018. <https://doi.org/10.1002/pro.3280>
- Laureanti J, Brandi J, Offor E, Engel D, Rallo R, Ginovska B, Martinez X, Baaden M, Baker NA. Visualizing biomolecular electrostatics in virtual reality with UnityMol-APBS. *Protein Sci*, 29 (1), 237-246, 2020. <https://doi.org/10.1002/pro.3773>

2.7 File formats

2.7.1 Mesh and scalar data formats

OpenDX scalar data format

We output most discretized scalar data (e.g., potential, accessibility, etc.) from APBS in the data format used by the OpenDX software package. The OpenDX data format is very flexible; the following sections describe the application of this format for APBS multigrid and finite element datasets.

The multigrid data format has the following form:


```

object 1 class gridpositions counts nx ny nz
origin xmin ymin zmin
delta hx 0.0 0.0
delta 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 items 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`

```

The variables in this format include:

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; $n = nx * ny * nz$

u(*,*,*) The data values, ordered with the z-index increasing most quickly, followed by the y-index, and then the x-index.

For finite element solutions, the OpenDX format takes the following form:

```

object 1 class array type float rank 1 shape 3 items N
v1x v1y v1z
v2x v2y v2z
...
vNx vNy vNz
object 2 class array type int rank 1 shape 4 items M
s1a s1b s1c s1d
s2a s2b s2c s2d
...
sMa sMb sMc sMd
attribute "element type" string "tetrahedra"
object 3 class array type float rank 0 items N
u1
u2
...
uN
attribute "dep" string "positions"
object "irregular positions irregular connections" class field
component "positions" value 1
component "connections" value 2
component "data" value 3
end

```


where the variables in this format are:

N Number of vertices

vix viy viz Coordinates of vertex *i*

M Number of simplices

sia sib sic sid IDs of vertices in simplex *i*

ui Data value associated with vertex *i*

MCSF mesh format

APBS reads and writes meshes in the *FETk MCSF format* <<http://fetc.org/codes/mc/>>.

UHBD scalar data format

We also support scalar data output in the legacy “UHBD format” for use with programs such as [UHBD](#) and [SDA](#).

UHBD data is written in the format:

```
/* Write out the header */
Vio_printf(sock, "%72s\n", title);
Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
    nz, 1, nz);
Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
    hx, (xmin-hx), (ymin-hx), (zmin-hx));
Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);

/* Write out the entries */
icol = 0;
for (k=0; k<nz; k++) {
    Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
    icol = 0;
    for (j=0; j<ny; j++) {
        for (i=0; i<nx; i++) {
            u = k*(nx)*(ny)+j*(nx)+i;
            icol++;
            Vio_printf(sock, " %12.5e", thee->data[u]);
            if (icol == 6) {
                icol = 0;
                Vio_printf(sock, "\n");
            }
        }
    }
}
```

2.7.2 Molecular structure formats

PQR molecular structure format

This format is a modification of the PDB format which allows users to add charge and radius parameters to existing PDB data while keeping it in a format amenable to visualization with standard molecular graphics programs. The

origins of the PQR format are somewhat uncertain, but has been used by several computational biology software programs, including MEAD and AutoDock. UHBD uses a very similar format called QCD.

APBS reads very loosely-formatted PQR files: all fields are whitespace-delimited rather than the strict column formatting mandated by the PDB format. This more liberal formatting allows coordinates which are larger/smaller than ± 999 Å. APBS reads data on a per-line basis from PQR files using the following format::

```
Field_name Atom_number Atom_name Residue_name Chain_ID Residue_number X Y Z Charge_
↔Radius
```

where the whitespace is the most important feature of this format. The fields are:

Field_name A string which specifies the type of PQR entry and should either be ATOM or HETATM in order to be parsed by APBS.

Atom_number An integer which provides the atom index.

Atom_name A string which provides the atom name.

Residue_name A string which provides the residue name.

Chain_ID An optional string which provides the chain ID of the atom. Note that chain ID support is a new feature of APBS 0.5.0 and later versions.

Residue_number An integer which provides the residue index.

X Y Z 3 floats which provide the atomic coordinates (in Å)

Charge A float which provides the atomic charge (in electrons).

Radius A float which provides the atomic radius (in Å).

Clearly, this format can deviate wildly from PDB due to the use of whitespaces rather than specific column widths and alignments. This deviation can be particularly significant when large coordinate values are used. However, in order to maintain compatibility with most molecular graphics programs, the PDB2PQR program and the utilities provided with APBS attempt to preserve the PDB format as much as possible.

PDB molecular structure format

The PDB file format is described in detail in the [Protein Data Bank documentation](#).

XML molecular structure format

The XML structure format was designed as a light-weight alternative to remediate some of the shortcomings of the flat-file format. By use of XML, issues related to extra fields in the file or columns merging together can easily be remedied. Additionally, APBS will only parse the necessary information from the XML file and will ignore all other information, so users wishing to store extra data related to a residue or atom can do so inline without affecting APBS.

This data format has the following form:

```
<roottag>
  <residue>
    <atom>
      <x>x</x>
      <y>y</y>
      <z>z</z>
      <charge>charge</charge>
      <radius>radius</radius>
    </atom>
```

(continues on next page)

(continued from previous page)

```

    ...
  </residue>
  ...
</roottag>

```

The variables in this example are:

roottag This is the root element of the XML file. The value is not important to APBS - APBS simply checks that it is closed at the end of the file.

x y z A float giving the {x, y, z}-coordinate of the atom in Å.

charge A float giving the atomic charge (in electrons).

atomradius A float giving the atomic Radius (in Å).

Note: Yes, we probably should have used [PDBML](#) instead.

2.7.3 Matrix formats

Harwell-Boeing matrix format

This is the sparse matrix output format used by APBS for analyses of the matrix operators which are constructed during PB solution. This format was implemented so matrix operators could be decomposed with SuperLU and ARPACK but this also serves as a useful general mechanism for sparse matrix input and output.

Parameter formats

APBS XML parameter format

This parameter file format has the following form:

```

<ffname>
  <residue>
    <name>resname</name>
    <atom>
      <name>atomname</name>
      <charge>atomcharge</charge>
      <radius>atomradius</radius>
      <epsilon>atomepsilon</epsilon>
    </atom>
    ...
  </residue>
  ...
</ffname>

```

The variables in this example are:

ffname The name of the forcefield. This is the root element of the XML file.

resname A string giving the residue name, as provided in the PDB file to be parameterized.

atomname A string giving the atom name, as provided in the PDB file to be parameterized.

atomcharge A float giving the atomic charge (in electrons).

atomradius A float giving the atomic Radius (in Å).

atomepsilon A float giving the Lennard-Jones well depth ϵ (in kJ/mol). This is used for the calculation of WCA energies in apolar solvation energies and forces. We assume that the Lennard-Jones potential is defined in the “AMBER style”

APBS flat-file parameter format

This parameter file format is a series of lines of the form:

Residue_name Atom_name Charge Radius Epsilon

where the whitespaces are important and denote separation between the fields. The fields here are:

Residue_name A string giving the residue name, as provided in the PDB file to be parametrized.

Atom_name A string giving the atom name, as provided in the PDB file to be parametrized.

Charge A float giving the atomic charge (in electrons).

Radius A float giving the atomic radius (in Å).

Epsilon A float giving the Lennard-Jones well depth (epsilon, in kJ/mol). This is used for the calculation of WCA energies in apolar solvation energies and forces. We assume that the Lennard-Jones potential is defined in the “AMBER style”

2.8 API reference

The **apbs** command provides a command-line interface to APBS’s functionality. It is built on classes and functions in the *apbs* module. The API (application programming interface) of *apbs* is documented here for developers who might want to directly use the APBS code.

Note: The API is still changing and there is currently no guarantee that it will remain stable between minor releases.

Todo: Add API documentation for all new Python routines

2.9 Release history

2.9.1 APBS 3.0 (May 2020)

- Binary releases may be found in [GitHub releases](#) and on [SourceForge](#).

New Features

- Poisson-Boltzmann Analytical Method (PBAM, see [Lotan & Head-Gordon](#)) and Semi-Analytical Method (PBSAM, see [Yap & Head-Gordon](#)) integrated with APBS. PBSAM is currently only available in the Linux and OS X distributions.
 - Examples are located with the APBS examples in the pbam/ and pbsam/ directories.

- Tree-Code Accelerated Boundary Integral Poisson-Boltzmann Method (TABI-PB) integrated with APBS (See [Geng & Krasny](#)).
 - Examples are located with the APBS examples in the bem/, bem-pKa/, and bem-binding-energies/ folders
 - Included NanoShaper alternative to MSMS.
 - More information and documentation may be found in the [Contributions](#) section of the APBS website
- Added binary DX format support to the appropriate APBS tools.
- Test suite amended and expanded.
- Removed hard-coded limitation to number of grid points used to determine surface accessibility.

Known Bugs and Limitations

- PBSAM not building in windows due to C standard restrictions in the Microsoft compiler implementation.

Minor Updates

- PB(S)AM now requires the key work ‘pos’ for the term argument.
- PB(S)AM ‘surf’ keyword has been replaced with the ‘usemesh’ keyword.
- PB(S)AM ‘salt’ keyword has been replaced with the ‘ion’ keyword.
- PB(S)AM dynamics parameters are no longer accepted in the ELEC section.
- PB(S)AM now has only one type of ELEC method: pb(s)am_auto.
- PB(S)AM ‘gridpts’ keyword has been replaced with ‘dime’ keyword.
- PB(S)AM ‘dx’ and ‘3dmap’ keywords are deprecated to use the ‘write’ one instead.
- BEM mesh keyword now requires method names instead of just integer values.
- GEOFLOW ELEC type has been change from ‘geoflow-auto’ to ‘geoflow’.
- Fixed miscellaneous Windows build issues.
- Update the build configurations for the Pythons libraries.

Notes

- The following are included in APBS as Git submodules:
 - [Geometric Flow](#)
 - [FETk](#)
 - [PBAM/PBSAM](#)
 - [TABI-PB](#)

2.9.2 APBS 1.5 (Oct 2016)

Dearest APBS users,

We are pleased to announce the latest release of APBS 1.5. The latest version of APBS includes several notable features and bug fixes. This release includes the addition of Poisson-Boltzmann Analytical-Method (PB-AM), Poisson-Boltzmann Semi-Analytical Method (PB-SAM) and the Treecode-Accelerated Boundary Integral Poisson-Boltzmann

method (TABI). Additionally, we have made improvements to the build system and the system tests, as well as miscellaneous bug fixes. A full change log may be found [here](#).

For help with installation, building, or running APBS, please visit <https://gitter.im/Electrostatics/help>.

We thank you for your continued support of APBS.

Sincerely,

The APBS Development Team

2.9.3 APBS 1.4.2.1 (Jan 2016)

New features

- Poisson-Boltzmann Semi-Analytical Method (PB-SAM) packaged and built with APBS.
- New Geometric flow API and improvements in speed.
- Support for BinaryDX file format.
- SOR solver added for mg-auto input file option.
- DXMath improvements.
- Test suit improvements:
 - APBS build in Travis-CI
 - Geometric Flow tests added.
 - Protein RNA tests enabled.
 - Intermediate results testing.
- Example READMEs onverted to markdown and updated with latest results.

Bug fixes

- OpenMPI (mg-para) functionality restored.
- Fixed parsing PQR files that contained records other than ATOM and HETATM.
- Geometric Flow boundary indexing bug fixed.
- Build fixes:
 - Out of source CMake build are again working.
 - Python library may be built.
 - CentOS 5 binary builds for glibc compatibility.
 - Pull requests merged.
- Removed irrelevant warning messages.

Notes

The following packages are treated as submodules in APBS:

- Geometric Flow has been moved to its own [repository](#).

- FETk has been [cloned](#) so that we could effect updates.
- PB-SAM lives here: <https://github.com/Electrostatics/PB-SAM>

Added a [chat](#) feature for users.

Known bugs

- Travis CI Linux builds are breaking because Geometric Flow relies on C++11 and Travis boxen have an old GCC that does not support C++11. This also an issue for CentOS 5.
- BEM is temporarily disabled due to build issues.
- Geometric Flow build is currently broken on Windows using Visual Studio.

2.9.4 APBS 1.4.2.0 (Jan 2016)

Binary builds

Binary releases are available.

New features

- Poisson-Boltzmann Semi-Analytical Method (PB-SAM) packaged and build with APBS.
- New Geometric flow API and improvements: <https://github.com/Electrostatics/apbs/issues/235>
- Support for BinaryDX file format: <https://github.com/Electrostatics/apbs/issues/216>
- SOR solver added for mg-auto input file option.
- DXMath improvements <https://github.com/Electrostatics/apbs/issues/168> and <https://github.com/Electrostatics/apbs/issues/216>
- Test suite improvements:
 - APBS build in Travis-CI
 - Geometric Flow test added.
 - Protein RNA test enabled <https://github.com/Electrostatics/apbs/issues/149>
 - Intermediate result testing <https://github.com/Electrostatics/apbs/issues/64>
- Example READMEs converted to markdown and updated with latest results.

Bug fixes

- OpenMPI (mg-para) functionality restored: <https://github.com/Electrostatics/apbs/issues/190>
- Fixed parsing PQR files that contained records other than ATOM and HETATM: <https://github.com/Electrostatics/apbs/issues/77> and <https://github.com/Electrostatics/apbs/issues/214>
- Geometric Flow boundary indexing bug fixed.
- Build fixes:
 - Out of source CMake build are again working.
 - Python library may be built: <https://github.com/Electrostatics/apbs/issues/372>

- CentOS 5 binary builds for glibc compability.
- Pull requests merged.
- Removed irrelevant warning messages: <https://github.com/Electrostatics/apbs/issues/378>

Notes

- The following packages are treated as submodules in APBS:
 - Geometric Flow has been moved to its own repository: https://github.com/Electrostatics/geoflow_c/
 - FETk has been cloned: <https://github.com/Electrostatics/FETK/>
 - PB-SAM lives here: <https://github.com/Electrostatics/PB-SAM/>
- Added chat feature at <https://gitter.im/Electrostatics/help/> for users.

Known bugs

- Travis-CI Linux builds are breaking because Geometric Flow relies on C++11 and Travis boxen have an old GCC that does not support C++11. This is also an issue for CentOS 5.
- BEM is temporarily disabled due to build issues.
- Geometric Flow build is currently broken on Windows using Visual Studio.

2.9.5 APBS 1.4.1 (Aug 2014)

Summary

We are pleased to announce the release of APBS 1.4.1. This was primarily a bug fix release; however, we have added a few features we'd like to highlight below. We would like to also highlight our new website, still located at: <http://www.poissonboltzmann.org>. This site is also hosted at GitHub and we hope that the new organization will make it easier for people to find the content they need. While we are still in the process of migrating some remaining content, we have added links to the previous page when needed. Thank you for your continuing support of APBS. As always, please use our mailing list to send up questions or comments about our software.

Detailed changes

- Multigrid bug fix for volumes with large problem domain.
- We have added a preliminary implementation of geometric flow.
- Finite element method support has been re-enabled.
- Migration of the APBS source tree to [GitHub](#) for better collaboration, issue tracking, and source code management.
- Improved test suite.

2.9.6 APBS 1.4.0 (Jul 2012)

Summary

We are pleased to announce the release of APBS 1.4.0. This version of APBS includes a massive rewrite to eliminate FORTRAN from the software code base to improve portability and facilitate planned optimization and parallelization activities. A more detailed list of changes is provided below. Starting with this release, we have created separate installation packages for the APBS binaries, examples, and programming documentation. This change is in response to user requests and recognition of the large size of the examples and documentation directories.

Detailed changes

- Removed FORTRAN dependency from APBS
- Direct line by line translation of all source from contrib/pmgZ
- Functions replaced and tested incrementally to ensure code congruence
- Created new subfolder src/pmgC for translated pmg library
- Created new macros for 2d, 3d matrix access
- In src/generic/apbs/vmatrix.h
- Simulate native FORTRAN 2 and 3 dimensional arrays
- Use 1-indexed, column-major ordering
- Allowed direct 1-1 translation from FORTRAN to ensure code congruence
- Added additional debugging and output macros to src/generic/apbs/vhal.h
- Added message, error message, assertion, warning, and abort macros
- Macro behavior modified by the `--enable-debug` flag for configure
- Non-error messages directed to stderr in debug, io.mc otherwise
- All error messages are directed to stdout
- In debug mode, verbose location information is provided
- Added additional flags to configure
- `--with-fetk` replaces FETK_INCLUDE, FETK_LIBRARY environment flags
- `--with-efence` enables compiling with electric fence library
- `--enable-debug` eliminates compiling optimization and includes line no info
- `--enable-profiling` adds profiling information and sets `--enable-debug`
- `--enable-verbose-debug` prints lots of function specific information

2.9.7 APBS 1.3 (Oct 2010)

New features

- Added in new read and write binary (gz) commands. Can read gzipped DX files directly.
- Added new write format to output the atomic potentials to a flat file (see atompot)
- Added new functionality for using a previously calculated potential map for a new calculation.

- Added a new program for converting delphi potential maps to OpenDX format. tools/mesh/del2dx
- Updated Doxygen manual with call/caller graphs. Replaced HTML with PDF.
- Added tools/matlab/solver with simple Matlab LPBE solver for prototyping, teaching, etc.
- Deprecated APBS XML output format.
- Deprecated nlev keyword.
- Added etol keyword, which allows user-defined error tolerance in LPBE and NPBE calculations (default errtol value is 1.0e-6).
- Added more explanatory error messages for the case in which parm keyword is missing from APBS input file for apolar calculations.
- Added a polar and apolar forces calculation example to examples/born/ .
- Added warning messages for users who try to compile APBS with `--enable-tinker` flag and run APBS stand-alone execution.
- Switched default Opal service urls from `scene.wustl.edu` to `NBCR`.
- Added a sanity check in routines.c: 'bcfl map' in the input file requires 'usemap pot' statement in the input file as well.
- Introduced `Vpmpg_size()` routine to replace `F77MGSZ` call in `vpmpg.c`
- Updated test results for APBS-1.3 release.

Bug fixes

- Modified `Vpmpg_dbForce` with some grid checking code provided by Matteo Rotter.
- Fixed a bug in `psize.py` per Michael Lerner's suggestion. The old version of `psize.py` gives wrong `cglen` and `fglen` results in special cases (e.g., all y coordinates are negative values).
- Fixed a bug in `examples/scripts/checkforces.sh`: the condition for "Passed with rounding error" is `abs(difference) < errortol`, not the other way around.
- Fixed the help string in `ApbsClient.py` .
- Fixed a bug in `Vacc_atomdSASA()`: the atom SASA needs to be reset to zero displacement after finite melement methods.
- Fixed a bug in `Vpmpg_dbForce()`: the initialization of `rtot` should appear before it is used.
- Fixed a bug in `initAPOL()`: center should be initialized before used.
- Fixed a bug in routines.c: eliminated spurious "Invalid data type for writing!" and "Invalid format for writing!" from outputs with "write atompot" statement in the input file.
- Fixed a bug in `vpmpg.c`: fixed zero potential value problem on edges and corners in non-focusing calculations.

2.9.8 APBS 1.2.1 (Dec 2009)

Bug fixes

- Added in warning into `focusFillBound` if there is a large value detected in setting the boundary conditions during a focusing calculation
- Added in a check and abort in `Vpmpg_qmEnergy` if chopped values are detected. This occurs under certain conditions for NPBE calculations where focusing cuts into a low-dielectric regions.

- Fixed a bug in Vpmg_MolIon that causes npbe based calculations to return very large energies. This occurs under certain conditions for NPBE calculations where focusing cuts into a low-dielectric regions.

2.9.9 APBS 1.2.0 (Oct 2009)

New features

- Updated NBCR opal service urls from <http://ws.nbcrc.net/opal/...> to <http://ws.nbcrc.net/opal2/...>
- Increased the number of allowed write statements from 10 to 20
- Updated inputgen.py with -potdx and -istrng options added, original modification code provided by Miguel Ortiz-Lombardía
- Added more information on PQR file parsing failures
- Added in support for OpenMP calculations for multiprocessor machines.
- Changed default Opal service from http://ws.nbcrc.net/opal2/services/APBS_1.1.0 to <http://scnc.wustl.edu:8082/opal2/services/apbs-1.2>

Modifications

- Applied Robert Konecny's patch to bin/routines.h (no need to include apbscfg.h in routines.h)

Bug fixes

- Added a remove_Valist function in Python wrapper files, to fix a memory leak problem in pdb2pka
- Fixed a bug in smooth.c: bandwidth iband, jband and kband (in grid units) should be positive integers
- Fixed a bug in psize.py: for a pqr file with no ATOM entries but only HETATM entries in it, inputgen.py should still create an APBS input file with reasonable grid lengths
- Fixed a bug in Vgrid_integrate: weight w should return to 1.0 after every i, j or k loop is finished
- Fixed a bug in routines.c, now runGB.py and main.py in tools/python/ should be working again instead of producing segfault
- Fixed a few bugs in ApbsClient.py.in related to custom-defined APBS Opal service urls, now it should be OK to use custom-defined APBS Opal service urls for PDB2PQR web server installations

2.9.10 APBS 1.1.0 (Mar 2009)

New features

- Moved APBS user guide and tutorial to MediaWiki
- Added in support for OpenMPI for parallel calculations
- Added in command line support for Opal job submissions (Code by Samir Unni)
- Allowed pathname containing spaces in input file, as long as the whole pathname is in quotes ("")
- Documented 'make test' and related features

Modifications

- Modified the function bcCalc to march through the data array linearly when setting boundary conditions. This removes duplication of grid points on the edge of the array and corners.
- Clarified documentation on the IDs assigned to input maps, PQRs, parameter files, etc.
- Updated tutorial to warn against spaces in APBS working directory path in VMD; updated user guide to warn against spaces in APBS installation path on Windows
- ‘make test’ has been reconfigured to run before issuing make install (can be run from top directory)
- Removed tools/visualization/vmd from tools directory in lieu of built-in support in VMD
- Path lengths can now be larger than 80 characters
- Expanded authorship list
- Added in ‘make test-opal’ as a post install test (run from the examples install directory)
- Added additional concentrations to protein-rna test case to better encompass experimental conditions used by Garcia-Garcia and Draper; this improves agreement with the published data

Bug fixes

- Fixed typos in User Guide (ion keyword) and clarified SMPBE keyword usage
- Fixed typo in User Guide (writemat: poission -> poisson)
- Updated psize.py with Robert’s patch to fix inconsistent assignment of fine grid numbers in some (very) rare cases
- Fixed bug with boundary condition assignment. This could potentially affect all calculations; however, probably has limited impact: many test cases gave identical results after the bug fix; the largest change in value was < 0.07%.

2.9.11 APBS 1.0.0 (Apr 2008)

New features

- Changed license to New BSD style open source license (see <http://www.opensource.org/licenses/bsd-license.php>) for more information
- Added in a feature limited version of PMG (Aqua) that reduces the memory footprint of an APBS run by 2-fold
- Modified several routines to boost the speed of APBS calculations by approximately 10% when combined with the low memory version of APBS
- Simplified parameter input for ION and SMPBE keywords (key-value pairs)
- Examples and documentation for size-modified PBE code (Vincent Chu et al)
- Added in “fast” compile-time option that uses optimized parameters for multigrid calculations
- mg-dummy calculations can be run with any number ($n > 3$) of grid points
- Updated PMG license to LGPL
- Added per-atom SASA information output from APOLAR calculations
- Added per-atom verbosity to APOLAR calculation outputs
- Ability to read-in MCSF-format finite element meshes (e.g., as produced by Holst group GAMER software)

- Updated installation instructions in user guide
- Updated inputgen.py to write out the electrostatic potential for APBS input file.

Bug fixes

- Updated tools/python/apbslib* for new NOsh functionality
- Clarified ELEC/DIME and ELEC/PDIME documentation
- Added more transparent warnings/error messages about path lengths which exceed the 80-character limit
- Fixed small typo in user guide in installation instructions
- Fixed memory leaks throughout the APBS code
- Fixed NOsh_parseREAD errors for input files with r line feeds.
- Fixed a variable setting error in the test examples
- Fixed a bug where memory usage is reported incorrectly for large allocations on 64-bit systems
- Added DTRSV to APBS-supplied BLAS to satisfy FETk SuperLU dependency
- Fixed a small bug in routines.c to print out uncharged molecule id
- Limited calculation of forces when surface maps are read in

2.9.12 APBS 0.5.1 (Jul 2007)

New features

- Replaced APOLAR->glen and APOLAR->dime keywords with APOLAR->grid
- Deprecated mergedx. Added mergedx2
 - mergedx2 takes the bounding box that a user wishes to calculate a map for, as well as a resolution of the output map. An output map meeting those specifications is calculated and store.
- Added pKa tutorial
- Added warning about strange grid settings (MGparm)
- Fixed a bug in vpmg.c that occurred when a user supplied a dielectric map with the ionic strength set to zero, causing the map to not be used.
- Removed deprecated (as of 0.5.0) tools/manip/acc in lieu of new APOLAR APBS features
- Added enumerations for return codes, new PBE solver (SMPBE) and linear/ nonlinear types
- Added in code for Size-Modified PBE (SMPBE)

Bug fixes and API changes

- Fixed buffer over-run problem
- Fixed case inconsistency with inputgen.py and psize.py scripts which caused problems with some versions of Python

- Fixed bug wherein ‘bcfl sdh’ behaved essentially like ‘bcfl zero’. Now we have the correct behavior: ‘bcfl sdh’ behaves like ‘bcfl mdh’ thanks to the multipole code added by Mike Schnieders. Interestingly, this bug didn’t have a major on the large-molecule test cases/examples provided by APBS but did affect the small molecule solvation energies. Thanks to Bradley Scott Perrin for reporting this bug.
- Added support for chain IDs in noinput.py
- Fixed bug in noinput.py where REMARK lines would cause the script to fail.

2.9.13 APBS 0.5.0 (Jan 2007)

New features

- Significantly streamlined the configure/build/install procedure:
 - Most common compiler/library options now detected by default
 - MALOC is now included as a “plugin” to simplify installation and compatibility issue
- Added new APOLAR section to input file and updated documentation – this function renders tools/manip/acc obsolete.
- Added support for standard one-character chain IDs in PQR files.
- Added a new “spl4” charge method (chgm) option to support a quintic B-spline discretization (thanks to Michael Schnieders).
- Updated psize.py
- Added a new “spl4” ion-accessibility coefficient model (srfm) option that uses a 7th order polynomial. This option provides the higher order continuity necessary for stable force calculations with atomic multipole force fields (thanks to Michael Schnieders).
- Modified the “sdh” boundary condition (bcfl) option to include dipoles and quadrupoles. Well-converged APBS calculations won’t change with the dipole and quadrupole molecular moments included in the boundary potential estimate, but calculations run with the boundary close to the solute should give an improved result (thanks to Michael Schnieders).
- Updated documentation to reflect new iAPBS features (NAMD support)
- Added Gemstone example to the tutorial
- New example demonstrating salt dependence of protein-RNA interactions.
- Added code to allow for an interface with TINKER (thanks to Michael Schnieders).
- The Python wrappers are now disabled by default. They can be compiled by passing the –enable-python flag to the configure script. This will allow users to attempt to compile the wrappers on various systems as desired.
- Added XML support for reading in parameter files when using PDB files as input. New XML files can be found in tools/conversion/param/vparam.
- Added XML support for reading “PQR” files in XML format.
- Added support for command line –version and –help flags.
- Added support for XML output options via the –output-file and –output-format flags.
- Updated runme script in ion-pmf example to use environmental variable for APBS path
- Modified the license to allow exceptions for packaging APBS binaries with several visualization programs. PMG license modified as well.

- Added a DONEUMANN macro to vfetk.c to test FEM problems with all Neumann boundary conditions (e.g., membranes).
- Added Vpmg_splineSelect to select the correct Normalization method with either cubic or quintic (7th order polynomial) spline methods.
- Modified the selection criteria in Vpmg_qfForce, Vpmg_ibForce and Vpmg_dbnpForce for use with the new spline based (spl4) method.
- Added ion-pmf to the make test suite.
- Updated splash screen to include new PMG acknowledgment
- Added runGB.py and readGB.py to the Python utilities, which calculate solvation energy based on APBS parameterized Generalized Born model.
- Updated authorship and tool documentation
- Deprecated ELEC->gamma keyword in lieu of APOLAR->gamma

Bug fixes and API changes

- Cleanup of documentation, new Gemstone example
- Clarified usage of dime in mg-para ELEC statements
- Massive cleanup of NOsh, standardizing molecule and calculation IDs and making the serial focusing procedure more robust
- Removed MGparm partOlap* data members; the parallel focusing centering is now done entirely within NOsh
- Updated the user manual and tutorial
- Updated psize.py to use centers and radii when calculating grid sizes (thanks to John Mongan)
- Fixed problems with FEM-based NPBE, LPBE, and LRPBE calculations
- Fixed a minor bug in the configure script that prevented MPI libraries from being found when using certain compilers.
- Updated acinclude.m4, aclocal.m4, config/* for new version (1.9) of automake and compatibility with new MALOC
- Fixed a bug where reading in a file in PDB format had atom IDs starting at 1 rather than 0, leading to a segmentation fault.
- Fixed a bug in mypde.f where double precision values were initialized with single precision number (causing multiplication errors).
- Fixed a bug in the FEM code. Now calls the npbe solver works properly with FEtk 1.40
- Modified the FEMParm struct to contain a new variable pkey, which is required for selecting the correct path in AM_Refine

2.9.14 APBS 0.4.0 (Dec 2005)

New features

- New version of the ‘acc’ program available.
- Added additional verbosity to APBS output.

- Added tools/python/vgrid to the autoconf script. The directory compiles with the rest of the Python utilities and is used for manipulating dx files.
- Modified the tools/python/noinput.py example to show the ability to get and print energy and force vectors directly into Python arrays.
- Added dx2uhbd tool to tools/mesh for converting from dx format to UHBD format (Thanks to Robert Konecny)
- Added ability of tools/manip/inputgen.py to split a single mg-para APBS input file into multiple asynchronous input files.
- Modified inputgen.py to be more flexible for developers wishing to directly interface with APBS.
- Added Vclist cell list class to replace internal hash table in Vacc
- Modified Vacc class to use Vclist, including changes to the Vacc interface (and required changes throughout the code)
- Consolidated Vpmg_ctor and Vpmg_ctorFocus into Vpmg_ctor
- Consolidated vpmg.c, vpmg-force.c, vpmg-energy.c, vpmg-setup.c
- Added autoconf support for compilation on the MinGW32 Windows Environment
- Added autoconf support (with Python) for Mac OS 10.4 (Tiger)
- Added the function Vpmg_solveLaplace to solve homogeneous versions of Poisson's equation using Laplacian eigenfunctions.
- Modified the dielectric smoothing algorithm (srfm smol) to a 9 point method based on Brucoleri, et al. J Comput Chem 18 268-276 (1997). NOTE: This is a faster and more flexible smoothing method. However, when combined with the the molecular surface bugfixes listed below, this change has the potential to make the srfm smol method give very different results from what was calculated in APBS 0.3.2. Users who need backwards compatibility are encouraged to use the spline based smoothing method (srfm spl2) or the molecular surface without smoothing (srfm mol).
- Added new 'sdens' input keyword to allow user to control the sphere density used in Vacc. This became necessary due to the Vacc_molAcc bug fix listed below. Only applies to srfm mol and srfm smol.
- Made the examples directory documentation much more streamlined.
- Added tests for examples directory. Users can now issue a "make test" in the desired directory to compare local results with expected results. Also includes timing results for tests for comparison between installations.

Bug fixes

- Fixed a bug in Vpmg_qmEnergy to remove a spurious coefficient of z_i^2 from the energy calculation. This generated incorrect results for multivalent ions (but then again, the validity of the NPBE is questionable for multivalents...) (Big thanks to Vincent Chu)
- Fixed a bug in vacc.c where atoms with radii less than 1Å were not considered instead of atoms with no radii.
- Fixed error in tools/mesh/dx2mol.c (Thanks to Fred Damberger)
- Fixed floating point error which resulted in improper grid spacings for some cases.
- Fixed a bug in Vacc_molAcc which generates spurious regions of high internal dielectric for molecular surface-based dielectric definitions. These regions were very small and apparently affected energies by 1-2% (when used with the 'srfm mol'; the 'srfm smol' can potentially give larger deviations). The new version of the molecular surface is actually faster (requires 50-70% of the time for most cases) but we should all be using the spline surface anyway – right? (Thanks to John Mongan and Jessica Swanson for finding this bug).
- Fixed a bug in vpmg.c that caused an assertion error when writing out laplacian maps (Thanks to Vincent Chu).

- Ensured Vpmg::ccf was always re-initialized (in the case where the Vpmg object is being re-used).
- Removed a spurious error estimation in finite element calculations.
- Clarified the role of ccf and other variables in mypde.f and vpmg.c by expanding/revising the inline comments.

2.9.15 APBS 0.3.2 (Nov 2004)

New features

- Updated tutorial with more mg-auto examples
- Updated apbs.spec file for generating RPMs on more platforms.
- Added new Python wrapper to tools/python directory showing how to run APBS without PQR and .in inputs.
- Python wrappers are now configured to compile on more architectures/ from more compilers.
- Updated tools/conversion/pdb2pqr to a new version (0.1.0) of PDB2PQR, which now can handle nucleic acids, rebuild missing heavy atoms, add hydrogens, and perform some optimization.

Bug fixes

- The dimensions of the fine grids in the pka-lig example calculations were increased to give more reliable results (albeit ones which don't agree with the reported UHBD values as well).
- hz in mgparse.c causes name clash with AIX environmental variable; fixed.
- Fixed documentation to state that using a kappa map does not ignore ELEC ION statements.
- Added a stability fix for printing charge densities for LPBE-type calculations.
- Fixed a bug in NPBE calculations which led to incorrect charge densities and slightly modified total energies.
- Modified the origin when creating UHBD grids to match standard UHBD format.
- Fixed VASSERT error caused by rounding error when reading in dx grid files.

2.9.16 APBS 0.3.1 (Apr 2004)

New features

- New APBS tutorial
- New tools/python/vgrid/mergedx.py script to merge dx files generated from parallel APBS runs back into a single dx file.

Bug fixes

- Fixed bug in parallel calculations where atoms or points on a border between two processors were not included. Modified setup algorithm for parallel calculations to allow partitions in order to obtain grid points and spacing from the global grid information.
- Modified extEnergy function to work with parallel calculations, giving better accuracy.

2.9.17 APBS 0.3.0 (Feb 2004)

News

APBS is now supported by the NIH via NIGMS grant GM69702-01.

Changes that affect users

- New version of the documentation
- New directory structure in tools/
- Finished fe-manual mode for ELEC calculations – this is the adaptive finite element solver
- Added documentation for fe-manual
- New apbs/tools/manip/inputgen.py script to automatically generate input APBS files from PQR data
- Added new asynchronous mode in mg-para parallel calculations to enable running on-demand and/or limited resources
- Added new script (tools/manip/async.sh) to convert mg-para calculations in mg-async calculations
- Added following aliases for some of the more obscure parameters in the input files:
 - chgm 0 ==> chgm spl0
 - chgm 1 ==> chgm spl2
 - srfm 0 ==> srfm mol
 - srfm 1 ==> srfm smol
 - srfm 2 ==> srfm spl2
 - bcfl 0 ==> bcfl zero
 - bcfl 1 ==> bcfl sdh
 - bcfl 2 ==> bcfl mdh
 - bcfl 4 ==> bcfl focus
 - calcenergy 0 ==> calcenergy no
 - calcenergy 1 ==> calcenergy total
 - calcenergy 2 ==> calcenergy comps
 - calcforce 0 ==> calcforce no
 - calcforce 1 ==> calcforce total
 - calcforce 2 ==> calcforce comps
- Example input files have been updated to reflect this change. NOTE: the code is backward-compliant; i.e., old input files WILL still work.
- Added new READ options “PARM” and “MOL PDB”, see documentation for more information. These options allow users to use unparameterized PDB files together with a parameter database.
- Updated the documentation
- Now include support for chain IDs and other optional fields in PQR/PDB files
- Added support for parsing PDB files

- Renamed:
- `amber2charmm` -> `amber2charmm.sh`
- `pdb2pqr` -> `pdb2pqr.awk`
- `qcd2pqr` -> `qcd2pqr.awk`
- Added a new Python-based `pdb2pqr` (`tools/conversion/pdb2pqr`) script that allows users to choose parameters from different forcefields.
- Updated Python wrappers (`tools/python`) and added the `python` directory to `autoconf/automake`.
- Reformatted `examples/README.html` for readability.

Bug fixes

- Fixed bug in PQR parsing that can cause PDB/PQR files to be mis-read when they contain residues with numbers in their names (Thanks to Robert Konecny and Joanna Trylska)
- Fixed bug when writing out number/charge density: unrealistic densities reported inside iVdW surface.
- Fixed bug in VMD `read_dx` utility
- Invalid map IDs now result in an error message instead of a core dump (thanks to Marco Berrera)
- Modified mechanism for cubic-spline output, fixing a bug associated with zero-radius atoms
- Fixed omission of `srfm` in sections of documentation (thanks to Sameer Varma)
- Made `autoconf/automake` configure setup more robust on Solaris 8 platforms (thanks to Ben Carrington)

Changes that affect developers

- New documentation setup
- New `tools/` directory structure
- Changed `Vgreen` interface and improved efficiency
- Changed `Vopot` interface to support multiple grids
- Added several norm and seminorm functions to `Vgrid` class
- Altered `-with-blas` syntax in configure scripts and removed `-disable-blas`
- Documented high-level frontend routines
- Cool new class and header-file dependency graphs courtesy of Doxygen and Graphviz
- Added substantial `mypde.c`-based functionality to `Vfetk`
- Moved `chgm` from `PBEparm` to `MGparm`
- Minor changes to `Vfetk`: removed `genIcos` and added `genCube`
- FEM solution of RPBE working again (see `test/reg-fem`) and is probably more up-to-date than `test/fem`
- Updated API documentation
- Changed many `NOsh`, `FEMparm`, `MGparm` variables to enums
- Changes to `Valist` and `Vatom` classes
- Fixed minor bugs in documentation formatting

- Made Vopot more robust
- Created Vparam class for parameter file parsing
- Added vparam* parameter database flat files to tools/conversion/param

2.9.18 APBS 0.2.6 (Jan 2003)

- Changed license to GPL
- Made a few routines compliant with ANSI X3.159-1989 by removing snprintf (compliant with ISO/IEC 9899:1999). This is basically for the sake of OSF support.

2.9.19 APBS 0.2.5 (Nov 2002)

- Improved consistency between energies evaluated with “chgm 0” and “chgm 1”
- Made charge-field energy evaluation consistent for user-supplied charge maps
- Added new psize.py script courtesy of Todd Dolinsky.
- Updated list of APBS-related tools in User Guide.
- Added RPM capabilities courtesy of Steve Bond.
- Removed annoying excess verbosity from Vgrid.
- Updated Blue Horizon compilation instructions (thanks to Robert Konecny and Giri Chukkapalli)
- Updated autoconf/automake/libtool setup and added –disable-tools option

2.9.20 APBS 0.2.4 (Oct 2002)

- Fixed bug which set one of the z-boundaries to zero for “bcfl 1”. This can perturb results for systems where the grid boundaries are particularly close to the biomolecule. While this is an embarrassing bug, most systems using settings suggested by the psize script appear largely unaffected (see examples/README.html). Thanks to Michael Grabe for finding this bug (Michael, you can stop finding bugs now...)
- Updated VMD scripts to agree with the current OpenDX output format
- A COMMENT: As far as I can tell, the current version of OpenDX-formatted output (same as version 0.2.3) is fully compliant with the OpenDX standards (see, for example, <http://opendx.npaci.edu/docs/html/pages/usrgu065.htm#HDRXMPLES>). However, I realize this is different than the format for previous versions and would encourage all users to update their APBS-based applications to accomodate these changes. The best solution would be for all downstream applications to use the APBS Vgrid class (see http://agave.wustl.edu/apbs/doc/api/html/group__Vgrid.html) to manipulate the data since this class should remain internally consistent between releases. Finally, I would love to have some OpenDX guru who uses APBS to contact me so I can solidify the data output format of APBS. I’m about ready to permanently switch to another format if I can’t reach a consensus with OpenDX...

2.9.21 APBS 0.2.3 (Oct 2002)

- Fixed bugs in salt-dependent Helmholtz/nonlinear term of PBE affecting both LPBE and NPBE calculations. While this bug fix only changes most energies by < 2 kJ/mol, it is recommended that all users upgrade. Many thanks to Michael Grabe for finding and carefully documenting this bug!

- A parameter (chgm) has been added which controls the charge discretization method used. Therefore, this version contains substantial changes in both the API and input file syntax. Specifically:
 - PBEparm has two new members (chgm, setchgm)
 - Vpmg_fillco requires another argument
 - Vpmg_*Force functions require additional arguments
 - Input files must now contain the keyword “chgm #” where # is an integer
 - Please see the documentation for more information.
- Fixed problems with “slicing” off chunks of the mesh during I/O of focused calculations
- Updated authors list
- New CHARMM parameters – Robert Konecny
- Created enumerations for common surface and charge discretization methods
- Added Vmgrid class to support easy manipulation of nested grid data
- Added more verbosity to error with NPBE forces
- Added working Python wrappers – Todd Dolinsky
- Modified VMD scripts read_dx and loadstuff.vmd

2.9.22 APBS 0.2.2 (Aug 2002)

- There were several other changes along the way... I lost track.
- Changed coordinate indexing in some energy calculations
- Updated documentation to reflect recent changes on Blue Horizon
- Improved speed of problem setup BUT NOW RESTRICT use of input coefficient maps (see documentation)
- Updated documentation, placing particular emphasis on use of Intel compilers and vendor BLAS on Intel Linux systems
- Fixed bug for nonpolar force evaluation in Vpmg_dbnpForce
- Removed MG test scripts; use `bin/*.c` for templates/testing
- Made main driver code completely memory-leak free (i.e., if you wanted to wrap it and call it repeatedly – Thanks to Robert Konecny)
- Fixed main driver code for compatibility with SGI compilers (Thanks to Fabrice Leclerc)
- Made focused evaluation of forces more sensible.
- Added ‘print force’ statement
- Fixed bug in OpenDX input/output (OpenDX documentation is lousy!)

2.9.23 APBS 0.2.1 (Apr 2002)

This version requires the latest version of MALOC to work properly!

- Syntax changes
 - The writopot and writeacc keywords have been generalized and new I/O features have been added. The syntax is now:

- * write pot dx potential
- * write smol dx surface
- * etc. Please see the User's Manual for more information
- The read keywords has been generalized and new I/O features have been added which support the use of pre-calculated coefficient grids, etc. The correct syntax for reading in a molecule is now “read mol pqr mol.pqr end”; please see the User's Manual for more information.
- The “mg” keyword is no longer supported; all input files should use “mg-manual” or one of the other alternatives instead.
- A change in the behavior of the “calcenergy” keyword; passing an argument of 2 to this keyword now prints out per-atom energies in addition to the energy component information
- A new option has been added to tools/manip/acc to give per-atom solvent-accessible surface area contributions
- A new option has been added to tools/manip/coulomb to give per-atom electrostatic energies
- Added tools/mesh/dxmath for performing arithmetic on grid-based data (i.e., adding potential results from two calculations, etc.)
- Added tools/mesh/uhbd_asc2bin for converting UHBD-format grid files from ASCII to binary (contributed by Dave Sept)
- Improvement of VMD visualization scripts (contributed by Dave Sept)
- The API has changed significantly; please see the Programmer's Manual.
- Working (but still experimental) Python wrappers for major APBS functions.
- More flexible installation capabilities (pointed out by Steve Bond)
- Added ability to use vendor-supplied BLAS
- Brought up-to-date with new MALOC

2.9.24 APBS 0.2.0 (Mar 2002)

This version is a public (beta) release candidate and includes:

- Slight modification of the user and programmer's guides
- Scripts for visualization of potential results in VMD (Contributed by Dave Sept)
- Corrections to some of the example input files
- A few additional API features

This release requires a new version of MALOC.

2.9.25 APBS 0.1.8 (Jan 2002)

This version is a public (beta) release candidate and includes the following bug-fixes:

- Added warning to parallel focusing
- Added several test cases and validated the current version of the code for all but one (see examples/README.html)
- Fixed atom partitioning bug and external energy evaluation during focusing
- Added new program for converting OpenDX format files to MOLMOL (by Jung-Hsin Lin)

You should definitely upgrade, the previous versions may produce unreliable results.

2.9.26 APBS 0.1.7 (Dec 2001)

This version is a public (beta) release candidate and includes the following bug-fixes:

- Fixed I/O for potential in UHBD format (thanks, Richard!)
- Re-arranged garbage collection routines in driver code
- Improved FORTRAN/C interfaces
- Re-configured autoconf/libtool setup for more accurate library version number reporting

2.9.27 APBS 0.1.6 (Nov 2001)

This version is a public (beta) release candidate and includes the following bug-fixes and features:

- Fixed printf formatting in UHBD potential output
- Added input file support for parallel focusing
- Fixed small bug in parsing writeacc syntax (thanks, Dave)
- Added output file support for parallel focusing
- Changed some documentation

You need to download a new version of MALOC for this release.

2.9.28 APBS 0.1.5 (Oct 2001)

This version features minor bug fixes and several new features:

- Fixed shift in center of geometry for OpenDX I/O
- Made energy evaluation more robust when using NPBE
- Rearrangements of files and modified compilation behavior
- Input file support for ion species of varying valency and concentration
- Input file support incorrect nlev/dime combinations; APBS now finds acceptable settings near to the user's requested values
- "Automatic focusing". Users now simply specify the physical parameters (temperature, dielectric, etc.), the coarse and fine grid lengths and centers, and APBS calculates the rest

2.9.29 APBS 0.1.4 (Sep 2001)

This version features major bug fixes introduced in the 0.1.3 release:

- Chain ID support has been **removed** from the PDB/PQR parser (if anyone has a nice, flexible PDB parser they'd like to contribute to the code, I'd appreciate it)
- Configure script has been made compatible with OSF
- Bug fix in disabling FEtk-specific header files

2.9.30 APBS 0.1.3 (Sep 2001)

This version features a few improvements in scripts, PDB parsing flexibility, and portability, including:

- Dave Sept upgraded the psize and shift scripts to allow more flexibility in PDB formats.
- Chain ID support has been added to the PDB/PQR parser
- Removed -g from compiler flags during linking of C and FORTAN under OSF (thanks to Dagmar Floeck and Julie Mitchell for help debugging this problem)

2.9.31 APBS 0.1.2 (Sep 2001)

This version is mainly designed to increase portability by switching to libtool for library creation and linking. Of course, it also contains a few bug fixes. Highlights include:

- Changes to the User Manual
- Addition of a Programmer's Manual
- Various FETk-related things (no particular impact to the user)
- Improvements to the test systems
- Change in the format for printing energies
- Change in directory structure
- Fixed centering bug in main driver (only impacted I/o)
- Fixed error message bug in VPMG class
- Fixed grid length bug (popped up during sanity checks) in VPMG class
- Switched to libtool for linking
- Note that Compaq Tru64 Alpha users may still experience problems while compiling due to some strangeness with linking C and FORTRAN objects.

2.9.32 APBS 0.1.1 (Aug 2001)

I am slightly less pleased to announce the first bug-fix for APBS, version 0.1.1. This fixes compilation problems that popped up for several folks, including:

- Syntax errors with non-GNU compilers
- Errors in the installation instructions
- Installation of binary in machine-specific directory

2.9.33 APBS 0.1.0 (Aug 2001)

I am pleased to announce the availability of a pre-beta version of the Adaptive Poisson-Boltzmann Solver (APBS) code to selected research groups. APBS is new software designed to solve the Poisson-Boltzmann equation for very large biomolecular systems. For more information, please visit the APBS web site at <http://mccammon.ucsd.edu/apbs>.

This release is designed to allow interested users to get familiar with the code. It is not currently fully functional; it only provides for the sequential multigrid (Cartesian mesh) solution of the linearized and nonlinear Poisson-Boltzmann equation. User-friendly parallel support will be incorporated into the next release. Other limitations that may impact its immediate usefulness are:

- No finite element support. This is awaiting the public release of the Holst group’s FEtk library.
- Somewhat inefficient coefficient evaluation (i.e., problem setup). This should be fixed in the next release or two.

Rather than serving as a production code, this release represents a request for help in breaking the software and finding its deficiencies before a public beta.

If you are interested in testing this early release, please go to <http://wasabi.ucsd.edu/~nbaker/apbs/download/>. Since this is not a public release of APBS, you will need to enter the user-name “apbs-beta” and the password “q94p\$fa!” for access to this site. Once there, please follow the instructions to download and install APBS.

If you are not interested in trying out this early release, but would like to stay informed about subsequent versions of APBS, please consider subscribing to the APBS announcements mailing list by sending the message “subscribe apbs-announce” to majordomo@mccammon.ucsd.edu.

Thank you for your time and interest in the APBS software.

2.10 Documentation “to-do” list

Todo: Add API documentation for all new Python routines

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/api/index.rst`, line 19.)

Todo: Under construction; please see <https://arxiv.org/abs/1705.10035> for an initial discussion. Saved as issue <https://github.com/Electrostatics/apbs/issues/481>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/background.rst`, line 96.)

Todo: Resolve unit confusion with geometric flow *gamma* keyword. <https://github.com/Electrostatics/apbs/issues/490>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/apolar/gamma.rst`, line 21.)

Todo: Resolve unit confusion with geometric flow *press* keyword and the apolar *press* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/499>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/apolar/press.rst`, line 22.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/3dmap.rst`, line 6.)

Todo: The PB-(S)AM 3dmap keyword should not exist; please replace it ASAP with the *write* command. Documented this todo as <https://github.com/Electrostatics/apbs/issues/482>

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/3dmap.rst, line 16.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/akeyPRE.rst, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/akeySOLVE.rst, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/async.rst, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/diff.rst, line 6.)

Todo: What are the units for dRot? Documented as <https://github.com/Electrostatics/apbs/issues/486>

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/diff.rst, line 34.)

Todo: Add a `mol id` flag rather than have an implicit ordering of the `diff` keywords. Documented in <https://github.com/Electrostatics/apbs/issues/487>

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/diff.rst, line 43.)

Todo: add manual finite difference documentation.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/dime.rst, line 14.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/domainLength.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/dx.rst`, line 6.)

Todo: The PB-(S)AM `dx` keyword should not exist; please replace it ASAP with the *write* command. Documented in <https://github.com/Electrostatics/apbs/issues/488>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/dx.rst`, line 17.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/ekey.rst`, line 6.)

Todo: add documentation links for other instances.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/etol.rst`, line 14.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/exp.rst`, line 6.)

Todo: It would be better to generalize the *READ input file section* of the input file rather than use the `exp` command. This command also needs to be cleaned up – it’s too fragile. Documented at <https://github.com/Electrostatics/apbs/issues/489>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/exp.rst`, line 19.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/fe-manual.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/gamma-geoflow.rst`, line 4.)

Todo: Resolve unit confusion with geometric flow *gamma* keyword. <https://github.com/Electrostatics/apbs/issues/490>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/gamma-geoflow.rst`, line 19.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/gcent.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/geoflow-auto.rst`, line 6.)

Todo: Add LPBE/NPBE support to geometric flow or remove the `ion` and `lpbe` keywords. Documented in <https://github.com/Electrostatics/apbs/issues/491>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/geoflow-auto.rst`, line 45.)

Todo: If there's only one mode, then we can change the keyword from `geoflow-auto` to just `geoflow`. Documented in <https://github.com/Electrostatics/apbs/issues/492>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/geoflow-auto.rst`, line 50.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/glen.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/grid2d.rst`, line 6.)

Todo: The PB-(S)AM `grid2d` keyword should not exist; please replace it ASAP with the *write* command. Documented in <https://github.com/Electrostatics/apbs/issues/493>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/grid2d.rst`, line 27.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/gridpts.rst`, line 6.)

Todo: The PB-(S)AM `gridpts` keyword should not exist; it's duplicative of the existing `dime` keyword! Documented in <https://github.com/Electrostatics/apbs/issues/494>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/gridpts.rst`, line 16.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/imat.rst`, line 6.)

Todo: It would be better to generalize the *READ input file section* of the input file rather than use the `imat` command. This command also needs to be cleaned up – it's too fragile. Documented in <https://github.com/Electrostatics/apbs/issues/495>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/imat.rst`, line 18.)

Todo: port other versions of command

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/index.rst`, line 12.)

Todo: port for other types of calculations.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/lpbe.rst`, line 17.)

Todo: port for other types of calculations.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/lrpbe.rst`, line 16.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/mac.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/maxsolve.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/maxvert.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/mesh.rst`, line 6.)

Todo: The integer flag values for `mesh` should be replaced by human-readable strings. Documented in <https://github.com/Electrostatics/apbs/issues/496>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/mesh.rst`, line 28.)

Todo: port for other types of calculations.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/npbe.rst`, line 16.)

Todo: port for other types of calculations.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/nrpbe.rst`, line 16.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/ntraj.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/ofrac.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/outdata.rst`, line 6.)

Todo: The integer flag values for `mesh` should really be replaced by human-readable strings.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/outdata.rst`, line 24.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pbam-auto.rst`, line 6.)

Todo: If there's only one mode to PBAM, let's call it `pbam` instead of `pbam-auto`. Documented in <https://github.com/Electrostatics/apbs/issues/498>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pbam-auto.rst`, line 13.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pbc.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pbsam-auto.rst`, line 6.)

Todo: If there's only one mode to PBAM, let's call it `pbsam` instead of `pbsam-auto`.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pbsam-auto.rst`, line 13.)

Todo: port for other calculation types

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pdie.rst`, line 14.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/pdime.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/press-geoflow.rst`, line 4.)

Todo: Resolve unit confusion with geometric flow `press` keyword and the apolar *press* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/499>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/press-geoflow.rst`, line 20.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/randomorient.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/runname.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/runtype.rst`, line 6.)

Todo: The dynamics part of the PB-(S)AM code should be moved out of the `ELEC` section. Documented in <https://github.com/Electrostatics/apbs/issues/500>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/runtype.rst`, line 39.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/salt.rst`, line 6.)

Todo: The PB-(S)AM `salt` keyword should be eradicated and replaced with the *ion* keyword. Documented in <https://github.com/Electrostatics/apbs/issues/501>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/salt.rst`, line 16.)

Todo: port for other calculation types

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/sdie.rst`, line 14.)

Todo: port other versions of `sr fm` to new syntax.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/srfm.rst`, line 15.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/surf.rst`, line 6.)

Todo: The PB-SAM `surf` command is redundant with and should be replaced by the existing `usemesh` command. Documented in <https://github.com/Electrostatics/apbs/issues/502>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/surf.rst`, line 17.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/tabi.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/targetNum.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/targetRes.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/term.rst`, line 6.)

Todo: Add a constant keyword (e.g., like `position`) before the `{pos}` argument of `term`. Documented in <https://github.com/Electrostatics/apbs/issues/503>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/term.rst`, line 35.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/termcombine.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/tolsp.rst`, line 6.)

Todo: We need documentation for `tolsp`. Documented in <https://github.com/Electrostatics/apbs/issues/504>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/tolsp.rst`, line 11.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/tree_n0.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/tree_order.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/units.rst`, line 6.)

Todo: It would be great to use the same units everywhere in APBS. Documented in <https://github.com/Electrostatics/apbs/issues/485>

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/units.rst`, line 27.)

Todo: Port other uses to new syntax.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/usemap.rst`, line 14.)

Todo: This command has not yet been ported to the *new APBS syntax* (see [YAML- and JSON-format input files](#)).

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/usemesh.rst`, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/vdwdisp.rst, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/write.rst, line 6.)

Todo: This command has not yet been ported to the *new APBS syntax* (see *YAML- and JSON-format input files*).

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/xyz.rst, line 6.)

Todo: It would be nice to incorporate the xyz functionality into the *READ input file section* block. Documented in <https://github.com/Electrostatics/apbs/issues/505>

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/elec/xyz.rst, line 22.)

Todo: port for other calculation types

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/generic/mol.rst, line 16.)

Todo: move other instances of this keyword to the new syntax

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/generic/swin.rst, line 14.)

Todo: add other uses to new syntax

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/old/generic/temp.rst, line 15.)

Todo: Make this more user-friendly by:

- Adding introductory text with contents
 - Only showing inherited members in API documentation
 - Only showing undocumented members in API documentation
-

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/new/calculate/finite_difference.rst, line 7.)

Todo: improve documentation with outline and/or example.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/new/calculate/index.rst, line 9.)

Todo: Provide overview and/or separate API documentation from main docs.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/new/calculate/nonpolar.rst, line 7.)

Todo: finish other required and optional sections

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/apbs/checkouts/nathan-issue_16/docs/using/input/new/index.rst, line 18.)

2.11 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

Bibliography

- [Azuara2006] Azuara C, Lindahl E, Koehl P, Orland H, and Delarue M, PDB_Hydro: incorporating dipolar solvents with variable density in the Poisson-Boltzmann treatment of macromolecule electrostatics. *Nucleic Acids Research*, 2006. 34: p. W38-W42.
- [Baker2005] Baker NA, Biomolecular Applications of Poisson-Boltzmann Methods, in *Reviews in Computational Chemistry*, Lipkowitz KB, Larter R, and Cundari TR, Editors. 2005, John Wiley and Sons.
- [Chen2010] Chen Z, Baker NA, Wei GW. Differential geometry based solvation model I: Eulerian formulation, *J Comput Phys*, 229, 8231-58, 2010.
- [Chu2007] Chu VB, Bai Y, Lipfert J, Herschlag D, and Doniach S, Evaluation of Ion Binding to DNA Duplexes Using a Size-Modified Poisson-Boltzmann Theory. *Biophysical Journal*, 2007. 93(9): p. 3202-9.
- [Fogolari2002] Fogolari F, Brigo A, and Molinari H, The Poisson-Boltzmann equation for biomolecular electrostatics: a tool for structural biology. *Journal of Molecular Recognition*, 2002. 15(6): p. 377-92.
- [Grochowski2007] Grochowski P, Istrok A, and Trylska J, Continuum molecular electrostatics, salt effects and counterion binding. A review of the Poisson-Boltzmann theory and its modifications. *Biopolymers*, 2007. 89(2): p. 93-113.
- [Lamm2003] Lamm G, The Poisson-Boltzmann Equation, in *Reviews in Computational Chemistry*, Lipkowitz KB, Larter R, and Cundari TR, Editors. 2003, John Wiley and Sons, Inc. p. 147-366.
- [Levy2003] Levy RM, Zhang LY, Gallicchio E, and Felts AK, On the nonpolar hydration free energy of proteins: surface area and continuum solvent models for the solute-solvent interaction energy. *Journal of the American Chemical Society*, 2003. 125(31): p. 9523-30.
- [Netz2000] Netz RR and Orland H, Beyond Poisson-Boltzmann: Fluctuation effects and correlation functions. *European Physical Journal E*, 2000. 1(2-3): p. 203-14.
- [Ren2012] Ren P, Chun J, Thomas DG, Schnieders M, Marucho M, Zhang J, Baker NA, Biomolecular electrostatics and solvation: a computational perspective. *Quarterly Reviews of Biophysics*, 2012. 45(4): p. 427-491.
- [Roux1999] Roux B and Simonson T, Implicit solvent models. *Biophysical Chemistry*, 1999. 78(1-2): p. 1-20.
- [Vitalis2004] Vitalis A, Baker NA, McCammon JA, ISIM: A program for grand canonical Monte Carlo simulations of the ionic environment of biomolecules, *Molecular Simulation*, 2004, 30(1), 45-61.
- [Wagoner2006] Wagoner JA and Baker NA, Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms. *Proceedings of the National Academy of Sciences of the United States of America*, 2006. 103(22): p. 8331-6.

[Warshel2006] Warshel A, Sharma PK, Kato M, and Parson WW, Modeling electrostatic effects in proteins. *Biochimica et Biophysica Acta (BBA) - Proteins & Proteomics*, 2006. 1764(11): p. 1647-76.

a

apbs, [88](#)

A

apbs (*module*), 88