
open3SPN2 Documentation

Release 0.1

Carlos Bueno

Sep 15, 2021

CONTENTS:

1	Installation	3
1.1	Requirements	3
1.2	From source code	3
1.3	From conda	3
2	Tutorial	5
2.1	DNA from sequence	5
2.2	Protein DNA system	7
3	Classes and functions	15
3.1	DNA	15
3.2	System	16
3.3	DNA Forces	16
3.4	DNA - Protein Forces	16
3.5	Utils	16
3.6	Tests	17
3.7	Exceptions	17
4	Indices and tables	19
	Index	21

open3SPN2 implements the 3SPN2 and 3SPN2.C forcefields in openmm. This library is intended to be used for DNA simulations and DNA-protein simulations.

INSTALLATION

1.1 Requirements

X3DNA is needed on the 3SPN2.C forcefield in order to calculate the equilibrium bonds, angles and dihedrals. It is also needed to create a structure from sequence. Please follow their instructions for installation and add the installation directory to the X3DNA environment variable.

open3SPN also requires the following python libraries:

- **biopython**
- **pandas**
- **numpy**
- **scipy**
- **mdtraj**
- **openmm**
- **pdbfixer**
- **nose**

1.2 From source code

The source code is available at <https://github.com/cabb99/open3spn2/>

1.3 From conda

```
$ conda install -c wolynes-lab open3spn2
```


TUTORIAL

“What I cannot create, I do not understand.” — Feynman

2.1 DNA from sequence

This section allows you to quickly simulate a piece of DNA.

First import the open3SPN2 module.

```
import open3SPN2
```

If you obtain a `ModuleNotFoundError` error then you may need to check that `open3SPN2` is in the installation path as detailed in *Installation*.

After importing the module, follow the next steps to create a DNA system and add the 3SPN2 forces. Make sure you have installed X3DNA before this step.

```
# Initialize the DNA from a sequence.
# DNA type can be changed to 'A' or 'B'

seq='ATACAAAGGTGCGAGGTTTCTATGCTCCCACG'
dna=open3SPN2.DNA.fromSequence(seq,dna_type='B_curved')

# Compute the topology for the DNA structure.
# Since the dna was generated from the sequence using X3DNA,
# it is not necessary to recompute the geometry.

dna.computeTopology(template_from_X3DNA=False)

# Create the system.
# To set periodic boundary conditions (periodicBox=[50,50,50]).
# The periodic box size is in nanometers.
dna.periodic=False
s=open3SPN2.System(dna, periodicBox=None)

#Add 3SPN2 forces
s.add3SPN2forces(verbose=True)
```

```
Bond
Angle
Stacking
```

(continues on next page)

(continued from previous page)

Dihedral
BasePair
CrossStacking
Exclusion
Electrostatics

To set up the simulation you will need the openmm package as detailed in *Installation*.

```
import simtk.openmm
import simtk.openmm.app
import simtk.unit
import sys
import numpy as np

#Initialize Molecular Dynamics simulations
s.initializeMD(temperature=300 * simtk.unit.kelvin,platform_name='OpenCL')
simulation=s.simulation

#Set initial positions
simulation.context.setPositions(s.coord.getPositions())

energy_unit=simtk.openmm.unit.kilojoule_per_mole
#Total energy
state = simulation.context.getState(getEnergy=True)
energy = state.getPotentialEnergy().value_in_unit(energy_unit)
print('TotalEnergy',round(energy,6),energy_unit.get_symbol())

#Detailed energy
energies = {}
for force_name, force in s.forces.items():
    group=force.getForceGroup()
    state = simulation.context.getState(getEnergy=True, groups=2**group)
    energies[force_name] =state.getPotentialEnergy().value_in_unit(energy_unit)

for force_name in s.forces.keys():
    print(force_name, round(energies[force_name],6),energy_unit.get_symbol())
```

```
TotalEnergy -2046.579102 kJ/mol
Bond 2.1e-05 kJ/mol
Angle 0.001745 kJ/mol
Stacking -596.408569 kJ/mol
Dihedral -839.999756 kJ/mol
BasePair -518.252075 kJ/mol
CrossStacking -135.335464 kJ/mol
Exclusion 0.11901 kJ/mol
Electrostatics 43.296059 kJ/mol
```

Please make sure that the energies obtained coincide with the energies shown here. Also you can check the energy obtained using other [platforms](#) by changing OpenCL to Reference, CUDA or CPU.

```
#Add simulation reporters
dcd_reporter=simtk.openmm.app.DCDReporter(f'output.dcd', 1000)
```

(continues on next page)

(continued from previous page)

```
energy_reporter=simtk.openmm.app.StateDataReporter(sys.stdout, 1000, step=True,time=True,
                                                    potentialEnergy=True,
                                                    ↪temperature=True)
simulation.reporters.append(dcd_reporter)
simulation.reporters.append(energy_reporter)

#Run simulation
simulation.step(10000)
```

```
#"Step","Time (ps)","Potential Energy (kJ/mole)","Temperature (K)"
1000,2.000000000000000013,-1651.121826171875,304.6066812070446
2000,3.9999999999999781,-1646.61328125,309.5945230237376
3000,5.9999999999999561,-1661.6788330078125,318.46432160647703
4000,7.9999999999999341,-1676.956298828125,268.04874840144447
5000,10.000000000000009,-1629.8892822265625,271.8654648104738
6000,12.0000000000000677,-1622.474853515625,312.1083112301662
7000,14.0000000000001345,-1704.033203125,283.5259033832464
8000,16.000000000000201,-1608.751708984375,281.82371603990293
9000,18.000000000000902,-1623.486572265625,283.86225823944585
10000,19.99999999999794,-1671.9105224609375,296.18167366285144
```

The forces and system in open3SPN can be treated as forces and systems from openmm. Please refer to [openmm](#) documentation to learn more about running the simulations or adding forces.

2.2 Protein DNA system

You can find this example on the [examples/Protein_DNA](#) folder. For this example you need to download the structure of the Lambda repressor-operator complex [1mlb.pdb](#). You will also need to have installed the [openAWSEM](#) library.

```
# If you want to specify the package address
# you can add them to the PYTHONPATH environment variable.
# Also you can add them on the run time uncommenting the lines below
# import sys
# open3SPN2_HOME = '/Users/weilu/open3spn2/'
# openAWSEM_HOME = '/Users/weilu/openmmawsem/'
# sys.path.insert(0,open3SPN2_HOME)
# sys.path.insert(0,openAWSEM_HOME)
```

```
#Import openAWSEM, open3SPN2 and other libraries
import open3SPN2
import fFAWSEM

import pandas
import numpy as np
import simtk.openmm

from functools import partial
import sys
```

```
#Fix the system (adds missing atoms)
fix=open3SPN2.fixPDB("1lmb.pdb")
```

```
#Create a table containing both the proteins and the DNA
complex_table=open3SPN2.pdb2table(fix)
```

```
# Create a single memory file
ffAWSEM.create_single_memory(fix)
```

```
#Generate a coarse-grained model of the DNA molecules
dna_atoms=open3SPN2.DNA.CoarseGrain(complex_table)
```

```
#Generate a coarse-grained model of the Protein molecules
protein_atoms=ffAWSEM.Protein.CoarseGrain(complex_table)
```

```
#Merge the models
Coarse=pandas.concat([protein_atoms,dna_atoms],sort=False)
Coarse.index=range(len(Coarse))
Coarse['serial']=list(Coarse.index)
```

```
#Save the protein_sequence
ffAWSEM.save_protein_sequence(Coarse,sequence_file='protein.seq')
```

```
# Create a merged PDB
ffAWSEM.writePDB(Coarse,'clean.pdb')
```

```
#Create the merged system

pdb=simtk.openmm.app.PDBFile('clean.pdb')
top=pdb.topology
coord=pdb.positions
forcefield=simtk.openmm.app.ForceField(ffAWSEM.xml,open3SPN2.xml)
s=forcefield.createSystem(top)
```

```
dna=open3SPN2.DNA.fromCoarsePDB('clean.pdb')
with open('protein.seq') as ps:
    protein_sequence_one=ps.readlines()[0]
protein=ffAWSEM.Protein.fromCoarsePDB('clean.pdb',sequence=protein_sequence_one)
dna.periodic=False
protein.periodic=False
```

```
#Create the DNA and Protein Objects
dna=open3SPN2.DNA.fromCoarsePDB('clean.pdb')
with open('protein.seq') as ps:
    protein_sequence_one=ps.readlines()[0]
protein=ffAWSEM.Protein.fromCoarsePDB('clean.pdb',sequence=protein_sequence_one)
dna.periodic=False
protein.periodic=False

#Copy the AWSEM parameter files
ffAWSEM.copy_parameter_files()
```

```

#Clear Forces from the system (optional)
keepCMMotionRemover=True
j=0
for i, f in enumerate(s.getForces()):
    if keepCMMotionRemover and i == 0 and f.__class__ == simtk.openmm.CMMotionRemover:
        # print('Kept ', f.__class__)
        j += 1
        continue
    else:
        # print('Removed ', f.__class__)
        s.removeForce(j)
if keepCMMotionRemover == False:
    assert len(s.getForces()) == 0, 'Not all the forces were removed'
else:
    assert len(s.getForces()) <= 1, 'Not all the forces were removed'

```

```

#Initialize the force dictionary
forces={}
for i in range(s.getNumForces()):
    force = s.getForce(i)
    force_name="CMMotionRemover"

#Add 3SPN2 forces
for force_name in open3SPN2.forces:
    print(force_name)
    force = open3SPN2.forces[force_name](dna)
    if force_name in ['BasePair','CrossStacking']:
        force.addForce(s)
    else:
        s.addForce(force)
    forces.update({force_name:force})

#Add AWSEM forces
openAWSEMforces = dict(Connectivity=ffAWSEM.functionTerms.basicTerms.con_term,
                        Chain=ffAWSEM.functionTerms.basicTerms.chain_term,
                        Chi=ffAWSEM.functionTerms.basicTerms.chi_term,
                        Excl=ffAWSEM.functionTerms.basicTerms.excl_term,
                        rama=ffAWSEM.functionTerms.basicTerms.rama_term,
                        rama_pro=ffAWSEM.functionTerms.basicTerms.rama_proline_term,
                        contact=ffAWSEM.functionTerms.contactTerms.contact_term,
                        frag = partial(ffAWSEM.functionTerms.templateTerms.fragment_
memory_term,
                                frag_file_list_file="./single_fragments.mem",
                                npy_frag_table="./single_fragments.npy",
                                UseSavedFragTable=False,
                                k_fm=0.04184/3),
                        beta1 = ffAWSEM.functionTerms.hydrogenBondTerms.beta_term_1,
                        beta2 = ffAWSEM.functionTerms.hydrogenBondTerms.beta_term_2,
                        beta3 = ffAWSEM.functionTerms.hydrogenBondTerms.beta_term_3,
                        pap1 = ffAWSEM.functionTerms.hydrogenBondTerms.pap_term_1,
                        pap2 = ffAWSEM.functionTerms.hydrogenBondTerms.pap_term_2,
                        )

```

(continues on next page)

(continued from previous page)

```

protein.setup_virtual_sites(s)

#Add DNA-protein interaction forces
for force_name in open3SPN2.protein_dna_forces:
    print(force_name)
    force = open3SPN2.protein_dna_forces[force_name](dna,protein)
    s.addForce(force)
    forces.update({force_name: force})

#Fix exlussions
for force_name in openAWSEMforces:
    print(force_name)
    if force_name in ['contact']:
        force = openAWSEMforces[force_name](protein, withExclusion=False,periodic=False)
        print(force.getNumExclusions())
        open3SPN2.addNonBondedExclusions(dna,force)
        print(force.getNumExclusions())
    elif force_name in ['Excl']:
        force = openAWSEMforces[force_name](protein)
        print(force.getNumExclusions())
        open3SPN2.addNonBondedExclusions(dna,force)
        print(force.getNumExclusions())
    else:
        force = openAWSEMforces[force_name](protein)
    s.addForce(force)
    forces.update({force_name: force})

```

```

Bond
Angle
Stacking
Dihedral
BasePair
CrossStacking
Exclusion
Electrostatics
ExclusionProteinDNA
ElectrostaticsProteinDNA
Connectivity
Chain
Chi
Excl
1205
1844
rama
rama_pro
contact
Number of atom: 1171 Number of residue: 179
Contact cutoff 1.0 nm
NonbondedMethod: 1
0
639
frag

```

(continues on next page)

(continued from previous page)

```

Loading Fragment files(Gro files)
Saving fragment table as npy file to speed up future calculation.
All gro files information have been stored in the ./single_frags.npy.
You might want to set the 'UseSavedFragTable'=True to speed up the loading next time.
But be sure to remove the .npy file if you modify the .mem file. otherwise it will keep
↳using the old frag memeory.
beta1
beta_1 term ON
beta2
beta_2 term ON
beta3
beta_3 term ON
pap1
pap_1 term ON
No ssweight given, assume all zero
pap2
pap_2 term ON
No ssweight given, assume all zero

```

```

# Set up the simulation
temperature=300 * simtk.openmm.unit.kelvin
platform_name='OpenCL' #'Reference','CPU','CUDA', 'OpenCL'

integrator = simtk.openmm.LangevinIntegrator(temperature, 1 / simtk.openmm.unit.
↳picosecond, 2 * simtk.openmm.unit.femtoseconds)
platform = simtk.openmm.Platform.getPlatformByName(platform_name)
simulation = simtk.openmm.app.Simulation(top,s, integrator, platform)
simulation.context.setPositions(coord)
energy_unit=simtk.openmm.unit.kilojoule_per_mole
state = simulation.context.getState(getEnergy=True)
energy = state.getPotentialEnergy().value_in_unit(energy_unit)
print(energy)

```

```
-899.144763339983
```

```

#Obtain total energy

energy_unit=simtk.openmm.unit.kilojoule_per_mole
state = simulation.context.getState(getEnergy=True)
energy = state.getPotentialEnergy().value_in_unit(energy_unit)
print('TotalEnergy',round(energy,6),energy_unit.get_symbol())

#Obtain detailed energy

energies = {}
for force_name, force in forces.items():
    group=force.getForceGroup()
    state = simulation.context.getState(getEnergy=True, groups=2**group)
    energies[force_name] =state.getPotentialEnergy().value_in_unit(energy_unit)

for force_name in forces.keys():

```

(continues on next page)

(continued from previous page)

```
print(force_name, round(energies[force_name],6),energy_unit.get_symbol())
```

```
TotalEnergy -899.147583 kJ/mol
Bond 327.558105 kJ/mol
Angle 973.859009 kJ/mol
Stacking 203.565979 kJ/mol
Dihedral -385.277161 kJ/mol
BasePair -284.232208 kJ/mol
CrossStacking -47.586143 kJ/mol
Exclusion 23.991552 kJ/mol
Electrostatics 23.268274 kJ/mol
ExclusionProteinDNA 296.033478 kJ/mol
ElectrostaticsProteinDNA -10.459805 kJ/mol
Connectivity 1899.296875 kJ/mol
Chain 1899.296875 kJ/mol
Chi 1899.296875 kJ/mol
Excl 1899.296875 kJ/mol
rama -1363.522705 kJ/mol
rama_pro -1363.522705 kJ/mol
contact -1041.547607 kJ/mol
frag -1213.29834 kJ/mol
beta1 -300.796692 kJ/mol
beta2 -300.796692 kJ/mol
beta3 -300.796692 kJ/mol
pap1 0.0 kJ/mol
pap2 0.0 kJ/mol
```

```
#Add simulation reporters
dcd_reporter=simtk.openmm.app.DCDReporter(f'output.dcd', 10000)
energy_reporter=simtk.openmm.app.StateDataReporter(sys.stdout, 10000, step=True,
↳time=True,
potentialEnergy=True,↳
↳temperature=True)
simulation.reporters.append(dcd_reporter)
simulation.reporters.append(energy_reporter)
```

```
#Run simulation
simulation.minimizeEnergy()
simulation.context.setVelocitiesToTemperature(temperature)
simulation.step(100000)
```

```
#"Step","Time (ps)","Potential Energy (kJ/mole)","Temperature (K)"
1000,2.000000000000000013,-3507.7216796875,291.6758662031427
2000,3.999999999999999781,-3233.395751953125,299.73657773689456
3000,5.999999999999999561,-3472.61083984375,314.04882500167656
4000,7.999999999999999341,-3199.33251953125,309.8776608226663
5000,10.0000000000000009,-3275.126220703125,321.27214200690065
6000,12.000000000000000677,-3167.643798828125,295.1803751740272
7000,14.0000000000000001345,-3187.2998046875,310.9197062404284
8000,16.000000000000000201,-3296.966064453125,305.38198987465074
9000,18.000000000000000902,-3182.54443359375,316.07187422798313
```

(continues on next page)

(continued from previous page)

```
10000,19.999999999999794,-3229.941650390625,309.6002450725328
```

```
#Get the detailed energy after the simulation
energies = {}
for force_name, force in forces.items():
    group=force.getForceGroup()
    state = simulation.context.getState(getEnergy=True, groups=2**group)
    energies[force_name] =state.getPotentialEnergy().value_in_unit(energy_unit)

for force_name in forces.keys():
    print(force_name, round(energies[force_name],6),energy_unit.get_symbol())
```

```
Bond 102.787193 kJ/mol
Angle 169.343231 kJ/mol
Stacking -411.438232 kJ/mol
Dihedral -452.744202 kJ/mol
BasePair -242.723328 kJ/mol
CrossStacking -48.275833 kJ/mol
Exclusion 1.823019 kJ/mol
Electrostatics 23.37781 kJ/mol
ExclusionProteinDNA -6.992864 kJ/mol
ElectrostaticsProteinDNA -10.225787 kJ/mol
Connectivity 1487.126465 kJ/mol
Chain 1487.126587 kJ/mol
Chi 1487.126465 kJ/mol
Excl 1487.126587 kJ/mol
rama -1456.633789 kJ/mol
rama_pro -1456.633789 kJ/mol
contact -1326.107178 kJ/mol
frag -922.991699 kJ/mol
beta1 -136.266449 kJ/mol
beta2 -136.266449 kJ/mol
beta3 -136.266449 kJ/mol
pap1 0.0 kJ/mol
pap2 0.0 kJ/mol
```


CLASSES AND FUNCTIONS

3.1 DNA

class open3SPN2.DNA(*periodic=True*)

A Coarse Grained DNA object.

parseConfigurationFile(*configuration_file='/home/docs/checkouts/readthedocs.org/user_builds/open3spn2/checkouts/latest'*)

Reads the configuration file for the forcefield. The default configuration file is 3SPN2.conf and it contains most of the parameters used in the simulation.

getSequences()

Returns the DNA sequence as a Pandas Series. The index of the Series is (Chain, resid)

computeGeometry(*sequence=None, temp_name='temp'*)

This function requires X3DNA. It returns a pdb table containing the expected DNA structure

computeTopology(*template_from_X3DNA=True, temp_name='temp'*)

Creates tables of bonds, angles and dihedrals with their respective parameters (bonded interactions). 3SPN2.C requires a template structure to calculate the equilibrium bonds, angles and dihedrals. If *template_from_structure* is True, it will try to compute the equilibrium geometry using X3DNA. If *template_from_structure* is False, then the initial structure is expected to be the equilibrium geometry

writePDB(*pdb_file='clean.pdb'*)

Writes a minimal version of the pdb file needed for openmm

classmethod fromCoarsePDB(*pdb_file, dna_type='B_curved', template_from_X3DNA=True, temp_name='temp', compute_topology=True*)

Initializes a DNA object from a pdb file containing the Coarse Grained atoms

classmethod fromPDB(*pdb_file, dna_type='B_curved', template_from_X3DNA=True, output_pdb='clean.pdb', temp_name='temp', compute_topology=True*)

Creates a DNA object from a complete(atomistic) pdb file

static CoarseGrain(*pdb_table*)

Selects DNA atoms from a pdb table and returns a table containing only the coarse-grained atoms for 3SPN2

classmethod fromSequence(*sequence, dna_type='B_curved', output_pdb='clean.pdb', temp_name='temp', compute_topology=True*)

Initializes a DNA object from a DNA sequence

classmethod fromXYZ(*xyz_file, dna_type='B_curved', template_from_X3DNA=True, output_pdb='clean.pdb', temp_name='temp', compute_topology=True*)

Initializes DNA object from xyz file (as seen on the examples)

3.2 System

```
class open3SPN2.System(dna, forcefield-  
                        Files=['/home/docs/checkouts/readthedocs.org/user_builds/open3spn2/checkouts/latest/open3SPN2/3SPN2'],  
                        periodicBox=None)  
    Wrapper of openmm system class, adds some openmm simulation attributes  
  
    __init__(self) → System  
    __init__(self, other) → System  
        Create a new System.
```

Methods

3.3 DNA Forces

```
class open3SPN2.Bond(dna, force_group=6, OpenCLPatch=True)  
class open3SPN2.Angle(dna, force_group=7, OpenCLPatch=True)  
class open3SPN2.Dihedral(dna, force_group=9, OpenCLPatch=True)  
class open3SPN2.Stacking(dna, force_group=8, OpenCLPatch=True)  
class open3SPN2.BasePair(dna, force_group=10, OpenCLPatch=True)  
class open3SPN2.CrossStacking(dna, force_group=11, OpenCLPatch=True)  
class open3SPN2.Exclusion(dna, force_group=12, OpenCLPatch=True)  
class open3SPN2.Electrostatics(dna, force_group=13, temperature=Quantity(value=300, unit=kelvin),  
                                salt_concentration=Quantity(value=100, unit=millimolar),  
                                OpenCLPatch=True)
```

3.4 DNA - Protein Forces

```
class open3SPN2.ExclusionProteinDNA(dna, protein, k=1, force_group=14)  
    Protein-DNA exclusion potential  
  
class open3SPN2.ElectrostaticsProteinDNA(dna, protein, k=1, force_group=15)  
    DNA-protein and protein-protein electrostatics.
```

3.5 Utils

```
open3SPN2.parseConfigTable(config_section)  
    Parses a section of the configuration file as a table. This function is used to parse the 3SPN2.conf file  
  
open3SPN2.parsePDB(pdb_file)  
    Transforms the pdb file into a pandas table for easy access and data editing  
  
open3SPN2.fixPDB(pdb_file)  
    Uses the pdbfixer library to fix a pdb file, replacing non standard residues, removing hetero-atoms and adding  
    missing hydrogens. The input is a pdb file location, the output is a fixer object, which is a pdb in the openawsem  
    format.
```

`open3SPN2.pdb2table(pdb)`

Parses a pdb in the openmm format and outputs a table that contains all the information on a pdb file

3.6 Tests

class `open3SPN2.TestEnergies`

Tests that the energies are the same as the example outputs from lammmps

`open3SPN2.test_DNA_from_xyz()`

Tests the correct parsing from an xyz file

`open3SPN2.test_parse_xyz()`

Tests the example trajectory parsing

`open3SPN2.test_parse_log()`

Tests the example log parsing

3.7 Exceptions

class `open3SPN2.DNATypeError(dna)`

Only some DNA types are defined (A, B, B_curved)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

Symbols

`__init__()` (*open3SPN2.System method*), 16

A

`Angle` (*class in open3SPN2*), 16

B

`BasePair` (*class in open3SPN2*), 16

`Bond` (*class in open3SPN2*), 16

C

`CoarseGrain()` (*open3SPN2.DNA static method*), 15

`computeGeometry()` (*open3SPN2.DNA method*), 15

`computeTopology()` (*open3SPN2.DNA method*), 15

`CrossStacking` (*class in open3SPN2*), 16

D

`Dihedral` (*class in open3SPN2*), 16

`DNA` (*class in open3SPN2*), 15

`DNATypeError` (*class in open3SPN2*), 17

E

`Electrostatics` (*class in open3SPN2*), 16

`ElectrostaticsProteinDNA` (*class in open3SPN2*), 16

`Exclusion` (*class in open3SPN2*), 16

`ExclusionProteinDNA` (*class in open3SPN2*), 16

F

`Feynman`, 5

`fixPDB()` (*in module open3SPN2*), 16

`fromCoarsePDB()` (*open3SPN2.DNA class method*), 15

`fromPDB()` (*open3SPN2.DNA class method*), 15

`fromSequence()` (*open3SPN2.DNA class method*), 15

`fromXYZ()` (*open3SPN2.DNA class method*), 15

G

`getSequences()` (*open3SPN2.DNA method*), 15

P

`parseConfigTable()` (*in module open3SPN2*), 16

`parseConfigurationFile()` (*open3SPN2.DNA method*), 15

`parsePDB()` (*in module open3SPN2*), 16

`pdb2table()` (*in module open3SPN2*), 16

S

`Stacking` (*class in open3SPN2*), 16

`System` (*class in open3SPN2*), 16

T

`test_DNA_from_xyz()` (*in module open3SPN2*), 17

`test_parse_log()` (*in module open3SPN2*), 17

`test_parse_xyz()` (*in module open3SPN2*), 17

`TestEnergies` (*class in open3SPN2*), 17

W

`writePDB()` (*open3SPN2.DNA method*), 15