APBS user manual

APBS user guide

APBS 1.0.0 User Guide

Adaptive Poisson-Boltzmann Solver
Nathan A. Baker [1]
Washington University in St. Louis [2]
Department of Biochemistry and Molecular Biophysics [3]
Center for Computational Biology [4]

Table of contents

External links

[1] http://agave.wustl.edu/
[2] http://www.wustl.edu/
[3] http://www.biochem.wustl.edu/
[4] http://www.ccb.wustl.edu/

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=APBS_user_guide&oldid=395

Principal Authors: Sobolevnrm

Introduction

APBS is a software package for the numerical solution of the Poisson-Boltzmann equation (PBE), one of the most popular continuum models for describing electrostatic interactions between molecular solutes in salty, aqueous media. Continuum electrostatics plays an important role in several areas of biomolecular simulation, including:

- simulation of diffusional processes to determine ligand-protein and protein-protein binding kinetics,
- implicit solvent molecular dynamics of biomolecules,
- solvation and binding energy calculations to determine ligand-protein and protein-protein equilibrium binding constants and aid in rational drug design,
- and biomolecular titration studies.

APBS was designed to efficiently evaluate electrostatic properties for such simulations for a wide range of length scales to enable the investigation of molecules with tens to millions of atoms.

APBS uses PMG to solve the Poisson-Boltzmann equation numerically. PMG is developed and maintained by the Holst Research Group at UC San Diego, and is designed to solve the nonlinear Poisson-Boltzmann equation and similar problems with linear space and time complexity through the use of box methods, inexact Newton methods, and algebraic multilevel methods. More information about PMG may be found at http://www.fetk.org.

Additionally, APBS uses the "Aqua" library, a version of Holst group PMG that has been specially optimized by Patrice Koehl for improved memory usage and speed when solving the Poisson-Boltzmann equation.

APBS also uses FEtk to solve the Poisson-Boltzmann equation numerically. FEtk is developed and maintained by the Holst Research Group at UC San Diego, and is designed to solve general coupled systems of nonlinear partial differential equations accurately and efficiently using adaptive multilevel finite element methods, inexact Newton methods, algebraic multilevel methods. More information about FEtk may be found at http://www.fetk.org.

Financial support

The development of APBS has been supported financially by:

- National Institutes of Health (grant GM069702-01)
- National Partnership for Advanced Computational Infrastructure
- National Biomedical Computation Resource

Citing APBS

Please acknowledge your use of APBS by citing:

- Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proceedings of the National Academy of Sciences USA*, **98**, 10037-10041, 2001. doi:10.1073/pnas.181342398 [1]

Please also acknowledge your use of PMG and FEtk as the primary numerical libraries in APBS by citing:

- Holst MJ, Saied F. Multigrid solution of the Poisson-Boltzmann equation. *Journal of Computational Chemistry*, **14**, 105-113, 1993. doi:10.1002/jcc.540140114 [2]
- Holst MJ, Saied F. Numerical solution of the nonlinear Poisson-Boltzmann equation: Developing more robust and efficient methods. *J Comput Chem*, **16**, 337-364, 1995. doi:10.1002/jcc.540160308 [3]
- Holst, M. Adaptive numerical treatment of elliptic systems on manifolds. *Advances in Computational Mathematics*, **15**, 139-191, 2001. doi:10.1023/A:1014246117321 [4]
- Bank RE, Holst M. A New Paradigm for Parallel Adaptive Meshing Algorithms. *SIAM Review*, **45**, 291-323, 2003. doi:10.1137/S1064827599353701 [5]

Contributing authors

APBS was initially written by Nathan Baker [6] during his graduate work with J. Andrew McCammon [7] and Michael Holst [8] and extensively developed over the subsequent years. APBS uses several libraries written by Mike Holst and members of the Holst group, including: PMG (multigrid solver for Cartesian mesh discretization), FEtk (provides finite element framework, error estimators, and solvers), and MALOC (hardware abstraction library for code portability). Additionally, a number of people have made important contributions to enhance APBS functionality and usability. The full author list (in alphabetical order) is:

Nathan A. Baker

Primary author

Steve Bond

Contributor: FEtk library compatibility and Vopot functions

Larry Canino

Contributor: solvent accessibility function modification

Todd Dolinsky

Developer and contributor: Python wrappers, PDB2PQR, output logging, examples, and other tools

Adrian Elcock

Contributor: parameter files

David Gohara

Developer

Yong Huang

Developer

Michael J. Holst

Advisor during initial development, author of several routines scattered about APBS, author of PMG (the multigrid library used by this code), author of MALOC (hardware abstraction library used by this code), author of FEtk (finite element library incorporated in developmental versions of this code)

Adrian Kaats

Contributor: multivalue.c

Patrice Koehl

Contributor: Aqua, a streamlined version of the Holst group PMG multigrid library

Robert Konecny

Contributor: parameters, CHARMM FORTRAN interface

Contributor: OpenDX to MOLMOL conversion

Chiansan Ma

Contributor: format conversion scripts

J. Andrew McCammon

Advisor and financial and equipment support during initial development, functionality suggestions

Jens Nielsen

Contributor: PDB2PQR

Jay Ponder

Contributor: TINKER Interface

Michael Schnieders

Contributor: Polarizable atomic multipole routines, TINKER interface, higher-order spline routines

David Sept

Contributor: scripts and tools, VMD compatibility, general suggestions about functionality

Tongye Shen

Contributor: finite element mesh generation

Samir Unni

Contributor: APBS Opal web services client code

Justin Xiang

Contributor: generalized Born Python utilities

External links

[1] http://dx.doi.org/10.1073/pnas.181342398
[2] http://dx.doi.org/10.1002/jcc.540140114
[3] http://dx.doi.org/10.1002/jcc.540160308
[4] http://dx.doi.org/10.1023/A:1014246117321
[5] http://dx.doi.org/10.1137/S1064827599353701
[6] http://agave.wustl.edu
[7] http://mccammon.ucsd.edu
[8] http://ccom.ucsd.edu/~mholst/

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=Introduction&oldid=437

Principal Authors: Sobolevnrm

Installation and availability

Availability

The latest version of APBS can always be found at http://apbs.sourceforge.net. APBS is available in both source code form and binaries for a variety of architectures.

Binary installation

We currently offer binaries for the RedHat Linux platform on a variety of architectures as well as command-line binaries for WinXP and Mac OS X. Binaries can be downloaded from the APBS download page. For all other systems, please install from source on your particular platform and feel free to contact the APBS users mailing list for more help and/or to request a binary for that system.

> **Warning** If you are using APBS on a Windows system, you may not want to install it in a directory with spaces in the path name (e.g., C:\Program Files\) as this can cause problems with some visualization programs.

APBS binaries are provided in compressed tar format (*.tgz). On most systems, the binaries can be unarchived by simply double-clicking or opening the archive. This can also be accomplished on the command line by

```
    gzip -dc apbs-1.0.0-XYZ.tgz | tar xvf -
```

where *XYZ* is the particular architecture of the binary you downloaded. Note that this will expand into a directory called apbs-1.0.0-*XYZ*. The contents of this directory can be placed anywhere on your system that you prefer (and have access to). The specific sub-directory of this archive is described in the APBS directory structure section, which contains information about the location of documentation, examples, libraries, header files, and binaries. The APBS binaries do not contain dependencies on special data files, etc. and can be moved out of this directory structure without causing any problems for APBS execution.

Source installation

If you were unable to find the binary package for your system, or would like to compile APBS yourself, you'll need to read the instructions in this section.

Prerequisites

In order to install APBS from the source code, you will need:

- C and FORTRAN compilers
- The APBS source code (see above)

It may also be useful to have:

- A version of MPI (try MPICH) for parallel jobs. However, MPI isn't strictly necessary if the async option is used.
- A vendor-supplied version of BLAS for optimal performance. APBS will attempt to locate common BLAS libraries during the configuration process; if it cannot find one, it will use a generic set of BLAS routines.

Tested systems

Source-code-based installation has been tested on the following systems

- AMD 64 (Opteron) running Fedora Core 6 Linux
- IBM Power4 running RedHat Enterprise Linux 3
- SGI Itanium2 (Altix) running IRIX/Linux
- Apple PowerPC running Mac OS X
- Apple x86_64 running Mac OS X

However, the installation procedure is rather generic and generally works on most UNIX-based systems. System-specific installation notes and caveats are provided in the System-specific notes section.

Preparation for installation

In what follows, I'll be assuming you're using bash [1], a shell available on most platforms (UNIX and non-UNIX). The APBS installation process does not require bash, but it's easiest for outlining the installation procedure.

Compiler variables

First, please look at the System-specific notes section of this document for appropriate compiler flags, etc. to be set via pre-configuration environmental variables. This section outlines generic installation procedures with default compilers and options.

Installation directories

There are a few directories you'll need to identify prior to installation. The first, which we'll call *APBS_SRC*, will contain the APBS and MALOC source code. This directory can be deleted after installation, if you wish. The second directory will be the permanent location for APBS and MALOC; we'll call this *APBS_PREFIX*. If you have root permission, you could pick a global directory such as /usr/local for this; otherwise, pick a directory for which you have write permission. The following commands set up the directories and environmental variables which point to them:

```
$ export APBS_SRC=/home/soft/src
$ export APBS_PREFIX=/home/soft
$ mkdir -p ${APBS_SRC} ${APBS_PREFIX}
```

Unpacking the source code

You're now ready to unpack the source code:

```
$ cd ${APBS_SRC}
$ gzip -dc apbs-1.0.0.tar.gz | tar xvf -
```

Configuring, compiling, installing, and testing

Before compiling/installing APBS, you need to configure the package with the included autoconf configure script. You can examine the various configure options by running `configure` with the --help option. For many platforms, no options need to be specified. Therefore, most users who want single-CPU (not parallel) binaries can configure as follows:

```
$ cd ${APBS_SRC}/apbs
$ ./configure --prefix=${APBS_PREFIX}
```

Other configuration options, including the compilation of parallel binaries and the use of machine-specific compilers and Python usage, are discussed in the System-specific notes section.

Assuming all has gone well with the configuration (you'll generally get an error message if configuration fails), you're ready to compile

```
$ make all
```

and install APBS:

```
$ make install
```

For those with obsessive-compulsive tendencies, you can also test the APBS binary you just compiled against a suite of benchmark jobs by running

```
$ make test
```

You should be able to watch the various tests pass on your platform. If they don't pass, please let us know by posting to the APBS-USERS mailing list [2]. These tests can also be run before make install is executed.

You can test the [Opal version of APBS] separately by typing

```
$ make test-opal
```

This follows a similar set of tests as the normal make test described above. However, because of runtime dependencies, the `make test-opal` command must be run after APBS has been installed (i.e. after `make install` has been issued).

APBS directory structure

The APBS installation process (whether compiled from source or installed as pre-compiled binaries) will create several directories under `${APBS_PREFIX}`:

bin

where the main apbs binary resides

share

contains the documentation (user guide, tutorial, programmer's guide) as well as a number of examples and test cases for APBS

include

contains header files for using APBS libraries with other applications

lib

contains library files for using APBS libraries with other applications

tools

which contains a number of "helper" applications for use with APBS.

At this point you are ready to use APBS; either by calling the binary directly or adding the above directory to your path. As mentioned above, there are also several tools provided with APBS that remain in the APBS directory; these are described in later portions of this manual. You may wish to copy these to a global location (or the same place as your APBS binary) at this time.

System-specific notes

Please note: If you have tips or tricks on improving APBS performance and/or installation on your machine, let us know [3]!

This section provides tips on compiling APBS on specific platforms and outlines some of the basic options available for parallel execution, Python linkage, etc. Note that many aspects of this section have changed from previous releases. In particular, APBS now tries to detect the optimal compilers and BLAS libraries on most systems without user intervention.

As described above, the default configure-make-install procedure is

```
$ ./configure --prefix=${APBS_PREFIX}
$ make
$ make install
```

The configure script includes a number of generic options to manually set default compilers, link behaviors, preprocessors, etc. These can be reviewed by running

```
$ ./configure --help
```

Using an external BLAS library

If an optimized (vendor) BLAS library is available, it should be used with APBS since it will generally provide better performance. This can be enabled by the `--with-blas=...` option. The configure-make-install procedure for a custom BLAS library is:

```
$ ./configure --with-blas='-L/path/to/blas -lblas'
$ make
 make install
```

where `/path/to/blas` is the the path to a directory which contains a library named `libblas.a`.

Python support

Core APBS functionality can also be invoked via Python for inclusion in other applications and more complicated scripting environments. Python library support is enabled at configure-time. Currently, Python libraries compile on most Linux and Mac systems. Other systems are untested. The configure-make-install procedure for enabling Python extensions is:

```
$ ./configure --enable-python --prefix=${APBS_PREFIX}
$ make
$ make install
```

Parallel (MPI) support

APBS uses MPI for parallel execution. In general, enabling MPI support requires informing the APBS configure script about:

- MPI compiler options. For most MPI implementations, you simply need to set the `CC` and `F77` variables to point to the MPI-savvy C and FORTRAN compilers. However, it is occasionally necessary to manually specify compiler options by setting `CFLAGS`, `CPPFLAGS`, `FFLAGS`, and `LDFLAGS` environmental variables before configuration.
- MPI library/header file locations. As outlined below, paths to the library and header files for LAM [4], MPICH [5], or Open MPI [6] implementations of MPI can be specified with the `--with-lam`, `--with-mpich`, or `--with-mpich2` configure options. Other MPI implementations will require the `CFLAGS`, `CPPFLAGS`, `FFLAGS`, and `LDFLAGS` environmental variables to be set correctly before configuration to locate the required headers and libraries.

LAM MPI

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling LAM [7] MPI. For example, if your C compiler is set in the environmental variable `CC` and your FORTRAN compiler is set in the environmental variable `F77`, then you should configure LAM with the command

```
$ ./configure --prefix=${MPI_PREFIX} --with-fc=${F77}
```

Let `MPI_PREFIX` be an environmental variable pointing to the directory where LAM MPI is installed. In other words, `${MPI_PREFIX}/lib` should contain the LAM MPI libraries and `${MPI_PREFIX}/include` should contain the LAM MPI header files. The

configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-lam=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with LAM MPI 7.2.1.

MPICH1

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling MPICH1 [8]. For example, if your C compiler is set in the environmental variable `CC` and your FORTRAN compiler is set in the environmental variable `F77`, then you should configure MPICH1 with the command

```
$ ./configure --prefix=${MPI_PREFIX} ... --enable-f77
```

where `...` denotes other machine-specific options required by the MPICH compiler (e.g., `--with-device=ch_p4` or `--with-arch=LINUX`).

Let `MPI_PREFIX` be an environmental variable pointing to the directory where MPICH1 is installed. In other words, `${MPI_PREFIX}/lib` should contain the MPICH1 libraries and `${MPI_PREFIX}/include` should contain the MPICH1 header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-mpich=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with MPICH1 1.0.6p1.

MPICH2

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling MPICH2 [9].

Let `MPI_PREFIX` be an environmental variable pointing to the directory where MPICH2 is installed. In other words, `${MPI_PREFIX}/lib` should contain the MPICH2 libraries and `${MPI_PREFIX}/include` should contain the MPICH2 header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-mpich2=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with MPICH2 1.2.7p1.

OpenMPI

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling OpenMPI [10].

Let `MPI_PREFIX` be an environmental variable pointing to the directory where OpenMPI is installed. In other words, `${MPI_PREFIX}/lib` should contain the OpenMPI libraries and `${MPI_PREFIX}/include` should contain the OpenMPI header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-openmpi=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with OpenMPI 1.2.8.

FEtk support

In order to enable support for the finite element features in APBS, you must compile it against the FEtk [11] libraries. Please use the same compilers for both APBS and FEtk. Let `FETK_PREFIX` be an environmental variable pointing to the directory where FEtk was installed. In other words, `${FETK_PREFIX}/lib` should contain the FEtk machine-specific library directories and `${FETK_PREFIX}/include` should contain the FEtk header files. Note that FEtk installs library files in directories designated with the GNU "machine triplet". Therefore, you'll first need to identify the appropriate library directory for your system in `${FETK_PREFIX}/lib`. For the purposes of this example, let's suppose this directory is `${FETK_PREFIX}/lib/x86_64-unknown-linux-gnu`. The configure-make-install procedure for APBS is then

```
$ ./configure --with-fetk-include=${FETK_PREFIX}/include
--with-fetk-library=${FETK_PREFIX}/lib/x86_64-unknown-linux-gnu
--prefix=${APBS_PREFIX}
    $ make
    $ make install
```

Windows

We provide native APBS command line binaries for Windows; these binaries are probably the best options for most Windows users.

However, if you would still like to compile your own binaries you will need to use either the Cygwin [12] or MinGW [13] environments. Binaries compiled under Cygwin tend to require Cygwin DLLs and thus can only be run on systems with Cygwin. Performance for the Windows binaries and all compiled systems will be fairly mediocre as they depend on the GNU compilers.

If you do choose to use Cygwin and compile your own code, compilation should be rather straightforward. Good luck.

Macintosh

We provide Mac install packages for Mac OS X versions greater than 10.4. Users on Mac OS 10.3 or older will have to compile binaries for themselves - and may want to examine the apbs-users mailing list [14] which has a number of threads which discuss installation on Mac OS platforms.

A few notes about compiling on Macintosh:

- It has become apparent from the mailing lists that some "packages" of the GNU development software available for MacOS contain different major versions of the C and FORTRAN compilers. This is very bad; APBS will not compile with different versions of the C and FORTRAN compilers. If you use GCC 4.0, for instance, gfortran 4.0 will work while g77 3.3 will not. If you see link errors involving "restFP" or "saveFP" this is most likely the cause.
- In gcc 4.0 (included in Xcode [15] 2.0 and higher) the -fast option turns on the -fast-math flag. This flag optimizes by using rounding, and thus can lead to inaccurate results and should be avoided.
- Currently the autoconf script does not support using the native vecLib framework as an architecture-tuned BLAS replacement. In testing there were only slight timing improvements over using the MALOC-supplied BLAS.
- For older PowerPC-based systems: we have had success using IBM's XLF for Mac in conjunction with GCC 4.0, although the corresponding XLC compilers do not seem to work under Tiger.

Finally, Dave Gohara has prepared a nice tutorial [16] on building APBS from within Xcode to take advantage of its development and debugging features.

Go back to the previous page, forward to the next page, or up to the main page.

External links

[1] http://www.gnu.org/software/bash/bash.html
[2] https://lists.sourceforge.net/lists/listinfo/apbs-users
[3] http://lists.sourceforge.net/lists/listinfo/apbs-users
[4] http://www.lam-mpi.org/
[5] http://www-unix.mcs.anl.gov/mpi/
[6] http://www.open-mpi.org/
[7] http://www.lam-mpi.org/
[8] http://www-unix.mcs.anl.gov/mpi/mpich1/
[9] http://www-unix.mcs.anl.gov/mpi/mpich2/
[10] http://www.open-mpi.org/
[11] http://www.fetk.org/
[12] http://www.cygwin.com/
[13] http://www.mingw.org/
[14] http://sourceforge.net/mailarchive/forum.php?forum=apbs-users
[15] http://www.apple.com/macosx/features/xcode/
[16] http://www.macresearch.org/
tutorial_building_configure_make_projects_in_xcode

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=Installation_and_availability&oldid=109

Principal Authors: Sobolevnrm

An overview of the APBS package

This section gives an overview of the binaries, tools, etc. distributed as part of the APBS software package. It is organized by directory; later chapters provide a more in-depth description on tools specific to particular applications.

The main APBS executable

As mentioned in the Installation and availability section, the main APBS binary is installed in `${APBS_PREFIX}/bin` where `${APBS_PREFIX}` is the top-level directory you chose for the installation. Of course, you can move the binary to any directory you choose. APBS is invoked with a very simple syntax discussed in detail in the Running APBS section.

Other tools distributed with APBS

APBS contains a number of tools to facilitate the preparation of APBS runs and analysis of the results.

> **Note** In addition to the tools provided with APBS, there are a number of other programs that are not distributed with APBS but which interoperate with our code. Please see the External programs section of this manual for more information.

Parameterization

Unfortunately, the majority of problems encountered during electrostatics calculations arise in process of taking a structure from the Protein Data Bank and transforming into a file that can be used by the APBS software. The PDB2PQR service [1], described in additional detail in the External programs section, was developed to address these issues. Additionally, APBS provides the ability to read plain PDB-format files and assign charges and radii from user-supplied parameter files. These features are described in the READ parm command description.

APBS provides a few other miscellaneous tools for converting and parameterizing structures:

tools/conversion/qcd2pqr.awk

Convert a QCD file (UHBD format for a molecule) to PQR format.

tools/conversion/amber2charmm.sh

A script which converts a PDB file with AMBER atom names to a PDB file with CHARMM atom names.

tools/conversion/WHATIF2AMBER.sed

A sed script for converting a PDB file with WHATIF atom names to a PDB file with AMBER atom names. Written by Chiansan Ma.

Problem setup

In addition to parametrization of the molecule, there are several common operations which are performed to setup the calculation. This section reviews some of the tools available for these operations. Please note that PDB2PQR service [2], described in additional detail in the External programs section, also prepares APBS input files.

The following scripts help generate or transform APBS input files:

tools/manip/psize.py

Get the dimensions and center of a molecule in PQR format. Very useful for setting up input files (i.e., grid dimensions, lengths, spacings, etc.) for APBS calculations. Written by Todd Dolinsky and Nathan Baker.

apbs/tools/manip/inputgen.py

Generate an APBS input file from PQR format data using "suggested" parameters. Also can decouple a parallel calculation into a series of sequential (asynchronous) calculations to be performed on a single processor. Written by Todd Dolinsky and Nathan Baker.

tools/mesh/mgmesh

List acceptable grid dimensions/multigrid levels combinations for mg-manual calculations. Written by Nathan Baker.

Output data processing

The following tools perform typical analyses of the output data, usually in OpenDX format. These scripts are not meant to be comprehensive; instead, they provide templates for users to generate their own tools.

Conversion

The following utilities convert APBS grid output (e.g., potentials, accessibility functions, etc.) into a variety of other formats:

tools/mesh/uhbd_asc2bin

Converts UHBD format grid files from ASCII to binary. Contributed by Dave Sept.

tools/mesh/dx2mol

Converts OpenDX format data to MOLMOL [3] format. Contributed by Jung-Hsin Lin with bug fixes by Fred Damberger.

tools/mesh/dx2uhbd

Converts OpenDX format data to UHBD format. Contributed by Robert Konecny.

Manipulation

The following utilities process APBS grid output in a variety of ways:

tools/mesh/mergedx and tools/mesh/mergedx2

Merge OpenDX format data from several domains (e.g., from a mg-para calculation into a single file. `mergedx2` is easier to use and will eventually replace `mergedx`. Contributed by Steve Bond and Dave Gohara.

tools/mesh/smooth

Apply a very inefficient Gaussian filter to OpenDX format data from APBS. Written by Nathan Baker.

Data visualization

This section describes the data visualization tools provided with APBS. A more complete discussion of the various ways to visualize APBS output is presented in the Visualization section of this manual. Old tools for VMD [4] have been removed since they are now completely replaced by OpenDX and APBS functionality available within VMD.

tools/visualization/opendx

This directory contains the OpenDX [5] program files (`*.net`) required to visualize APBS data with OpenDX. In particular, one can visualize single-file potential isocontours (`pot.*`), single-file potential data mapped onto molecular surfaces (`potacc.*`), or multiple-file potential data (`multipot.*`).

Solvent accessibility

The main APBS executable calculates molecular volumes, surface areas, and other surface-based properties from PQR-format structural data. Such calculations are often used to determine apolar solvation contributions to binding events, etc. See the new APOLAR input file section for more documentation on this APBS feature. With the introduction of this new feature, we have deprecated and removed the stand-alone tools which used to perform these functions.

Coulomb's Law and Generalized Born calculations

These utilities are provided for occasional use and are definitely not optimized for speed.

Coulomb's Law calculations

The program `tools/manip/coulomb` calculates vacuum Coulomb law energies from a PQR file. It has a number of options which can be viewed by running the coulomb program with no arguments.

Generalized Born calculations

The program `tools/manip/born` is a crude, non-optimal, buggy program for calculating Generalized Born electrostatic energies. This is only intended for hacking and general comparison with Poisson-Boltzmann results.

The Python-based program `tools/python/runGB.py` is a test program designed to calculate generalized Born radii from APBS Poisson-Boltzmann calculations following the general methods of

Onufriev A, Case DA, Bashford D. Effective Born radii in the generalized Born approximation: The importance of being perfect. *J Comput Chem*, **23** (14), 1297-304, 2002. `http://dx.doi.org/10.1002/jcc.10126.`

More information on this program can be obtained by running it from the command line with the --help option.

The Python-based program `tools/python/readGB.py` is a test program designed to use radii calculated from `runGB.py` (see above) and print out solvation energies. More information on this program can be obtained by running it from the command line with the

`--help` option.

Both of these Python-based programs were written by Justin Xiang.

Eigenvalue analysis

If APBS is linked with ARPACK (see `./configure --help`), the `tools/arpack/driver` routine will perform eigenvalue analyses of Poisson or Poisson-Boltzmann matrices produced by APBS.

Python development tools

There are a number of example Python [6] tools and wrappers provided in the `tools/python directory`. These tools all make use of the APBS SWIG [7] wrappers developed by Todd Dolinsky, Nathan Baker, Alex Gillet, and Michel Sanner. The SWIG wrappers are compiled by default during normal installation. The Python scripts which link to the wrappers (and thereby illustrate their use) include:

tools/python/main.py

Drop-in replacement for main APBS executable. Only permits sequential runs.

tools/python/noinput.py

Similar to `main.py`, but adds the ability to read input files and PQR files as Python strings and return energies and forces as Python lists. This makes it a very useful tool for working with APBS via Python without dealing with a great deal of file I/O.

tools/python/vgrid/

Python wrappers for Vgrid class to allow OpenDX format file I/O in Python scripts

Examples and tutorial

The APBS sub-directory `examples` contains several test systems which show how to use APBS for binding energy, solvation energy, and force calculations. The file `examples/README.html` contains descriptions of the test cases and links to anticipated results. Examples can be run and compared to expected results by running make test in each example directory.

Additional examples are provided as part of the APBS tutorial (`doc/html/tutorial/`), described in more detail in the Documentation section.

Documentation

The APBS sub-directory doc contains guides for using APBS and developing code based on APBS libraries. The sub-directories include:

doc/html/user-guide/index.html

HTML-format User Guide

doc/html/programmer/index.html

HTML-format Programmer Guide

doc/html/tutorial/index.html

HTML-format APBS tutorial

Source code

The APBS sub-directory `src` contains the source code for the APBS libraries and main executable. These files are described in more detailed in the Programming section.

Go back to the previous page, forward to the next page, or up to the main page.

External links

[1] `http://pdb2pqr.sf.net`
[2] `http://pdb2pqr.sf.net`
[3] `http://www.mol.biol.ethz.ch/wuthrich/software/molmol/`
[4] `http://www.ks.uiuc.edu/Research/vmd/`
[5] `http://www.opendx.org`
[6] `http://www.python.org`
[7] `http://www.swig.org`

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=An_overview_of_the_APBS_package&oldid=436

Principal Authors: Sobolevnrm

Running APBS

APBS can either be used by itself or with other programs.

Invocation

As mentioned in the Installation and availability section, the main APBS binary is installed in `${APBS_PREFIX}/bin` where `${APBS_PREFIX}` is the top-level directory you chose for the installation. Of course, you can move the binary to any directory you choose. APBS is invoked with a very simple syntax:

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

Command line options include:

--outputfile=name

Sets the output logging path (as described in the output logging section of the manual) to `name`, or `name_N` for parallel runs, where `N` is the processor ID. If `--outputformat` is not specified, a flat-file format will be used as the default.

--outputformat=type

Sets the output logging format. Accepted values are:

; flat

Flat-file format (default).

; xml

XML format

--help

Displays command line usage

--version

Displays the current APBS version

`input-file` is an input file with a specific syntax described in the section Input files. Besides the output files specified from within `input-file` and the optional logs as specified by use of the `--output-file` command line option, APBS writes data to three additional places:

- Standard output. This will appear on your screen (if you don't redirect it somewhere) and will contain all the basic information about the electrostatics calculation.
- Standard error. This will also appear on your screen (if you don't redirect it somewhere) and will contain warnings and error messages.
- The file `io.mc` (or `io.mc_N` for parallel runs, where `N` is the processor ID. This gives you detailed information about the progress of the run with a particular focus on the numerical solver.

Input files

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. Specific examples of APBS input are described in the Examples section.

> **Please note** that there are several tools which help prepare APBS input files based on molecular structures, memory constraints, etc. These tools are described in more detail in the Problem setup section.

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

- READ section for specifying input.
- ELEC section for specifying polar solvation (electrostatics) calculation parameters.
- APOLAR section for specifying apolar solvation calculation parameters.
- PRINT section for specifying summary output.

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 interdependent. 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.

> **NOTE:** There are a number of places in the APBS input files where pathnames can be specified. If the pathname contains 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"`.

Each section of the APBS input file has its own syntax, described in more detail in the following pages:

- READ syntax
- ELEC syntax
- APOLAR syntax
- PRINT syntax

Using APBS with other programs

APBS was designed to facilitate use with other programs. This section outlines some of the programs with which APBS is known to work. However, it is likely that applications which use APBS have been inadvertently omitted from this list. If you know of software that uses APBS and is not listed here, please contact Nathan Baker [1].

Web interfaces

Gemstone

The Gemstone extension (`http://gemstone.mozdev.org/`) for the Firefox web browser **used to** provide a very easy-to-use interface to older versions of APBS (0.4.0) with all of the functionality of the command-line interface. However, this extension was created by external developers and is no longer actively maintained. Please let Nathan Baker [2] know if this extension was important to your research.

NBCR web services

The NBCR Web Services portal (`http://ws.nbcr.net:8080/opal/GetServicesList.do`) provides an off-site alternative for running computationally-expensive APBS calculations that won't fit on local resources. This portal is currently running APBS 1.0.0.

The NBCR Opal Web Services client for APBS (`http://nbcr.net/services/#Software`) provides a more basic interface which allows users to execute APBS jobs remotely via Python or build such functionality into their own software.

However, in recent developmental versions of APBS, we have integrated Opal web services support into the existing command line client. If you are interested in testing this experimental functionality, please let Nathan Baker [3] know.

Graphical user interfaces

PyMOL

PyMOL (`http://pymol.sourceforge.net/`) is a molecular visualization and animation package which provides an interface to APBS. The APBS plugin to PyMOL (developed by Michael George Lerner) permits isocontour and surface map visualization of APBS results. More information about using PyMOL with APBS is provided in the APBS tutorial.

VMD

VMD (`http://www.ks.uiuc.edu/Research/vmd/`) is a molecular visualization and animation package which provides an interface to APBS. It permits visualization of APBS results as isocontours, electric field lines, or on biomolecular surfaces. VMD also a graphical plugin to setup APBS calculations and execute them either locally or remotely via BioCoRE. More information is available at `http://www.ks.uiuc.edu/Research/vmd/plugins/apbsrun/`and in the APBS tutorial.

PMV

PMV (`http://www.scripps.edu/~sanner/python`) is a Python-based molecular visualization package which provides an interface to APBS. It not only permits visualization of APBS results but it also integrates setup and executation of APBS calculations. The PMV/APBS interface (`http://mccammon.ucsd.edu/pmv_apbs/`) is under active development and future versions will offer even more setup, visualization, and analysis functionality.

The APBS interface is distributed with recent beta versions of PMV, avalaible from `http://www.scripps.edu/~sanner/python`. Additional documention for using APBS with PMV is provided at `http://mccammon.ucsd.edu/~jswanson/apbsDoc/command_doc2.html`.

Simulation software

Robert Konecny (McCammon Group [4]) has developed iAPBS (`http://mccammon.ucsd.edu/iapbs/`), an interface between APBS and the simulation packages AMBER, CHARMM, and NAMD. More information is available from the iAPBS homepage: `http://mccammon.ucsd.edu/iapbs/`.

Visualization software

Electrostatic potentials are commonly visualized in the context of biomolecular structure to better understand functional aspects of biological systems. This section describes molecular graphics software which can display potentials and other data output from APBS. Note that the graphical user interfaces discussed above can also be used to visualize APBS output.

Dino3D

Dino3D (`http://www.dino3d.org/`is a molecular graphics program which can read UHBD-format electrostatic data. APBS can write multigrid results in UHBD format (see the write ELEC command) and therefore can be used with Dino3D.

MOLMOL

MOLMOL (`http://www.mol.biol.ethz.ch/wuthrich/software/molmol/)` is a molecular graphics package with an emphasis on NMR-generated structural data. A program is provided with APBS (see tools/mesh directory in the APBS distribution and the Data conversion tools in this manual) which converts OpenDX format data to MOLMOL format.

OpenDX

OpenDX (`http://www.opendx.org)` is a general data visualization package which can read APBS output using the scripts provided in tools/visualization/opendx (see the discussion of Data visualization tools in this manual). However, as there is no straightforward way to visualizate the potential in the context of the atomic structure, OpenDX should not a first choice for APBS visualization.

External links

[1] `mailto:baker@biochem.wustl.edu`
[2] `mailto:baker@biochem.wustl.edu`
[3] `mailto:baker@biochem.wustl.edu`
[4] `http://mccammon.ucsd.edu/`

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=Running_APBS&oldid=440

Principal Authors: Sobolevnrm

READ input file section

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

```
READ
    [ keywords... ]
END
```

(where 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 incremented 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.

Keywords

All keywords in this section are optional.

mol

```
mol {format} {path}
```

This command specifies the molecular data to be read into APBS. The required arguments are:

format

The format of the input data. Acceptable values include:

; pqr

Specify that molecular data is in PQR format

; pdb

Specify that molecular data is in pseudo-PDB format.

path

The location of the molecular data file.

parm

```
parm {format} {path}
```

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

format

The format of the parameter file. Acceptable flags include:

; 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.

diel

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

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 write commands in earlier APBS calculations.

NOTE: if you choose this option and have a non-zero ionic strength, you must also include a read kappa statement

Required arguments for this command are:

format

The format of the dielectric map. Acceptable values include:

; dx

OpenDX format

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.

kappa

```
kappa {format} {path}
```

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 $Å^{-2}$. In general, this command will read kappa-maps written by write commands in earlier APBS calculations.

NOTE: if you choose this option, you must also include a `read diel` statement.

Arguments for this command are:

format

The format of the kappa map. Acceptable values include:

; dx

OpenDX format

path

The location of the kappa map file.

charge

```
charge {format} {path}
```

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 $e_c Å^{-2}$, where e_c is the electron charge. In general, this command will read charge-maps written by write commands in earlier APBS calculations. Arguments for this command are:

format

The format of the charge map. Acceptable values include:

; dx

OpenDX format

path

The location of the charge map file.

mesh

```
mesh {format} {path}
```

This command allows APBS to read a finite element mesh to use as a starting point for finite element calculations. The input is simply the mesh geometry; e.g., as produced by a finite element mesh generation program such as LBIE-Mesher [1] or GAMer [2]. Arguments for this command are:

format

The format of the input mesh. Acceptable values include:

; mcsf

MCSF format

path

The location of the meshes file.

Examples

Reading a PQR file

The following is an example of a minimal READ section that only imports PQR format molecular structure files.

```
READ
    mol pqr ligand.pqr
    mol pqr receptor.pqr
    mol pqr complex.pqr
END
```

Reading a PDB file with parameters

```
READ
    mol pdb molecule.pdb
    parm flat param.dat
END
```

Reading external dielectric maps

```
READ
    mol pqr molecule.pqr
    diel dx dielx.dx diely.dx dielz.dx
END
```

External links

[1] http://cvcweb.ices.utexas.edu/ccv/projects/project.php?proID=10
[2] http://www.fetk.org/

Principal Authors: Sobolevnrm

ELEC input file section

The ELEC block of an APBS input file has the following syntax:

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

where the indentation and linefeeds are included for clarity; only whitespace is needed in the input file.

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 matters since certain types of boundary conditions (bcfl) can require information about previous calculations.

ELEC block naming

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 to each ELEC block to simplify the organizational process. This syntax has the form

```
ELEC name {id}
    ...
```

where `ELEC` is the start of the ELEC block and `{id}` is an alphanumeric string denoting the "name" of the calculation block.

Types of ELEC calculations

Each ELEC statement must start with the type of calculation to be performed. The type is specified with the following syntax

```
ELEC [ [#ELEC block naming|name {id}] ]
    {type}
    {keywords...}
```

where `{type}` is one of the following keywords denoting different types of electrostatics calculations

mg-auto

Automatically-configured sequential focusing multigrid Poisson-Boltzmann calculations.

mg-para

Automatically-configured parallel focusing multigrid Poisson-Boltzmann calculations.

mg-manual

Manually-configured multigrid Poisson-Boltzmann calculations.

fe-manual

Manually-configured adaptive finite element Poisson-Boltzmann calculations.

mg-dummy

Calculations of surface and charge distribution properties which do not require solution of the PBE.

and `{keywords...}` are the calculation-specific keywords. As a general rule, all keywords in an APBS input file are required. This rule is broken in a few places with very rare keywords but should be observed for most typical calculations.

Each of these types of calculations, and their accompanying keywords, is described in more detail below.

Automatic sequential focusing multigrid calculation (mg-auto)

This 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 probably use this version of ELEC. The following keywords are present in mg-auto ELEC blocks; all keywords are required unless otherwise noted:

- dime
- cglen
- fglen
- cgcent
- fgcent
- mol
- lpbe or npbe or smpbe
- bcfl
- ion (optional)
- pdie
- sdie
- chgm
- usemap (optional)
- useaqua (optional)
- sdens
- srfm
- srad
- swin
- temp
- calcenergy
- calcforce
- write
- writemat

Automatic parallel focusing multigrid calculation (mg-para)

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

Manual multigrid calculation (mg-manual)

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 bcfl 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.

Manual adaptive finite element calculation (fe-manual)

This is a single-point PBE calculation performed by our adaptive finite element PBE solver. It requires that APBS be linked to the Holst group FEtk finite element library `http://www.fetk.org` 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

until a global error tolerance is reached.

> **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.

Manual non-numerical calculations (mg-dummy)

This 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. The syntax for this command is identical to mg-manual.

ELEC keywords

This is a list of keywords used in the ELEC statements of APBS. Note that not all keywords are used in every ELEC statement; see above.

akeyPRE

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. 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

Specifies how the the finite element mesh should be adaptively subdivided during the solve-estimate-refine iterations. This allows for various a posteriori refinement schemes. 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

async

This optional keyword allows users to perform the different tasks in a parallel run asynchronously. Specifically, a processor masquerades as process rank in a parallel focusing run and provides output (data files and energies/forces) appropriate to that processor's local partition. The user must then assemble the results after all processes complete. First, this option is useful for scheduling on-demand resources: this makes it easy for users to backfill into the available processes in a queue. Second, this option is useful for running on limited resources: this enables users without access to large parallel machines to still perform the same calculations. 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 $p = n_x n_y k + n_x j + i$ 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.

bcfl

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 sphere radii are set to the atomic radii of the biomolecule and the sphere charges are set to the total charge of the protein. 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 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).

calcenergy

This optional keyword controls electrostatic energy output from a Poisson-Boltzmann calculation. **Note** that this option must be used consistently for all calculations that will appear in subsequent PRINT statements. For example, if the statement `print energy 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have calcenergy keywords present with the same values for `flag`. The syntax for this keyword is:

```
      calcenergy { flag }
```

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

no

(Deprecated) don't calculate any energies.

total

Calculate and return total electrostatic energy for the entire molecule.

comps

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

calcforce

This optional keyword controls electrostatic force output from a Poisson-Boltzmann calculation. **Note** that this option must be used consistently for all calculations that will appear in subsequent PRINT statements. For example, if the statement `print force 1 -2 end` appears in the input file, then both calculations 1 and 2 must have calcforce keywords present with the same values for `flag`. The syntax for this keyword 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.

cgcent

Specify the center of the coarse grid (in a focusing calculation) based on a molecule's center or absolute coordinates. 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.

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.

See also: gcent and fgcent

cglen

Specify the coarse mesh domain lengths in a focusing calculation; this may be different in each direction. Its syntax is

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

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

xlen ylen zlen

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

See also: fglen or glen

chgm

Specify the method by which the biomolecular point charges (i.e., Dirac delta functions) are mapped onto the grid. As we are attempting to model delta functions, the support (domain) of these discretized charge distributions is always a function of the grid spacing. The syntax for this command is:

```
    chgm {flag}
```

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

spl0

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.

spl2

Cubic B-spline discretization. The charge is mapped onto the nearest- and next-nearest-neighbor grid points. Resulting potentials are somewhat less sensitive (than spl0) to grid spacing, length, and position.

spl4

Quintic B-spline discretization. Similar to spl2, except the charge/multipole is additionally mapped to include next-next-nearest neighbors (125 grid points receive charge density).

dime

Specifies the number of grid points per processor for grid-based discretization. Its 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.

domainLength

Specify the rectangular finite element mesh domain lengths; this 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

Specify the method used to determine the error tolerance in the solve-estimate-refine iterations of the finite element solver. 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

See also: etol

etol

Specify the tolerance for error-based adaptive refinement during the solve-estimate-refine iterations of the finite element solver. The syntax is

```
etol { tol }
```

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

See also: ekey

fgcent

Specify the center of the fine grid (in a focusing calculation) based on a molecule's center or absolute coordinates. 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. 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 PDB coordinate frame.

See also: cgcent

fglen

Specifies the fine mesh domain lengths in a focusing calculation; 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.

See also: cglen

gcent

Specify the center of the grid based on a molecule's center or absolute coordinates. The syntax is:

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

where the user can specify *either*:

mol {id}

Center the grid on molecule with integer ID `id`; as assigned in the READ section. Molecule IDs are assigned in the order they are read, starting at 1.

or the user can specify

xcent ycent zcent

The floating point coordinates (in Å) at which the grid is centered. Based on the PDB coordinate frame.

glen

Specify the mesh domain lengths; this 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.

See also: grid

grid

Specify the mesh grid spacings; this 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.

ion

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 e_c)

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

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

lpbe

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

```
lpbe
```

See also: npbe, lrpbe, nrpbe, smpbe

lrpbe

Specifies that the linearized regularized Poisson-Boltzmann equation should be solved. The syntax is

```
lrpbe
```

See also: npbe, lpbe, nrpbe, smpbe

maxsolve

Specify the number of times to perform the solve-estimate-refine iteration of the finite element solver. The syntax is

```
    maxsolve { num }
```

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

See also: maxvert, targetRes

maxvert

Specify the maximum number of vertices to allow during solve-estimate-refine cycle of finite element solver. 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.

See also: targetRes, maxsolve.

mol

Specify the molecule for which the PBE is to be solved. IDs are based on the order in which molecules are read by READ mol statements, starting from 1. The syntax is

```
    mol {id}
```

where `id` is the integer ID of the molecule for which the Poisson-Boltzmann equation is to be solved.

nlev

Specify the depth of the multilevel hierarchy used in the 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.

npbe

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

```
    npbe
```

See also: lpbe, lrpbe, nrpbe, smpbe

nrpbe

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. The syntax is:

```
    nrpbe
```

Note that this functionality is only available with FEM-based solvers.

See also: lpbe, npbe, lrpbe, smpbe

ofrac

Specify the amount of overlap to include between the individual processors meshes in a parallel focusing calculation. This should be a value between 0 and 1. 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 ofrac 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).

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

ofrac value	Energy (kJ/mol)	Visual artifact in ±
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.

pdie

Specify the dielectric constant of the biomolecule. 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 syntax is:

```
    pdie {diel}
```

where `diel` is the floating point value of the unitless biomolecular dielectric constant.

See also: sdie

pdime

Specify the processor array to be used in a parallel focusing calculation. The product npx × npy × npz 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. The syntax is:

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

where `npx npy npz` are the integer number of processors to be used in the x-, y- and z-directions of the system.

See also: ofrac

sdie

Specify the dielectric constant of the solvent. Bulk water at biologically-relevant temperatures is usually modeled with a dielectric constant of 78-80. The syntax is

```
    sdie {diel}
```

where `diel` is a floating point number representing the solvent dielectric constant (unitless).

See also: pdie

sdens

Specify the number of grid points per square-angstrom to use in discontinuous surface constructions (e.g., molecular surface and solvent-accessible surfaces). Ignored when srad is 0.0 or srfm is spl2. There is a direct correlation between this value used for the surface sphere density, the accuracy of the surface calculations, and the APBS calculation time. The APBS "suggested" value is 10.0. The syntax of this command is

```
    sdens {density}
```

where `density` is the floating point surface sphere density (in grid points/$Å^2$).

See also: srfm

smpbe

Specifies that the size-modified PBE should be solved as described by Chu V, et al Biophys J, **93**(9):3202-9, 2007 (doi:10.1529/biophysj.106.099168 [1]). The syntax is

```
smpbe vol { spacing } size { num }
```

The parameter `spacing` is a floating point number in Ångstroms used specify the lattice spacing such that each lattice site has a volume equal to `spacing`^{3}. The parameter `num` controls the relative size of the ions (in Ångstroms) such that each lattice site can contain a single ion of volume `spacing`^{3} or `num` ions of volume `spacing`^{3}/`size`.

See also: lpbe, npbe, lrpbe, nrpbe

srad

Specify the radius of the solvent molecules; this parameter is used to define the dielectric function for probe-based dielectric definitions (see srfm). This value is usually set to 1.4 Å for water. This keyword is ignored when any of the spline-based surfaces are used (e.g., `spl2`, see srfm), since they are not probe-based. The syntax for this command is:

```
srad {radius}
```

where `radius` is the floating point solvent radius (in Å).

See also: srfm

swin

Specify the size of the support (i.e., the rate of change) for spline-based surface definitions (see srfm). Usually 0.3 Å. The syntax is:

```
swin {win}
```

where `win` is a floating point number for the spline window width (in Å). Note that, per the analysis of Nina, Im, and Roux (doi:10.1016/S0301-4622(98)00236-1 [2]), the force field parameters (radii) generally need to be adjusted if the spline window is changed.

srfm

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

```
srfm {flag}
```

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

mol

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.

smol

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 (journal web site [3]).

spl2

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 (doi:`http://dx.doi.org/10.1016/S0010-4655(98)00016-2` 10.1016/S0010-4655(98)00016-2). 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 (doi:10.1016/S0301-4622(98)00236-1 [4]). Additionally, these surfaces can generate unphysical results with non-zero ionic strengths; this is an on-going area of development.

spl4

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).

targetRes

Specify the target resolution of the simplices in a finite element mesh; refinement will continue until the longest edge of every simplex is below this value. 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). See also: maxvert, maxsolve, targetNum.

targetNum

Specify the target number of vertices in the initial finite element mesh; intiial refinement will continue until this number is reached or the the longest edge of every simplex is below targetNum. The syntax is

```
    targetNum { num }
```

where `num` is an integer denoting the target number of vertices in initial mesh. See also: targetRes

temp

Specify the temperature for the Poisson-Boltzmann calculation. The syntax is:

```
    temp { T }
```

where `T` is a floating point number indicating the temperature in K.

useaqua

This keyword enables experimental support for Aqua, a verison of the Holst group FEtk [5] PMG multigrid library optimized by Patrice Koehl for improved memory usage and speed when solving the Poisson-Boltzmann equation. This keyword is temporary and will eventually disappear as Aqua becomes the default multigrid solver for APBS. The syntax is:

```
    useaqua
```

usemap

Specify pre-calculated coefficient maps to be used in the Poisson-Boltzmann calculation. These must have been input via an earlier READ statement. The syntax 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 diel); this causes the pdie, sdie, srad, swin, and srfm parameters and the radii of the biomolecular atoms to be ignored when computing dielectric values for the Poisson-Boltzmann equation.

; kappa

Mobile ion-accessibility function map (as read by read kappa); this causes the swin and srfm 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 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.

id

As described in the READ command documentation, 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.

usemesh

Specify the external finite element mesh to be used in the finite element Poisson-Boltzmann calculation. These must have been input via an earlier READ mesh statement. The syntax is

```
usemesh {id}
```

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

write

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 $Å^3$. (multigrid only)

; pot

Write out the electrostatic potential 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 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 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 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 smol). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)

; lap

Write out the Laplacian of the potential in units of k_B T e_c^{-1} $Å^{-2}$. (multigrid only)

; edens

Write out the "energy density" in units of k_B T e_c^{-1} $Å^{-2}$. (multigrid only)

; ndens

Write out the total mobile ion number density for all ion species in units of M. (multigrid only)

; qdens

Write out the total mobile charge density for all ion species in units of e_c M. (multigrid only)

; dielx

Write out the dielectric map shifted by 1/2 grid spacing in the x-direction (see READ diel). The values are unitless. (multigrid only)

; diely

Write out the dielectric map shifted by 1/2 grid spacing in the y-direction (see READ diel). The values are unitless. (multigrid only)

; dielz

Write out the dielectric map shifted by 1/2 grid spacing in the z-direction (see READ diel). The values are unitless. (multigrid only)

; kappa

Write out the ion-accessibility kappa map (see READ kappa). The values are in units of $Å^{-2}$ (multigrid only) ; format : A string that specifies the format for writing out the data. :; dx :: Write out data in OpenDX 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 format. (multigrid only) ; 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; e.g., "/path with spaces/foo.in".

writemat

This controls the output of the mathematical operators in the Poisson-Boltzmann equation as matrices in Harwell-Boeing 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; files are written to `stem.mat`. If the pathname contains spaces, then it must be surrounded by double quotes; e.g., "/path with spaces/foo.in".

External links

[1] http://dx.doi.org/10.1529/biophysj.106.099168
[2] http://dx.doi.org/10.1016/S0301-4622(98)00236-1
[3] http://www3.interscience.wiley.com/journal/48460/abstract
[4] http://dx.doi.org/10.1016/S0301-4622(98)00236-1
[5] http://fetk.org

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=ELEC_input_file_section&oldid=439

Principal Authors: Sobolevnrm

APOLAR input file section

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 statements). The `keywords...` describing the parameters of the apolar calculation are discussed in more detail in the section Keyword descriptions.

Basic apolar solvation calculations

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 [1]).

In particular, nonpolar solvation potentials of mean force (energies) are calculated according to:

$$W^{(\mathrm{np})} = \gamma A(x) + pV(x) + \overline{\rho} \int_{\Omega} u^{(\mathrm{att})}(x,y)\theta(y)dy$$

and mean nonpolar solvation forces are calculated according to:

$$F^{(\mathrm{np})} = -\gamma \frac{\partial A(x)}{\partial x} - p\frac{\partial V(x)}{\partial x} - \overline{\rho} \int_{\Omega} \frac{\partial u^{(\mathrm{att})}(x,y)}{\partial x}\theta(y)dy$$

In these equations, γ (see gamma below) is the repulsive (hard sphere) solvent surface tension, A is the conformation-dependent solute surface area (see srad and srfm below), p (see press below) is the repulsive (hard sphere) solvent pressure, V is the

conformation-dependent solute volume (see srad and srfm below), ρ(see bconc below) 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 (see XML Parameter Format section) as read in through an APBS input file READ statement.

Important: 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").

Note that the above expressions can easily be reduced to simpler apolar solvation formalisms by setting one or more of the coefficients (γ , ρ , or p) to zero through the keywords below.

Keyword descriptions

This is a list of keywords used in the APOLAR statements of APBS.

bconc

This keyword specifies the bulk solvent density $\overline{\rho}$ in Å^{-3}. 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 Å^{-3}.

calcenergy

This optional keyword controls electrostatic 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 electrostatic energy components for each atom.

Note that this option must be used consistently for all calculations that will appear in subsequent PRINT statements. For example, if the statement `print energy 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have `calcenergy` keywords present with the same values for `flag`.

calcforce

This optional keyword controls apolar force output. Its syntax is

```
    calcforce {flag}
```

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

no

(Deprecated) don't calculate any forces.

total

Calculate and return total apolar forces for the entire molecule.

comps

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

Note that this option must be used consistently for all calculations that will appear in subsequent PRINT statements. For example, if the statement `print force 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have `calcforce` keywords present with the same values for `flag`.

dpos

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. 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.

Important: this parameter is very dependent on sdens; e.g., smaller values of dpos require larger values of sdens.

gamma

This keyword specifies the γ surface tension coefficient for apolar solvation models. Its syntax is:

```
    gamma { value }
```

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

grid

This keyword specifies the quadrature grid spacing for volume integral calculations. The syntax is:

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

where `hx hy hz` are the quadrature spacings in the x-, y-, and z-directions in Å.

mol

This term specifies the molecule for which the apolar calculation is to be performed. Its 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, starting from 1.

press

This term specifies the solvent pressure p in kJ mol^{-1} Å^{-3}. This coefficient multiplies the volume term of the apolar model discussed above 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 kJ mol^{-1} Å^{-3}.

sdens

This keyword specifies the number of grid points per Å^2 to use in surface calculations (e.g., molecular surface, solvent accessible surface). The keyword is ignored when srad is 0.0 (e.g., for van der Waals surfaces) or when srfm is `spl2` (e.g., for spline surfaces). The syntax is

```
    sdens {density}
```

where `density` is a floating point number indicating the number of grid points per Å^2.

Users beware: there is a direct correlation between the value used for the sphere density, the accuracy of the results, and the APBS calculation time.

srad

This keyword specifies the radius of the solvent molecules; this parameter is used to define various solvent-related surfaces and volumes (see srfm). This value is usually set to 1.4 Å for water. Its syntax is

```
    srad {radius}
```

where `radius` is the floating point value of the solvent radius (in Å).

swin

This keyword specifies the size of the support (i.e., the rate of change) for spline-based surface definitions (see srfm `spl2`). Usually 0.3 Å. The syntax is

```
    swin {win}
```

where `win` is the floating point value of the spline window (in Å).

srfm

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

```
    srfm {flag}
```

where `flag` is a string that specifies the model used for surface and volume:

sacc

Solvent-accessible (also called "probe-inflated") surface and volume.

Note that this keyword is under construction: we're in the process of adding additional surface definitions.

temp

This keyword specifies the temperature for the calculation. Its syntax is

```
    temp {T}
```

where `T` is the floating point value of the temperature for calculation.

External links

[1] http://dx.doi.org/10.1073/pnas.0600118103

PRINT input file section

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:

energy

Print energies as calculated with an earlier calcenergy ELEC command. **Warning:** this keyword is deprecated and will be removed soon. Please use `elecEnergy` or `apolEnergy` as appropriate to obtain the desired energy output. For now, use of this keyword will return the old results of `elecEnergy`.

force

Print forces as calculated with an earlier calcforce ELEC command. **Warning:** this keyword is deprecated and will be removed soon. Please use `elecForce` or `apolForce` as appropriate to obtain the desired energy output.

elecEnergy

Print electrostatic energies as calculated with an earlier calcenergy ELEC command.

elecForce

Print forces as calculated with an earlier calcforce ELEC command.

apolEnergy

Print energies as calculated with an earlier calcenergy APOLAR command.

apolForce

Print forces as calculated with an earlier calcforce APOLAR 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`. These options have the following form:

id

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

op

Specify the arithmetic operation to be performed on the calculated quantities:

; +

Addition

; -

Subtraction

Given all these options, a typical PRINT declaration might look like:

```
# 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
```

See the examples/ directory provided with the APBS distribution for more examples.

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=PRINT_input_file_section&oldid=373

Principal Authors: Sobolevnrm

Getting help

APBS follows an open source software development model and relies heavily on the user community for feedback to enhance the software, identify bugs, etc. We offer several mechanisms for user support (see [#Trackers|Trackers] and [#Mailing lists|Mailing lists] below) and encourage users to explore these venues *before* than directly contacting the developers. Discussion of problems and solutions in these public forums benefits the APBS user community much more than direct, private communication between users and developers.

Trackers

The SourceForge project page [1] offers many user forums for discussion on APBS topics. Users are encouraged to post bugs and request for support and features via the following mechanisms:

Bugs [2]

A tracker for listing and displaying progress on APBS bugs.

Support Requests [3]

A tracker for requesting support for installation on various architectures and general APBS usage.

Feature Requests [4]

A tracker for feature requests and other desired additions to APBS.

Other discussion related to APBS can be posted to the mailing lists.

Mailing lists

There are two mailing lists [5] for the APBS software:

Announcements [6]

This is a very low-traffic list to inform users of new APBS releases and APBS-related news (updates, bugs, etc.).

User discussion [7]

This is a higher-traffic list and is the primary forum for discussion of APBS functionality, usage, possible enhancements, and bugs.

External links

[1] http://sourceforge.net/projects/apbs/
[2] http://sourceforge.net/tracker2/?atid=771704&group_id=148472&func=browse
[3] http://sourceforge.net/tracker2/?atid=771707&group_id=148472&func=browse
[4] http://sourceforge.net/tracker2/?atid=771705&group_id=148472&func=browse
[5] http://sourceforge.net/mail/?group_id=148472
[6] http://lists.sourceforge.net/lists/listinfo/apbs-announce
[7] http://sourceforge.net/mailarchive/forum.php?forum_name=apbs-users

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=Getting_help&oldid=392

Principal Authors: Sobolevnrm

Customizing and expanding APBS

APBS is an open source project and we encourage user modifications of the code. The APBS programmer's guide is available online [1] and distributed with the APBS software.

External links

[1] http://apbs.sourceforge.net/doc/programmer/html/index.html

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=Customizing_and_expanding_APBS&oldid=394

Principal Authors: Sobolevnrm

File formats

This section introduces some of the input and output file formats which are used by APBS.

PQR biomolecular structure format

The PQR format is the primary input format for biomolecular structure in APBS package.

PQR flat-file format

This format is a modification of the PDB format [1] 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 [2]. 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** 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 coordiantes.

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 utilities provided with apbs (see the Parameterization section) attempt to preserve the PDB format as much as possible.

PQR XML Format

The PQR XML format was designed 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:

```
    # Comments
    <roottag>
        <residue>
            <atom>
```

```
                <x>x

                <y>y

                <z>z

                <charge>charge

                <radius>radius

            ...

        ...
```

The variables in this example are:

Comments

Any number of comment lines, each line starting with the "#" symbol. *Yes, we know this isn't valid XML syntax... this will be fixed in future versions...*'

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

A float giving the x-coordinate of the atom in Å.

y

A float giving the y-coordinate of the atom in Å.

z

A float giving the 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).

Parameter file format

APBS uses parameter files with the READ parm command.

Flat-file format

The parameter file is a series of lines of the format:

```
    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 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":

$$U_{\mathrm{LJ}}(r) = \epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right)$$

XML Parameter Format

This data format has the following form:

```
# Comments
<ffname>
    <residue>
        <name>resname

        <atom>
            <name>atomname

            <charge>atomcharge

            <radius>atomradius

            <epsilon>atomepsilon

        ...

    ...
```

The variables in this example are:

Comments

Any number of comment lines, each line starting with the "#" symbol. *This is not valid XML syntax...*

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":

$$U_{\mathrm{LJ}}(r) = \epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right)$$

Harwell-Boeing column-compressed 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 by decomposed with SuperLU and ARPACK but this also serves as a useful general mechanism for sparse matrix input and output.

Important: Details of this format are complicated; this section is under construction.

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 [3]. The OpenDX data format is very flexible; the following sections describe the application of this format for APBS multigrid and finite element datasets.

Multigrid

This data format has the following form:

```
# Comments
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 times n data follows
u(0,0,0) u(0,0,1) u(0,0,2)
...
u(0,0,nz-3) u(0,0,nz-2) u(0,0,nz-1)
u(0,1,0) u(0,1,1) u(0,1,2)
...
u(0,1,nz-3) u(0,1,nz-2) u(0,1,nz-1)
...
u(0,ny-1,nz-3) u(0,ny-1,nz-2) u(0,ny-1,nz-1)
u(1,0,0) u(1,0,1) u(1,0,2)
...
attribute "dep" string "positions"
object "regular positions regular connections" class field
```

```
component "positions" value 1
component "connections" value 2
component "data" value 3
```

The variables in this format have been shown in bold and include

Comments

Any number of comment lines, each line starting with the "#" symbol

nx ny nz

The number of grid points in the x-, y-, and z-directions.

xmin ymin zmin

The coordinates of the grid lower corner.

hx hy hz

The grid spacings in the x-, y-, and z-directions.

n

The total number of grid points; 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.

Finite element

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 shown in bold and defined as:

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

External links

[1] http://www.rcsb.org/pdb/info.html#File_Formats_and_Standards
[2] http://www.rcsb.org/pdb/info.html#File_Formats_and_Standards
[3] http://www.opendx.org

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=File_formats&oldid=412

Principal Authors: Sobolevnrm

APBS license

prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

External links

[1] http://agave.wustl.edu/
[2] http://www.wustl.edu/
[3] http://www.biochem.wustl.edu/
[4] http://www.ccb.wustl.edu/

Source: http://cardon.wustl.edu/MediaWiki/index.php?title=APBS_license&oldid=29

Principal Authors: Sobolevnrm