NC STATE

Jiaze He <jhe7@ncsu.edu>

Your comparison from Text Compare!

no-reply@text-compare.com <no-reply@text-compare.com>
To: jhe7@ncsu.edu

Wed, Jul 4, 2018 at 7:37 PM

```
1
2  import sys
3  from os.path import basename, join
4  from glob import glob
5
6  import numpy as np
7
8  from seisflows.plugins.solver.specfem2d import smooth_legacy
9  from seisflows.tools.seismic import getpar, setpar
10
11 from seisflows.tools import msg
12 from seisflows.tools import unix
13 from seisflows.tools.seismic import call_solver
14 from seisflows.tools.tools import exists
15 from seisflows.config import ParameterError, custom_import
16
17 PAR = sys.modules['seisflows_parameters']
18 PATH = sys.modules['seisflows_paths']
19
20 system = sys.modules['seisflows_system']
21 preprocess = sys.modules['seisflows_preprocess']
22
23
24 class specfem2d_new(custom_import('solver', 'base')):
25    """ Python interface for SPECFEM2D
26
27      See base class for method descriptions
28    """
29    if PAR.MATERIALS == 'LegacyAcoustic':
30        parameters = []
31        parameters += ['vs']
32
33
34    def check(self):
35        """ Checks parameters and paths
36        """
37        super(specfem2d_new, self).check()
38
39        # check time stepping parameters
40        if 'NT' not in PAR:
41            raise Exception
42
43        if 'DT' not in PAR:
44            raise Exception
45
46        if 'F0' not in PAR:
47            raise Exception
48
49        # check data format
50        if 'FORMAT' not in PAR:
51            raise Exception()
52
53        if PAR.FORMAT != 'su':
54            raise Exception()
```

```
1
2  import sys
3  from os.path import basename, join
4  from glob import glob
5
6  import numpy as np
7
8  from seisflows.plugins.solver.specfem2d import smooth_legacy
9  from seisflows.tools.seismic import getpar, setpar
10
11 from seisflows.tools import msg
12 from seisflows.tools import unix
13 from seisflows.tools.seismic import call_solver
14 from seisflows.tools.tools import exists
15 from seisflows.config import ParameterError, custom_import
16
17 PAR = sys.modules['seisflows_parameters']
18 PATH = sys.modules['seisflows_paths']
19
20 system = sys.modules['seisflows_system']
21 preprocess = sys.modules['seisflows_preprocess']
22
23
24 class specfem2d_new(custom_import('solver', 'base')):
25    """ Python interface for SPECFEM2D
26
27      See base class for method descriptions
28    """
29    if PAR.MATERIALS == 'LegacyAcoustic':
30        parameters = []
31        parameters += ['vs']
32
33
34    def check(self):
35        """ Checks parameters and paths
36        """
37        super(specfem2d_new, self).check()
38
39        # check time stepping parameters
40        if 'NT' not in PAR:
41            raise Exception
42
43        if 'DT' not in PAR:
44            raise Exception
45
46        if 'F0' not in PAR:
47            raise Exception
48
49        # check data format
50        if 'FORMAT' not in PAR:
51            raise Exception()
52
53        if PAR.FORMAT != 'su':
54            raise Exception()
```

```
55
56
57  def check_solver_parameter_files(self):
58      """ Checks solver parameters
59      """
60      nt = getpar('NSTEP', cast=int)
61      dt = getpar('DT', cast=float)
62      f0 = getpar('f0', file='DATA/SOURCE', cast=float)
63
64      if nt != PAR.NT:
65          if self.taskid == 0: print "WARNING: nt != PAR.NT"
66          setpar('nt', PAR.NT)
67
68      if dt != PAR.DT:
69          if self.taskid == 0: print "WARNING: dt != PAR.DT"
70          setpar('deltat', PAR.DT)
71
72      if f0 != PAR.F0:
73          if self.taskid == 0: print "WARNING: f0 != PAR.F0"
74          setpar('f0', PAR.F0, filename='DATA/SOURCE')
75
76      if self.mesh_properties.nproc != PAR.NPROC:
77          if self.taskid == 0:
78              print 'Warning: mesh_properties.nproc != PAR.NPROC'
79
80      if 'MULTIPLES' in PAR:
81          if PAR.MULTIPLES:
82              setpar('absorbtop', '.false.')
83          else:
84              setpar('absorbtop', '.true.')
85
86
87  def generate_data(self, **model_kwargs):
88      """ Generates data
89      """
90      self.generate_mesh(**model_kwargs)
91
92      unix.cd(self.cwd)
93      setpar('SIMULATION_TYPE', '1')
94      setpar('SAVE_FORWARD', '.false.')
95
96      call_solver(system.mpiexec(), 'bin/xmeshfem2D')
97      call_solver(system.mpiexec(), 'bin/xspecfem2D')
98
99      if PAR.FORMAT in ['SU', 'su']:
100         src = glob('OUTPUT_FILES/*.su')
101         dst = 'traces/obs'
102         unix.mv(src, dst)
103
104     if PAR.SAVETRACES:
105         self.export_traces(PATH.OUTPUT+'/'+'traces/obs')
106
107
108 def initialize_adjoint_traces(self):
109     super(specfem2d_new, self).initialize_adjoint_traces()
110
111     # work around SPECFEM2D's use of different name conventions for
112     # regular traces and 'adjoint' traces
113     if PAR.FORMAT in ['SU', 'su']:
114         files = glob('traces/adj/*.su')
115         unix.rename('.su', '.su.adj', files)
116
117     # work around SPECFEM2D's requirement that all components exist,
118     # even ones not in use
119     if PAR.FORMAT in ['SU', 'su']:
```

```
55
56
57  def check_solver_parameter_files(self):
58      """ Checks solver parameters
59      """
60      nt = getpar('NSTEP', cast=int)
61      dt = getpar('DT', cast=float)
62      f0 = getpar('f0', file='DATA/SOURCE', cast=float)
63
64      if nt != PAR.NT:
65          if self.taskid == 0: print "WARNING: nt != PAR.NT"
66          setpar('nt', PAR.NT)
67
68      if dt != PAR.DT:
69          if self.taskid == 0: print "WARNING: dt != PAR.DT"
70          setpar('deltat', PAR.DT)
71
72      if f0 != PAR.F0:
73          if self.taskid == 0: print "WARNING: f0 != PAR.F0"
74          setpar('f0', PAR.F0, filename='DATA/SOURCE')
75
76      if self.mesh_properties.nproc != PAR.NPROC:
77          if self.taskid == 0:
78              print 'Warning: mesh_properties.nproc != PAR.NPROC'
79
80      if 'MULTIPLES' in PAR:
81          if PAR.MULTIPLES:
82              setpar('absorbtop', '.false.')
83          else:
84              setpar('absorbtop', '.true.')
85
86
87  def generate_data(self, **model_kwargs):
88      """ Generates data
89      """
90      self.generate_mesh(**model_kwargs)
91
92      unix.cd(self.cwd)
93      setpar('SIMULATION_TYPE', '1')
94      setpar('SAVE_FORWARD', '.false.')
95
96      call_solver(system.mpiexec(), 'bin/xmeshfem2D',output='mesh.log')
97      call_solver(system.mpiexec(), 'bin/xspecfem2D')
98
99      if PAR.FORMAT in ['SU', 'su']:
100         src = glob('OUTPUT_FILES/*.su')
101         dst = 'traces/obs'
102         unix.mv(src, dst)
103
104     if PAR.SAVETRACES:
105         self.export_traces(PATH.OUTPUT+'/'+'traces/obs')
106
107
108 def initialize_adjoint_traces(self):
109     super(specfem2d_new, self).initialize_adjoint_traces()
110
111     # work around SPECFEM2D's use of different name conventions for
112     # regular traces and 'adjoint' traces
113     if PAR.FORMAT in ['SU', 'su']:
114         files = glob('traces/adj*/*.su')
115         unix.rename('.su', '.su.adj', files)
116
117     # work around SPECFEM2D's requirement that all components exist,
118     # even ones not in use
119     if PAR.FORMAT in ['SU', 'su']:
```

```
            unix.cd(self.cwd +'/'+ 'traces/adj')
            for channel in ['x', 'y', 'z', 'p']:
                src = 'U%s_file_single.su.adj' % PAR.CHANNELS[0]
                dst = 'U%s_file_single.su.adj' % channel
                if not exists(dst):
                    unix.cp(src, dst)


    def generate_mesh(self, model_path=None, model_name=None, model_type='gll'):
        """ Performs meshing and database generation
        """
        assert(model_name)
        assert(model_type)

        self.initialize_solver_directories()
        unix.cd(self.cwd)

        assert(exists(model_path))
        self.check_mesh_properties(model_path)

        src = glob(join(model_path, '*'))
        dst = join(self.cwd, 'DATA')
        unix.cp(src, dst)

        if self.taskid == 0:
            self.export_model(PATH.OUTPUT +'/'+ model_name)


    ### low-level solver interface

    def forward(self, path='traces/syn'):
        """ Calls SPECFEM2D forward solver
        """
        setpar('SIMULATION_TYPE', '1')
        setpar('SAVE_FORWARD', '.true.')
#         call_solver(system.mpiexec(), 'bin/xmeshfem2D')

        call_solver(system.mpiexec(), 'bin/xspecfem2D')

        if PAR.FORMAT in ['SU', 'su']:
            filenames = glob('OUTPUT_FILES/*.su')
            unix.mv(filenames, path)


    def adjoint(self):
        """ Calls SPECFEM2D adjoint solver
        """
        setpar('SIMULATION_TYPE', '3')
        setpar('SAVE_FORWARD', '.false.')
        unix.rm('SEM')
        unix.ln('traces/adj', 'SEM')

        # hack to deal with different SPECFEM2D name conventions for
        # regular traces and 'adjoint' traces
        if PAR.FORMAT in ['SU', 'su']:
            files = glob('traces/adj/*.su')
            unix.rename('.su', '.su.adj', files)

        call_solver(system.mpiexec(), 'bin/xspecfem2D')
```

```
            unix.cd(self.cwd +'/'+ 'traces/adj')
            for channel in ['x', 'y', 'z', 'p']:
                src = 'U%s_file_single.su.adj' % PAR.CHANNELS[0]
                dst = 'U%s_file_single.su.adj' % channel
                if not exists(dst):
                    unix.cp(src, dst)


    def generate_mesh(self, model_path=None, model_name=None, model_type='gll'):
        """ Performs meshing and database generation
        """
        assert(model_name)
        assert(model_type)

        self.initialize_solver_directories()
        unix.cd(self.cwd)

        assert(exists(model_path))
        self.check_mesh_properties(model_path)

        src = glob(join(model_path, '*'))
        dst = join(self.cwd, 'DATA')
        unix.cp(src, dst)

        if self.taskid == 0:
            self.export_model(PATH.OUTPUT +'/'+ model_name)


    ### low-level solver interface

    def forward(self, path='traces/syn'):
        """ Calls SPECFEM2D forward solver
        """
        setpar('SIMULATION_TYPE', '1')
        setpar('SAVE_FORWARD', '.true.')
#         call_solver(system.mpiexec(), 'bin/xmeshfem2D')

        call_solver(system.mpiexec(), 'bin/xspecfem2D')

        if PAR.FORMAT in ['SU', 'su']:
            filenames = glob('OUTPUT_FILES/*.su')
            unix.mv(filenames, path)


    def adjoint(self):
        """ Calls SPECFEM2D adjoint solver
        """
        setpar('SIMULATION_TYPE', '3')
        setpar('SAVE_FORWARD', '.false.')
        unix.rm('SEM')
        unix.ln('traces/adj', 'SEM')

        # hack to deal with different SPECFEM2D name conventions for
        # regular traces and 'adjoint' traces
        if PAR.FORMAT in ['SU', 'su']:
            files = glob('traces/adj/*.su')
            unix.rename('.su', '.su.adj', files)

        call_solver(system.mpiexec(), 'bin/xspecfem2D')

    def adjoint_att(self):
        """ Calls SPECFEM2D adjoint solver
        """
        unix.rm('SEM')
        unix.ln('traces/adj_att', 'SEM')
```

```
    def rename_kernels(self):
        """ Works around conflicting kernel filename conventions
        """
        files = []
        files += glob('*proc??????_alpha_acoustic_kernel.bin')
        unix.rename('alpha_acoustic', 'vp', files)
        files = []
        files += glob('*proc??????_alpha[hv]_kernel.bin')
        files += glob('*proc??????_reg1_alpha_kernel.bin')
        files += glob('*proc??????_reg1_alpha[hv]_kernel.bin')
        unix.rename('alpha', 'vp', files)
        files += glob('*proc??????_c_acoustic_kernel.bin')
        unix.rename('c_acoustic', 'vp', files)
     #   unix.rename('c_acoustic', 'Qkappa', files)
        files = []
        files += glob('*proc??????_rhop_acoustic_kernel.bin')
        unix.rename('rhop_acoustic', 'rho', files)

        files = []
        files += glob('*proc??????_beta_kernel.bin')
        files += glob('*proc??????_beta[hv]_kernel.bin')
        files += glob('*proc??????_reg1_beta_kernel.bin')
        files += glob('*proc??????_reg1_beta[hv]_kernel.bin')
        unix.rename('beta', 'vs', files)


    def initialize_solver_directories(self):
        """ Creates directory structure expected by SPECFEM3D, copies
          executables, and prepares input files. Executables must be supplied
          by user as there is currently no mechanism for automatically
          compiling from source.
        """
        unix.mkdir(self.cwd)
        unix.cd(self.cwd)

        # create directory structure
        unix.mkdir('bin')
        unix.mkdir('DATA')
        unix.mkdir('OUTPUT_FILES')

        unix.mkdir('traces/obs')
        unix.mkdir('traces/syn')
```

```

        # hack to deal with different SPECFEM2D name conventions for
        # regular traces and 'adjoint' traces
        if PAR.FORMAT in ['SU', 'su']:
            files = glob('traces/adj_att/*.su')
            unix.rename('.su', '.su.adj', files)

        call_solver(system.mpiexec(), 'bin/xspecfem2D')


    def rename_kernels(self):
        """ Works around conflicting kernel filename conventions
        """
        files = []
        files += glob('*proc??????_alpha_acoustic_kernel.bin')
        unix.rename('alpha_acoustic', 'vp', files)
        files = []
        files += glob('*proc??????_alpha[hv]_kernel.bin')
        files += glob('*proc??????_reg1_alpha_kernel.bin')
        files += glob('*proc??????_reg1_alpha[hv]_kernel.bin')
        unix.rename('alpha', 'vp', files)
        files += glob('*proc??????_c_acoustic_kernel.bin')
        unix.rename('c_acoustic', 'vp', files)
     #   unix.rename('c_acoustic', 'Qkappa', files)
        files = []
        files += glob('*proc??????_rhop_acoustic_kernel.bin')
        unix.rename('rhop_acoustic', 'rho', files)

        files = []
        files += glob('*proc??????_beta_kernel.bin')
        files += glob('*proc??????_beta[hv]_kernel.bin')
        files += glob('*proc??????_reg1_beta_kernel.bin')
        files += glob('*proc??????_reg1_beta[hv]_kernel.bin')
        unix.rename('beta', 'vs', files)

    def export_att_kernel(self, path):
        unix.cd(self.kernel_databases)

        # work around conflicting name conventions
        files = []
        files += glob('*proc??????_c_acoustic_kernel.bin')
        unix.rename('c_acoustic', 'Qkappa', files)

        src = glob('*Qkappa_kernel.bin')
        dst = join(path, 'kernels', self.source_name)
        unix.mkdir(dst)
        unix.mv(src, dst)


    def initialize_solver_directories(self):
        """ Creates directory structure expected by SPECFEM3D, copies
          executables, and prepares input files. Executables must be supplied
          by user as there is currently no mechanism for automatically
          compiling from source.
        """
        unix.mkdir(self.cwd)
        unix.cd(self.cwd)

        # create directory structure
        unix.mkdir('bin')
        unix.mkdir('DATA')
        unix.mkdir('OUTPUT_FILES')

        unix.mkdir('traces/obs')
        unix.mkdir('traces/syn')
```

```
221         unix.mkdir('traces/adj')
222
223         unix.mkdir(self.model_databases)
224         unix.mkdir(self.kernel_databases)
225
226         # copy exectuables
227         src = glob(PATH.SPECFEM_BIN +'/'+ '*')
228         dst = 'bin/'
229         unix.cp(src, dst)
230
231         # copy input files
232         src = glob(PATH.SPECFEM_DATA +'/'+ '*')
233         dst = 'DATA/'
234         unix.cp(src, dst)
235
236         src = 'DATA/' + self.source_prefix +'_'+ self.source_name
237         dst = 'DATA/' + self.source_prefix
238         unix.cp(src, dst)
239
240         src = 'DATA/STATIONS'+'_'+ self.source_name
241         dst = 'DATA/STATIONS'
242         unix.cp(src, dst)
243
244         self.check_solver_parameter_files()
245
246
247
248     ### file transfer utilities
249
250     def import_model(self, path):
251         src = glob(path +'/'+ 'model/*')
252         dst = join(self.cwd, 'DATA/')
253         unix.cp(src, dst)
254
255     def export_model(self, path):
256         unix.mkdir(path)
257         src = glob(join(self.cwd, 'DATA/*.bin'))
258         dst = path
259         unix.cp(src, dst)
260
261
262     @property
263     def data_filenames(self):
264         if PAR.CHANNELS:
265             if PAR.FORMAT in ['SU', 'su']:
266                filenames = []
267                for channel in PAR.CHANNELS:
268                    filenames += ['U%s_file_single.su' % channel]
269                return filenames
270
271         else:
272             unix.cd(self.cwd)
273             unix.cd('traces/obs')
274
275             if PAR.FORMAT in ['SU', 'su']:
276                 return glob('U?_file_single.su')
277
278     @property
279     def model_databases(self):
280         return join(self.cwd, 'DATA')
281
282     @property
283     def kernel_databases(self):
```

```
250         unix.mkdir('traces/adj')
251         if PAR.ATTENUATION == 'yes' :
252           unix.mkdir('traces/adj_att')
253
254         unix.mkdir(self.model_databases)
255         unix.mkdir(self.kernel_databases)
256
257         # copy exectuables
258         src = glob(PATH.SPECFEM_BIN +'/'+ '*')
259         dst = 'bin/'
260         unix.cp(src, dst)
261
262         # copy input files
263         src = glob(PATH.SPECFEM_DATA +'/'+ '*')
264         dst = 'DATA/'
265         unix.cp(src, dst)
266
267         src = 'DATA/' + self.source_prefix +'_'+ self.source_name
268         dst = 'DATA/' + self.source_prefix
269         unix.cp(src, dst)
270
271         src = 'DATA/STATIONS'+'_'+ self.source_name
272         dst = 'DATA/STATIONS'
273         unix.cp(src, dst)
274
275         self.check_solver_parameter_files()
276
277
278
279     ### file transfer utilities
280
281     def import_model(self, path):
282         src = glob(path +'/'+ 'model/*')
283         dst = join(self.cwd, 'DATA/')
284         unix.cp(src, dst)
285
286     def export_model(self, path):
287         unix.mkdir(path)
288         src = glob(join(self.cwd, 'DATA/*.bin'))
289         dst = path
290         unix.cp(src, dst)
291
292
293     @property
294     def data_filenames(self):
295         if PAR.CHANNELS:
296             if PAR.FORMAT in ['SU', 'su']:
297                filenames = []
298                for channel in PAR.CHANNELS:
299                    filenames += ['U%s_file_single.su' % channel]
300                return filenames
301
302         else:
303             unix.cd(self.cwd)
304             unix.cd('traces/obs')
305
306             if PAR.FORMAT in ['SU', 'su']:
307                 return glob('U?_file_single.su')
308
309     @property
310     def model_databases(self):
311         return join(self.cwd, 'DATA')
312
313     @property
314     def kernel_databases(self):
```

```
284         return join(self.cwd, 'OUTPUT_FILES')
285
286     @property
287     def source_prefix(self):
288         return 'SOURCE'
289
290     # workaround for older versions of SPECFEM2D,
291     # which lacked a smoothing utility
292     #if not exists(PATH.SPECFEM_BIN+'/'+'xsmooth_sem'):
293     #    smooth = staticmethod(smooth_legacy)
294     #smooth = staticmethod(smooth_legacy)
295
```

```
315         return join(self.cwd, 'OUTPUT_FILES')
316
317     @property
318     def source_prefix(self):
319         return 'SOURCE'
320
321     # workaround for older versions of SPECFEM2D,
322     # which lacked a smoothing utility
323     #if not exists(PATH.SPECFEM_BIN+'/'+'xsmooth_sem'):
324     #    smooth = staticmethod(smooth_legacy)
325     #smooth = staticmethod(smooth_legacy)
326
```