JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

MMPBSA.py: An Efficient Program for End-State Free Energy Calculations

Bill R. Miller, III,[†] T. Dwight McGee, Jr.,[†] Jason M. Swails,[†] Nadine Homeyer,[‡] Holger Gohlke,[‡] and Adrian E. Roitberg*,[†]

[†]Department of Chemistry, Quantum Theory Project, University of Florida, Gainesville, Florida 32611, United States

[‡]Institute of Pharmaceutical and Medicinal Chemistry, Department of Mathematics and Natural Sciences, Heinrich-Heine-University, 40225 Düsseldorf, Germany

ABSTRACT: MM-PBSA is a post-processing end-state method to calculate free energies of molecules in solution. *MMPBSA.py* is a program written in Python for streamlining end-state free energy calculations using ensembles derived from molecular dynamics (MD) or Monte Carlo (MC) simulations. Several implicit solvation models are available with *MMPBSA.py*, including the Poisson−Boltzmann Model, the Generalized Born Model, and the Reference Interaction Site Model. Vibrational frequencies may be calculated using normal mode or quasi-harmonic analysis to approximate the solute entropy. Specific interactions can also be dissected using free energy decomposition or alanine scanning. A parallel implementation significantly speeds up the calculation by dividing frames evenly across available processors. *MMPBSA.py* is an efficient, user-friendly program with the flexibility to accommodate the needs of users performing end-state free energy calculations. The source code can be downloaded at http://ambermd.org/ with AmberTools, released under the GNU General Public License.

1. INTRODUCTION

Free energy calculations have proven useful for a number of topics in computational biology, such as drug design and protein structure determination.[1,2] Several methods are available to calculate free energies, such as Free Energy Perturbation,[3] Replica Exchange Free Energy Perturbation,[4] and Thermodynamic Integration.[5] These methods, although theoretically rigorous, are computationally demanding and become prohibitively expensive as system size increases.

These methods converge poorly for complex systems, so the reaction coordinate is often divided into intermediate states. End-state free energy methods, on the other hand, reduce computational cost by eliminating the need for simulating these intermediate states. Modeling the solvent implicitly further reduces the computational cost by eliminating the noise caused by explicit solvent molecules. As a result, end-state methods have been used extensively in computational studies.[6−8] We will not discuss the advantages and disadvantages of applying end-state methods here; for a recent review, see refs 1 and 9.

This paper will focus on the implementation of end-state methods in *MMPBSA.py,* which is a program released with the open source AmberTools package.[10] *MMPBSA.py* has already been applied to several systems, including calculating the binding free energies of DNA-binding molecules[11] and HIV protease inhibitors.[12]

2. THEORY AND METHODS

Before describing the capabilities of *MMPBSA.py*, we will briefly review the theory and calculations of the end-state methods that *MMPBSA.py* is capable of performing.

2.1. Stability and Binding Free Energy Calculations. End-state calculations are frequently used for two types of analyses—calculating the relative stability of multiple conformations of a system and calculating the binding free energy in a noncovalently bound, receptor−ligand complex,[1] shown as thermodynamic cycles in Figure 1. Stability calculations compare the free energies of multiple conformations to determine their relative stability. If we consider the process of a biomolecule changing conformations from state A to state B, then the free energy associated with that conformational change is calculated according to eq 1:

$$\Delta G_{\mathrm{A}\rightarrow\mathrm{B,solvated}} = \Delta G_{\mathrm{B,solvated}} - \Delta G_{\mathrm{A,solvated}} \qquad (1)$$

Similarly, binding free energies are calculated by subtracting the free energies of the unbound receptor and ligand from the free energy of the bound complex, shown in eq 2:

$$\Delta G_{\mathrm{binding,solvated}} = \Delta G_{\mathrm{complex,\,solvated}} - [\Delta G_{\mathrm{receptor,\,solvated}} + \Delta G_{\mathrm{ligand,solvated}}] \qquad (2)$$

The free energy change associated with each term on the right-hand side of eqs 1 and 2 is estimated according to eq 3:

$$\Delta G_{\mathrm{solvated}} = E_{\mathrm{gas}} + \Delta G_{\mathrm{solvation}} - TS_{\mathrm{solute}} \qquad (3)$$

In eq 3, $\Delta G_{\mathrm{solvation}}$ represents a true free energy, since the solvent has been averaged, because of the use of an implicit solvent model. We then use $\Delta G_{\mathrm{solvated}}$ in eqs 1 and 2 for consistency; but, it is important to note that, at this point, we only have one structure for the solute. The gas-phase energies (E_{gas}) are often the molecular mechanical (MM) energies from the force field, while the solvation free energies ($\Delta G_{\mathrm{solvation}}$) are calculated using an implicit solvent model, and the entropic contribution (S) is estimated using known approximations. The free energy of solvation is further decomposed as the sum of

Received: May 22, 2012
Published: July 25, 2012

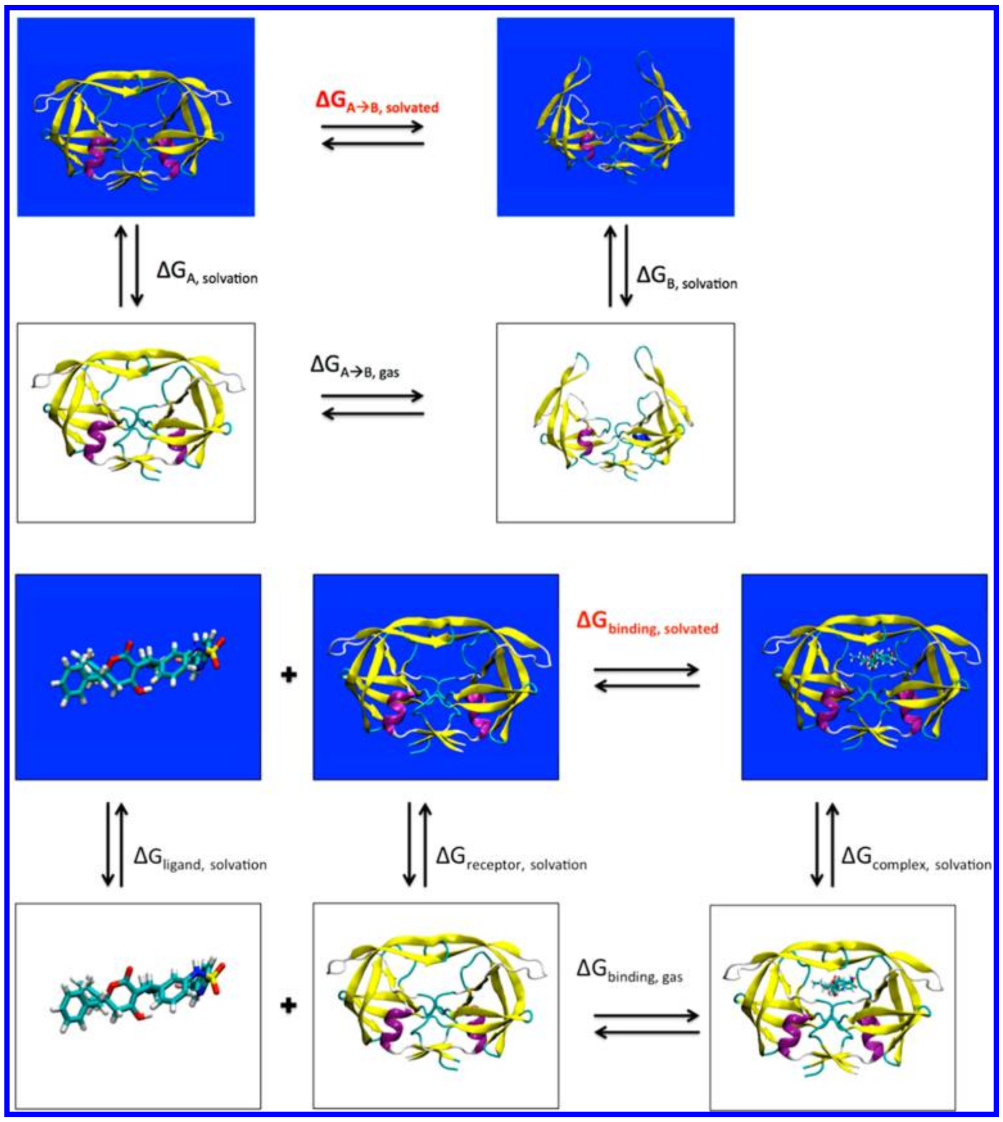

Figure 1. Thermodynamic cycles for (a) stability free energy calculations for conformation A and B, and (b) binding free energy calculations for a protein–ligand complex. Solvated systems are shown in blue boxes, while systems in the gas phase are in white boxes. Free energies that are directly calculated are shown in black, while the free energy of interest is shown in red.

electrostatic and nonpolar contributions. Several implicit solvent models are available to calculate the solvation free energies, including Generalized Born (GB),[1] Poisson–Boltzmann (PB),[1] and Reference Interaction Site Model (RISM).[13]

The energies described in the equations above are single point energies of the system. However, in practice, end-state calculations estimate these energies according to averages from an ensemble of representative structures. For example, expressing eq 3 in terms of averages yields eq 4:

$$\begin{aligned}\Delta G_{\text{solvated}} &\cong \langle E_{\text{gas}}\rangle + \langle \Delta G_{\text{solvation}}\rangle - T\langle S_{\text{solute}}\rangle \\ &= \frac{1}{N}\sum_{i=1}^{N} E_{i,\text{gas}} + \frac{1}{N}\sum_{i=1}^{N}\Delta G_{i,\text{solvation}} \\ &\quad - \frac{T}{N}\sum_{i=1}^{N} S_{i,\text{solute}}\end{aligned} \tag{4}$$

where i is the index of a particular frame and N is the total number of frames analyzed.

There are two approaches to generating the necessary ensembles for the bound and unbound state of binding energy calculations—all ensembles can be extracted from a single molecular dynamics (MD) or Monte Carlo (MC) trajectory of the bound complex, or trajectories can be generated for each state using separate simulations.[14] These approaches are called the *single trajectory protocol* (STP) and *multiple trajectory protocol* (MTP), respectively, and each approach has distinct advantages and disadvantages.

STP is less computationally expensive than MTP, because only a single trajectory is required to generate all three ensembles. Furthermore, the internal potential terms (e.g., bonds, angles, and dihedrals) cancel exactly in the STP, because the conformations in the bound and unbound ensembles are the same, leading to lower fluctuations and easier convergence in the binding free energy. The STP is appropriate if the receptor and ligand ensembles are comparable in the bound and unbound states. However, the conformations populating the unbound ensembles typically adopt strained configurations when extracted from the bound state ensemble, thereby overstabilizing the binding, compared to the MTP.

2.2. Free Energy Decomposition. Amber[10] provides several schemes to decompose calculated free energies into specific residue contributions using either the GB or PB implicit solvent models,[15] following the work of Gohlke et al.[6,15] Interactions can be decomposed for each residue by including only those interactions in which one of the residue's atoms is involved—a scheme called *per-residue* decomposition. Alternatively, interactions can be decomposed by specific residue pairs by including only those interactions in which one atom from each of the analyzed residues is participating—a scheme called *pairwise* decomposition. These decomposition schemes can provide useful insights into important interactions in free energy calculations.[6,15]

However, it is important to note that solvation free energies using GB and PB are not strictly pairwise decomposable, since the dielectric boundary defined between the protein and the bulk solvent is inherently nonlocal and depends on the arrangement of all atoms in space. Thus, care must be taken when interpreting free energy decomposition results.

An alternative way of decomposing free energies is to introduce specific mutations in the protein sequence and analyze how binding free energies or stabilities are affected.[16] Alanine scanning, which is a technique in which an amino acid in the system is mutated to alanine, can highlight the importance of the electrostatic and steric nature of the original side chain.[17] Assuming that the mutation will have a negligible effect on protein conformation, we can incorporate the mutation directly into each member of the original ensemble. This avoids the need to perform an additional MD or MC simulation to generate an ensemble for the mutant.

2.3. Entropy Calculations. The implicit solvent models used to calculate relative stability and binding free energies in end-state calculations often neglect some contributions to the solute entropy. If we assume that biological systems obey a rigid rotor model, we can calculate the translational and rotational entropies using standard statistical mechanical formulas,[18] and we can approximate the vibrational entropy contribution using one of two methods. First, the vibrational frequencies of normal modes can be calculated at various local minima of the potential energy surface—a method we will refer to as *nmode*.[18] Alternatively, the eigenvalues of the mass-weighted covariance matrix constructed from every member of the ensemble can be approximated as frequencies of global, orthogonal motions—a technique we will refer to as the *quasi-harmonic* approximation.[19] Using either the nmode or quasi-harmonic frequencies, we can sum the vibrational entropies of each mode calculated from standard formulas.[18]

Typically, nmode calculations are computationally demanding for large systems, because they require minimizing every frame, building the Hessian matrix, and diagonalizing it to obtain the vibrational frequencies (eigenvalues). Because of the Hessian diagonalization, normal-mode calculations scale as roughly $(3N)^3$, where N is the number of atoms in the system. While the quasi-harmonic approach is less computationally expensive, a large number of frames, or members of the ensemble, are typically needed to extrapolate the asymptotic limit of the total entropy for each ensemble, which increases the computational cost of the original simulation.[20]

Relative free energy calculations between related systems often assume that the solute entropy will be the same for each system being compared, thereby removing the need to explicitly calculate them.[16]

3. RESULTS AND DISCUSSION

Here, we will demonstrate many of the calculations that *MMPBSA.py* can perform. First, we will describe the general workflow for using *MMPBSA.py* to perform end-state free energy calculations, followed by specific examples.

3.1. General Workflow. *MMPBSA.py* is a program written in Python and *nab*[21] that streamlines the procedure of preparing and calculating free energies for an ensemble generated by MD or MC simulations whose general workflow is shown in Figure 2. *MMPBSA.py* builds upon original ideas by

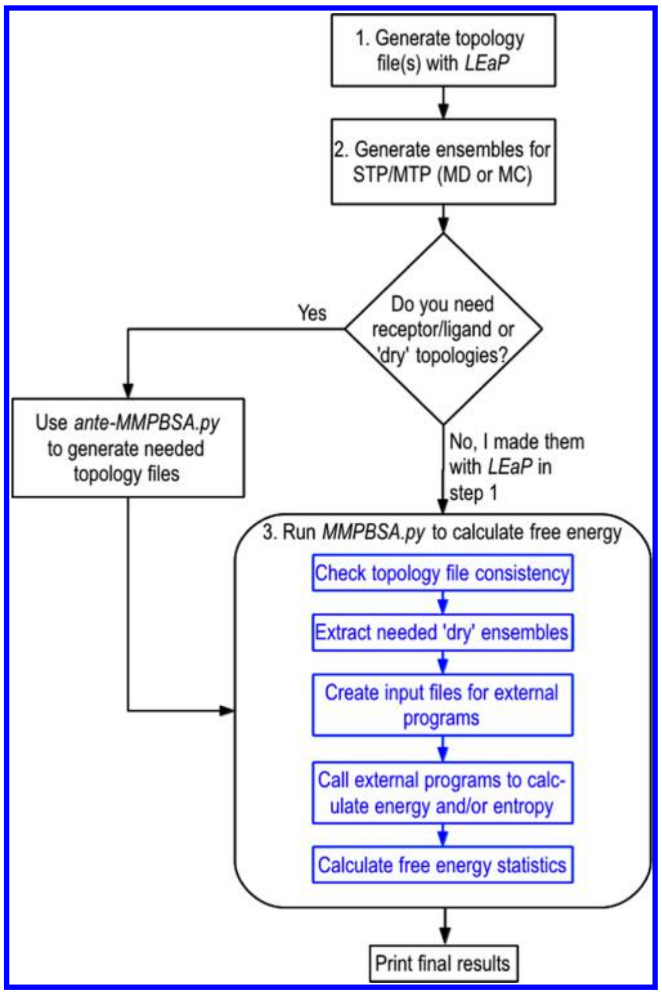
1. Generate topology file(s) with LEaP
2. Generate ensembles for STP/MTP (MD or MC)
Do you need receptor/ligand or 'dry' topologies?
Yes
Use ante-MMPBSA.py to generate needed topology files
No, I made them with LEaP in step 1
3. Run MMPBSA.py to calculate free energy
Check topology file consistency
Extract needed 'dry' ensembles
Create input files for external programs
Call external programs to calculate energy and/or entropy
Calculate free energy statistics
Print final results

Figure 2. General workflow for performing end-state calculations with *MMPBSA.py*. *LEaP* is a program in Amber used to create topology files for dynamics. The workflow shown in step 3 is the series of steps that *MMPBSA.py* automates. "Dry" topologies and ensembles are systems without explicit solvent that are subsequently treated using an implicit solvent model. "External programs" refers to the executables that perform the energy calculations (e.g., *sander*).

Massova and Kollman, as described in ref 16. The process of calculating binding free energies can be a tedious procedure that *MMPBSA.py* aims to shorten and simplify.

Python is a useful programming language for performing tasks that are not numerically intensive, and because it is available on virtually every platform, Python programs are highly portable. Nucleic Acid Builder[21] (*nab*), which is a molecule-based programming language included with AmberTools, contains functionality pertinent to building, manipulating, and performing energy calculations on biological systems, such as proteins and nucleic acids.

End-state calculations often require multiple *topology files* that contain the parameters corresponding to the force field. We recommend users run simulations using explicit solvent, which would require the user to provide both solvated and unsolvated topology files to *MMPBSA.py*. It is necessary that all topology files have a consistent set of parameters, especially for binding free energy calculations. *MMPBSA.py* checks the user's topology files prior to binding free energy calculations to prevent erroneous results due to inconsistencies that may not be immediately obvious. The Python utility *ante-MMPBSA.py* (released alongside *MMPBSA.py*) allows a user to easily create topology files with a consistent set of parameters, including changing the intrinsic implicit solvent radius set to fit the desired solvent model.

The use of *MMPBSA.py* is similar to that of Amber's MD engines *sander* and *pmemd*. The command-line flags common to both *MMPBSA.py* and the MD engines are identical, and input files are separated with similar, Fortran-style namelists, indicated with an ampersand (&) prefix.

The *MMPBSA.py* input file contains a &general namelist for variables that control general behavior. For example, variables that control the subset of frames analyzed (startframe, endframe, and interval) and the amount of information printed in the output file (verbose) are specified here. An example of this section is shown below.

```
General MMPBSA.py input file

&general

  startframe=1, endframe=100, interval=2,

  keep_files=0, verbose=1, strip_mask=:WAT:Cl-:Na+,

/
```

Users should avoid analyzing correlated structures from their simulations. Because the autocorrelation time of the free energy is system dependent, the autocorrelation function of the free energy can be calculated to determine the ideal frequency to extract snapshots to avoid analyzing correlated structures (e.g., 5 ps).[7] The keep_files variable controls which temporary files remain after the calculations conclude successfully, allowing users to optionally inspect the intermediate steps or regenerate the final output file using a different verbose value. By default, *MMPBSA.py* removes all known ions from the original trajectory file via the strip_mask variable. This can be modified in the input file; however, care must be taken if the user chooses to retain particular ions or water molecules during the free energy calculations, because the implicit solvent models utilized by *MMPBSA.py* are not parametrized to account for these species. Therefore, it is not recommended to include water or ions during the calculations.

The following sections introduce the other namelists allowed in the *MMPBSA.py* input file and their functionality.

3.2. Poisson–Boltzmann: MM-PBSA. Solvation free energies can be calculated using the Poisson–Boltzmann (PB) implicit solvent method with a nonpolar solvation term, based on the solvent accessible surface area (SASA),[8] by including the &pb namelist in the *MMPBSA.py* input file, demonstrated below.

```
MMPBSA.py input file for running PB

&general

  startframe=1, endframe=100, interval=2,

/

&pb

  istrng=0.1, exdi=80, indi=1.0,

  inp=2, cavity_surften=0.0378, cavity_offset=-0.5692,

  fillratio=4, scale=2.0,

  linit=1000, prbrad=1.4, radiopt=1,

/
```

Users can specify external thermodynamic variables based on experimental conditions to control the calculation, such as the ionic strength (istrng) and the external and internal dielectric constants (exdi and indi, respectively). Variables specific to the PB calculation are shown on the final three lines of the &pb section of the example input file above. The default method for calculating nonpolar solvation free energies (inp) and its associated variables (cavity_surften and cavity_offset) are recommended, based on a previous optimization of these parameters.[22] The default fillratio value of 4.0 is recommended to prevent errors for small molecules lying outside the focusing finite-difference grid that occurs when fillratio is set too low. Scale is the number of PB grid points per angstrom, and may be adjusted to obtain coarser or finer resolution, although increasing the resolution comes at the cost of computational time. The default value of 2.0 for the scale control variable was determined as a compromise between speed and accuracy. The suggested value of 1000 for the number of iterations to perform of the linear PB equation (linit) is usually sufficient to reach convergence. A 1.4 Å probe radius (prbrad) is recommended for simulations performed with water as the solvent to model the size of a typical water molecule. The optimized radii set (radiopt) described previously[23] shows improved accuracy over other sets and, thus, is recommended for most users.

The PB solvation free energy can be calculated using two possible programs. The default solver is *pbsa*,[24] whose routines can be called from either a *nab* program (default) or *sander*, although users can access the third-party *Adaptive Poisson–Boltzmann Solver*[25] (*apbs*) by linking *apbs* with *sander* using *iAPBS*.[26]

3.3. Generalized Born: MM-GBSA. The Generalized Born (GB) implicit solvent method with a SASA term can be used to calculate the solvation free energy[6] by including the &gb namelist in the *MMPBSA.py* input file, demonstrated below.

```
MMPBSA.py input file for running GB

&general

  startframe=10, interval=2, endframe=2010,

/

&gb

  igb=5, saltcon=0.1,

/
```

The GB calculation has its own set of control variables, such as salt concentration (saltcon) and the specific GB model (igb). Amber provides five different GB models[27–30] that can be used with *MMPBSA.py*, depending on user preference. Similar to PB, the GB solvation free energies can be solved using either a *nab* program (default) or *sander*. For analysis of proteins, we recommend setting igb to 8 (*sander* only), based on recent results, showing an increased stability for salt bridges with this GB model.[30] However, a value of 2 for igb is preferred for many users analyzing proteins without access to *sander*.[31]

MMPBSA.py can use *sander* to treat part of the system in a MM-GBSA calculation quantum mechanically using a hybrid quantum-mechanical/molecular-mechanical (QM/MM) Hamiltonian[32] with the ifqnt control variable. The residues treated quantum mechanically (qm_residues) are defined in the &gb namelist of the *MMPBSA.py* input file, as demonstrated below.

```
MMPBSA.py input file for running QM/MMGBSA
&general
  startframe=7, interval=9, endframe=2011,
/
&gb
  igb=5, saltcon=0.1, ifqnt=1, qmcharge_com=0,
  qmcharge_lig=0, qm_residues=241,
  qm_theory='PM3',
/
```

The charge of the region treated quantum mechanically must be specified for both the complex (qmcharge_com) and ligand (qmcharge_lig). Several semiempirical Hamiltonians are available in *MMPBSA.py* for the QM region,[33] including AM1,[34] MNDO,[35] and PM3.[36]

3.4. Reference Interaction Site Model: MM/3D-RISM. *MMPBSA.py* can also use the Reference Interaction Site Model (RISM)[13] to calculate solvation free energies through a *nab* program. Specifically, three-dimensional RISM (3D-RISM) determines the equilibrium distribution of solvent around the solute to calculate thermodynamic properties. The MM/3D-RISM method has its own namelist in the *MMPBSA.py* input file with its own control variables. An example is shown below.

```
MMPBSA.py input file for running MM/3D-RISM
&general
  endframe=200,
/
&rism
  closure=kh, polardecomp=1,
/
```

Three different closure relations (closure) are available to approximate the RISM integrals: the hypernetted-chain approximation (HNC), Kovalenko-Hirata (KH), and the partial series expansion of order-*n* (PSE-*n*).[37] While 3D-RISM can directly decompose the solvation free energy into the polar and nonpolar (cavity and dispersion) terms (polardecomp), this decomposition nearly doubles the computational cost of the calculation, and 3D-RISM calculations already take substantially longer than PB or GB.

3.5. Free Energy Decomposition. Free energy decomposition can provide information about the local interactions of a system in addition to global free energies.[6] *MMPBSA.py* can calculate per-residue and pairwise energy decompositions with *sander*. Decomposition must be combined with GB and/or PB in the *MMPBSA.py* input file, and control variables for both per-residue and pairwise decomposition schemes are specified in the &decomp namelist, as shown below.

```
MMPBSA.py input file for running per-residue decomp
&general
  startframe=6, interval=26, endframe=2007,
/
&gb
  igb=5, saltcon=0.1,
/
&decomp
  idecomp=1, print_res='1-10,200-241',
/
```

Per-residue decomposition (idecomp options 1 and 2) estimates the contribution of each residue to the total free energy. Pairwise energy decomposition (idecomp options 3 and 4) estimates the interaction energy of specific residue pairs. By default, *MMPBSA.py* will print the decomposition results for all residues of the system, but a subset can be specified using print_res to reduce the amount of data printed. We recommend new users perform a decomposition analysis with GB solvent models before attempting to use the PB models, because of the difficulty and time-consuming nature of the PB decomposition analysis. Furthermore, we reemphasize that decomposition analysis is not pairwise decomposable since the dielectric boundary defined between the protein and the bulk solvent is inherently nonlocal and depends on the arrangement of all atoms in space.

3.6. Alanine Scanning. Alanine scanning is an alternative to decomposition analysis that involves mutating a single amino acid in the system to alanine. Alanine scanning is enabled by including the &alanine_scanning namelist in the *MMPBSA.py* input file and must be specified with at least one type of calculation to be performed on the mutated structure. An example of alanine scanning using GB is shown below.

```
MMPBSA.py input file for running alanine scanning
&general
  startframe=1, endframe=200, interval=5,
/
&gb
  igb=5, saltcon=0.1,
/
&alanine_scanning
  mutant_only=1,
/
```

The free energy can be calculated for the mutant trajectory only (mutant_only = 1) or both the mutant and original trajectories (mutant_only = 0). When alanine scanning is enabled, *MMPBSA.py* determines the mutated residue by comparing the original and mutant topology files. *MMPBSA.py* then creates a mutant trajectory by modifying the original atomic coordinates of the mutated residue to the atomic positions of alanine, discarding extraneous atoms and adjusting bond lengths to their equilibrium values. Once the mutated trajectory is created, the normal workflow is resumed.

3.7. Solute Entropy. Including solute entropy with the energies calculated in the sections above is necessary to obtain true free energies. *MMPBSA.py* assumes a rigid rotor model and calculates the rotational and translational solute entropies according to standard formulas. Vibrational entropies are calculated from the frequencies of global motions, which are obtained via normal-mode analysis (nmode) or quasi-harmonic analysis.

3.7.1. Normal Mode Analysis. In nmode, *MMPBSA.py* uses a *nab* program to minimize frames to a local minimum, then constructs and diagonalizes the Hessian matrix to obtain the vibrational frequencies. Control variables for nmode calculations are specified in the &nmode namelist of the *MMPBSA.py* input file, as demonstrated below.

```
MMPBSA.py input file for running normal mode analysis

&general

  startframe=1, endframe=200, interval=5,

/

&nmode

  maxcyc=10000, drms=0.001,

  nmode_igb=1, nmode_istrng=0.1,

/
```

Each frame is minimized to a local minimum using the *xmin* minimizer in *nab* prior to building the Hessian matrix. Details of the minimization, such as the number of minimization cycles (maxcyc) and the convergence criteria (drms), can be adjusted in the input file; however, the default values are sufficient for most systems that are able to be minimized. The normal modes can be calculated in the gas phase or in implicit solvent using a GB model (nmode_igb). Nmode calculations can be time-consuming, and they require large amounts of computational resources. For this reason, we advise using only a small number of frames for nmode analysis compared to typical PB and GB analyses. *MMPBSA.py* also provides control variables (nmstartframe, nmendframe, and nminterval) in the &nmode section that allow users to define a subset of the frames analyzed by PB or GB for nmode calculations. This provides the users with the ability to combine the PB and/or GB results with the entropic contributions using a different number of frames with only a single call to *MMPBSA.py*.

3.7.2. Quasi-harmonic Approximation. Entropy can also be calculated using the quasi-harmonic approximation in which all analyzed frames are aligned to the average structure of the ensemble. The mass-weighted covariance matrix of all atoms is calculated from these frames and then diagonalized to obtain pseudo-vibrational frequencies. Quasi-harmonic calculations are run when the control variable entropy is set to 1 in the &general namelist of the *MMPBSA.py* input file. Because minimizations are not required and only one matrix diagonalization is necessary, quasi-harmonic calculations are faster than normal mode calculations. However, obtaining converged results can be difficult, because the method depends on the number of frames used to calculate the covariance matrix.

3.8. Running in Parallel. *MMPBSA.py* is implemented in parallel, so users with access to multiple processors can speed up their calculations. *MMPBSA.py.MPI* is the parallel implementation of *MMPBSA.py* that uses a Message Passing Interface (MPI) for Python (*mpi4py*). Since energy calculations for each frame are independent, the calculation can be trivially parallelized, given enough available processors. *MMPBSA.py.MPI* divides frames evenly across all processors, which allows calculations using many frames to scale better than if *MMPBSA.py* invoked parallel executables to calculate free energies. However, perfect scaling is not attained, because

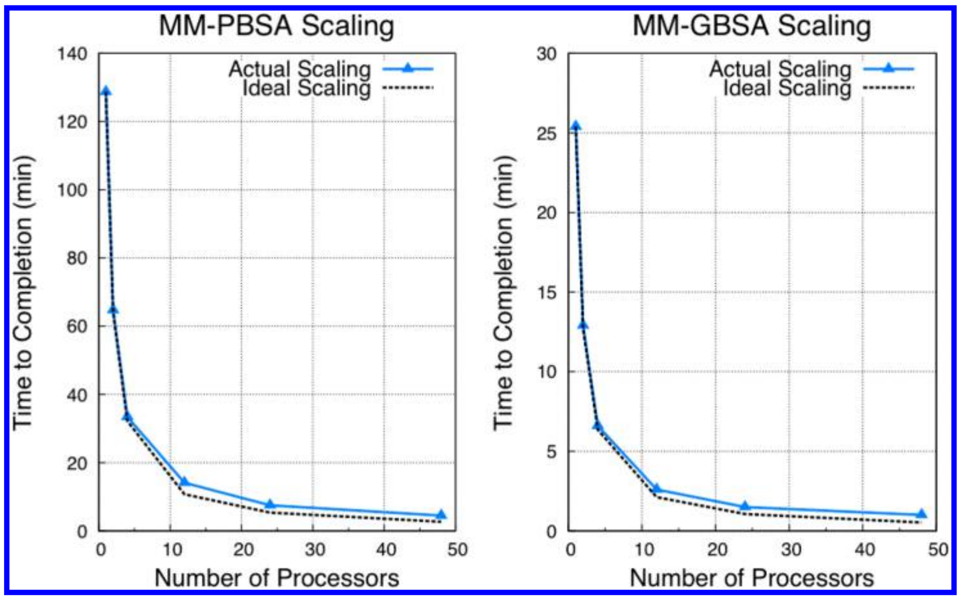

Figure 3. *MMPBSA.py* scaling comparison for MM-PBSA and MM-GBSA calculations on 200 frames of a 5910-atom complex. Times shown are the times required for the calculation to finish. Note that MM-GBSA calculations are ~5 times faster than MM-PBSA calculations. All calculations were performed on NICS Keeneland (2 Intel Westmere 6-core CPUs per node, QDR infiniband interconnect).

certain setups tasks and file input/output can only be done with a single processor. Figure 3 demonstrates scaling for a sample MM-PBSA and MM-GBSA calculation.

3.9. Differences to mm_pbsa.pl. Prior to the development of *MMPBSA.py*, end-state free energy calculations could be performed in Amber using a script written in Perl called *mm_pbsa.pl*. Both programs allow users to perform free energy calculations using the STP and MTP, although *MMPBSA.py* offers more flexibility when using the MTP. Both programs have the ability to use different PB and GB models contained within Amber and estimate entropic contributions. Finally, *MMPBSA.py* and *mm_pbsa.pl* can run free energy calculations in parallel, although only *MMPBSA.py* can run on distributed memory systems (i.e., on multiple nodes connected over a network).

Despite their obvious similarities, there are many differences that exist in their accessibility, implementation, and capabilities. *MMPBSA.py* is available free of charge alongside AmberTools, while an Amber license is necessary to obtain *mm_pbsa.pl*. The usage of *MMPBSA.py* is intended to resemble Amber's MD engines for ease of the user, while *mm_pbsa.pl*'s input file and usage has its own syntax. Only *MMPBSA.py* has an intuitive mechanism for guessing the ligand and receptor masks of a complex based on the topology files provided and analyzes topology files for parameter consistency. Furthermore, only *MMPBSA.py* can calculate entropic contributions to the free energy using the quasi-harmonic approximation. An interface to external PB solvers such as *Delphi*, *MEAD*, and *UHBD* is available with *mm_pbsa.pl* only, although both can use *apbs*. Surface-area nonpolar solvation free energies calculated using the *molsurf* program is also only available with *mm_pbsa.pl*. *MMPBSA.py* allows users to provide their own input files for external programs, which gives users the ability to adjust all parameters, not just the variables described in the *MMPBSA.py* manual; in comparison, *mm_pbsa.pl* has no similar functionality without directly altering the source code. Finally, QM/MM-GBSA and MM/3D-RISM calculations are only available through the *MMPBSA.py* implementation.

4. CONCLUSIONS

MMPBSA.py is a program written in Python and *nab* that allows users to easily and efficiently calculate free energies from MD and MC simulations. Using *MMPBSA.py* simplifies the end-state free energy calculation workflow by automating many tasks and checking topology files for consistency to minimize user error. The command line syntax and input file structure mimics those of other Amber programs, thereby reducing the learning curve. Accompanying test cases provide users with example inputs for performing each supported type of calculation. *MMPBSA.py.MPI* can be run in parallel to increase the speed of end-state free energy calculations.

MMPBSA.py is flexible enough to accommodate the needs of most users. Several methods may be used to calculate the free energy of solvation, such as PB, GB, and RISM. Furthermore, when using a GB model, part of the system can be treated quantum mechanically. Specific interactions can be analyzed using alanine scanning or free energy decomposition. Solute entropy is approximated using a rigid rotor model in which the vibrational frequencies can be calculated using either normal mode or quasi-harmonic approaches. For increased flexibility, advanced users can supply custom input files to control all variables, including those not available in the *MMPBSA.py* input file.

MMPBSA.py is released with AmberTools under the GNU General Public License (version 2), available for download from http://ambermd.org/. Documentation and tutorials are also available on the Amber website.

AUTHOR INFORMATION

Corresponding Author

*E-mail: roitberg@ufl.edu.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We would like to acknowledge financial support from the NSF (No. CHE-0822935). J.M.S. acknowledges financial support from the NSF GRFP award, and T.D.M. acknowledges financial support from the NSF Bridge to Doctorate and FAMU Feeder Fellowship. H.G. acknowledges financial support by the initiative "Fit for Excellence" at the Heinrich-Heine-University, Düsseldorf. We would like to thank David Case and Tyler Luchko for their help with the 3D-RISM settings and support, as well as Ray Luo and Qin Cai for their help with PB settings and support. We would also like to acknowledge the early users of *MMPBSA.py* for their suggestions, comments, and bug reports.

REFERENCES

(1) Homeyer, N.; Gohlke, H. *Mol. Inform.* **2012**, *31*, 114−122.
(2) de Ruiter, A.; Oostenbrink, C. *Curr. Opin. Chem. Biol.* **2011**, *15*, 547−552.
(3) Zwanzig, R. W. *J. Chem. Phys.* **1954**, *22*, 1420−1426.
(4) Meng, Y.; Dashti, D. S.; Roitberg, A. E. *J. Chem. Theory Comput.* **2011**, *7*, 2721−2727.
(5) McQuarrie, D. A. *Statistical Thermodynamics*; Harper and Row: New York, 1973.
(6) Gohlke, H.; Kiel, C.; Case, D. A. *J. Mol. Biol.* **2003**, *330*, 891−913.
(7) Genheden, S.; Ryde, U. *J. Comput. Chem.* **2010**, *31*, 837−846.
(8) Srinivasan, J.; Cheatham, T. E.; Cieplak, P.; Kollman, P. A.; Case, D. A. *J. Am. Chem. Soc.* **1998**, *120*, 9401−9409.
(9) Steinbrecher, T.; Labahn, A. *Curr. Med. Chem.* **2010**, *17*, 767−785.
(10) Case, D. A.; Darden, T. A.; Cheatham, T. E., III; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Walker, R. C.; Zhang, W.; Merz, K. M.; Roberts, B. P.; Hayik, S.; Roitberg, A. E.; Seabra, G.; Swails, J. M.; Kolossváry, I.; Wong, K. F.; Paesani, F.; Vanicek, J.; Wolf, R. M.; Liu, J.; Wu, X.; Brozell, S. R.; Steinbrecher, T.; Gohlke, H.; Cai, Q.; Ye, X.; Wang, J.; Hsieh, M.-J.; Cui, G.; Roe, D. R.; Mathews, D. H.; Seetin, M. G.; Salomon-Ferrer, R.; Sagui, C.; Babin, V.; Luchko, T.; Gusarov, S.; Kovalenko, A.; Kollman, P. A. *AMBER 12*; University of California: San Francisco, CA, 2012.
(11) Vargiu, A. V.; Magistrato, A. *Inorg. Chem.* **2012**, *51*, 2046−2057.
(12) Kar, P.; Knecht, V. *J. Phys. Chem. B* **2012**, *116*, 2605−2614.
(13) Genheden, S.; Luchko, T.; Gusarov, S.; Kovalenko, A.; Ryde, U. *J. Phys. Chem. B* **2010**, *114*, 8505−8516.
(14) Wang, J.; Hou, T.; Xu, X. *Curr. Comput.-Aided Drug Des.* **2006**, *2*, 287−306.
(15) Metz, A.; Pfleger, C.; Kopitz, H.; Pfeiffer-Marek, S.; Baringhaus, K.-H.; Gohlke, H. *J. Chem. Inf. Model.* **2011**, *52*, 120−133.
(16) Massova, I.; Kollman, P. *Perspect. Drug Discovery Des.* **2000**, *18*, 113−135.
(17) Massova, I.; Kollman, P. A. *J. Am. Chem. Soc.* **1999**, *121*, 8133−8143.
(18) McQuarrie, D. A. *Statistical Mechanics*; 2nd ed.; University Science Books, 2000.
(19) Brooks, B. R.; Janežič, D.; Karplus, M. *J. Comput. Chem.* **1995**, *16*, 1522−1542.

(20) Wang, J.; Morin, P.; Wang, W.; Kollman, P. A. *J. Am. Chem. Soc.* **2001**, *123*, 5221−5230.
(21) Macke, T. J.; Case, D. A. In *Molecular Modeling of Nucleic Acids*; ACS Symposium Series 682; American Chemical Society: Washington, DC, 1997; Vol. *682*, pp 379−393.
(22) Tan, C.; Tan, Y.; Luo, R. *J. Phys. Chem. B* **2007**, *111*, 12263−12274.
(23) Tan, C.; Yang, L.; Luo, R. *J. Phys. Chem. B* **2006**, *110*, 18680−18687.
(24) Luo, R.; David, L.; Gilson, M. K. *J. Comput. Chem.* **2002**, *23*, 1244−1253.
(25) Baker, N. A.; Sept, D.; Joseph, S.; Holst, M. J.; McCammon, J. A. *Proc. Natl. Acad. Sci. U.S.A.* **2001**, *98*, 10037−10041.
(26) Konecney, R.; Baker, N. A.; McCammon, J. A. *Comput. Sci. Disc.*, submitted.
(27) Onufriev, A.; Bashford, D.; Case, D. A. *Proteins* **2004**, *55*, 383−394.
(28) Mongan, J.; Simmerling, C.; McCammon, J. A.; Case, D. A.; Onufriev, A. *J. Chem. Theory Comput.* **2007**, *3*, 156−169.
(29) Hawkins, G. D.; Cramer, C. J.; Truhlar, D. G. *Chem. Phys. Lett.* **1995**, *246*, 122−129.
(30) Simmerling, C. Manuscript in preparation.
(31) Hou, T.; Wang, J.; Li, Y.; Wang, W. *J. Chem. Inf. Model.* **2011**, *51*, 69−82.
(32) Gleeson, M. P.; Gleeson, D. *J. Chem. Inf. Model.* **2009**, *49*, 1437−1448.
(33) Walker, R. C.; Crowley, M. F.; Case, D. A. *J. Comput. Chem.* **2008**, *29*, 1019−1031.
(34) Dewar, M. J. S.; Zoebisch, E. G.; Healy, E. F.; Stewart, J. J. P. *J. Am. Chem. Soc.* **1985**, *107*, 3902−3909.
(35) Dewar, M. J. S.; Thiel, W. *J. Am. Chem. Soc.* **1977**, *99*, 4899−4907.
(36) Stewart, J. J. P. *J. Comput. Chem.* **1989**, *10*, 209−220.
(37) Hansen, J. P.; McDonald, I. R. *Theory of Simple Liquids*; Academic Press: London, 2006.