```python
"""
Created on Mon Mar 010 16:28:20 2025
EPO-RSAC Algorithm
@author: Bingbing
"""
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
os.environ["CUDA_VISIBLE_DEVICES"] = '0'
import tensorflow._api.v2.compat.v1 as tf
tf.reset_default_graph()
tf.disable_v2_behavior()
import numpy as np
import pandas as pd
import time
import datetime
import seaborn as sns
from logx import EpochLogger
from mpi_tf import sync_all_params, MpiAdamOptimizer
from mpi_tools import mpi_fork, mpi_sum, proc_id, mpi_statistics_scalar, num_procs
from scipy.stats import norm
import globalvar as gl
import json
import argparse
from run_utils import setup_logger_kwargs
import matplotlib.pyplot as plt
from env3 import unPower
n_T = gl.get_value('n_T')
n_wind = gl.get_value('n_wind')
n_gen = gl.get_value('n_gen')
n_E = gl.get_value('n_E')
n_wind_data = gl.get_value('n_wind_data')
obs_dim = gl.get_value('state_dim')
act_dim = gl.get_value('action_dim')
P_load = gl.get_value('P_load')
P_wind = gl.get_value('P_wind')
load_capacity = gl.get_value('load_capacity')
wind_capacity = gl.get_value('wind_capacity')
pg_max = gl.get_value('pg_max')
pg_min = gl.get_value('pg_min')
ROCOFmax = gl.get_value('ROCOFmax')
delt_fmax = gl.get_value('delt_fmax')
EPS = 1e-8

snow = datetime.datetime.now()
```

```python
45.  seed_now = str(snow.month)+str(snow.day)+str(snow.hour)
46.  seed_now = int(seed_now)
47.
48.  #==============================================configuration parameter===
     ==============================================
49.  parser = argparse.ArgumentParser()
50.  parser.add_argument('--Train', type=bool, default=True, help='True or False')
51.  parser.add_argument('--
     Continue', type=bool, default=False, help='Whether to continue training the previous model')
52.  parser.add_argument('--Portion', type=bool, default=False, help='Whether to load only part of the model')
53.  parser.add_argument('--
     SaveFig', type=bool, default=True, help='Whether to download the debugged image to    path')
54.  parser.add_argument('--
     load_buffer', type=bool, default=False, help='Whether to load the replay buffer for path')
55.  parser.add_argument('--Model_version', type=int, default=10000, help='2000,Load model version')
56.  parser.add_argument('--epochs', type=int, default=10001, help='2001')
57.  parser.add_argument('--steps_per_epoch', type=int, default=3, help='n_wind_data*2+1')
58.  parser.add_argument('--
     update_freq', type=int, default=60, help='120,Need to be less than steps_per_epoch*24')
59.  parser.add_argument('--save_freq', default=2000, type=int, help='500')
60.  parser.add_argument('--cost_lim', type=float, default=0.1, help='10')
61.  parser.add_argument('--cl', type=float, default=0.95, help='0.5')
62.  parser.add_argument('--lr', type=float, default=1e-3,help='1e-
     3,If Continue, there should be a smaller lr tweak')
63.
64.  parser.add_argument('--
     hidden_sizes_actor', type=list, default=[64, 256, 256, 64], help='FCs of policy network actor')
65.  parser.add_argument('--
     hidden_sizes_critic', type=list, default=[64, 256, 256, 64], help='FCs of network critic')
66.  parser.add_argument('--
     hidden_sizes_var', type=list, default=[64, 256, 256, 64], help='FCs of network var')
67.  parser.add_argument('--cnn_actor', type=list, default=[], help='CNNs of policy network actor')
68.  parser.add_argument('--cnn_critic', type=list, default=[], help='CNNs of    network critic')
69.  parser.add_argument('--cnn_var', type=list, default=[], help='CNNs of network var')
70.
71.  parser.add_argument('--
     hid', type=int, default=256, help='Number of neurons per layer of a neural network')
72.  parser.add_argument('--l', type=int, default=2, help='The number of layers of the neural network')
73.  parser.add_argument('--gamma', type=float, default=0.99)
74.  parser.add_argument('--seed', '-s', type=int, default=seed_now)
75.  parser.add_argument('--exp_name', type=str, default='sac')
76.  parser.add_argument('--cpu', type=int, default=2)
77.  parser.add_argument('--render', default=False, action='store_true')
78.  parser.add_argument('--local_start_steps', default=3600, type=int, help='当 Continue 时，最好设为 0')
```

```python
79.    parser.add_argument('--local_update_after', default=3600, type=int, help='表示从几代后开始优化，当
           Continue 时，最好设为 0')
80.    parser.add_argument('--batch_size', default=256, type=int)    # 256
81.    parser.add_argument('--fixed_entropy_bonus', default=None, type=float)
82.    parser.add_argument('--entropy_constraint', type=float, default=-1)
83.    parser.add_argument('--fixed_cost_penalty', default=None, type=float)
84.    parser.add_argument('--cost_constraint', type=float, default=None)
85.    parser.add_argument('--lr_s', type=int, default=50)
86.    parser.add_argument('--damp_s', type=int, default=10)
87.    parser.add_argument('--logger_kwargs_str', type=json.loads, default='{"output_dir": "./logger"}')
88.    args = parser.parse_args()
89.    #========================================================================
           ====================================================
90.
91.    def placeholder(dim=None):
92.        return tf.placeholder(dtype=tf.float32, shape=(None, dim) if dim else (None,))
93.
94.    def placeholders(*args):
95.        return [placeholder(dim) for dim in args]
96.
97.    def cnn_mlp(x, hidden_sizes=(64,), activation=tf.tanh, output_activation=None, cnn_sizes=[16, 32]):
98.        j = 1
99.        if cnn_sizes!=[]:
100.           dim = int(x.shape[1])
101.           Img_shape = (dim, 1)
102.           x = tf.reshape(x,(-1,*Img_shape))
103.           for i in cnn_sizes:
104.               x = tf.layers.conv1d(
105.                   inputs=x,
106.                   filters=i,
107.                   kernel_size=3,
108.                   padding='same',
109.                   use_bias=False,
110.                   activation=None,
111.                   name='cnn_%d'%(j))
112.               x = tf.layers.batch_normalization(
113.                   inputs=x,
114.                   training=True,
115.                   name='bn_%d'%(j))
116.               x = tf.tanh(x)
117.               j += 1
118.
119.           x = tf.reshape(x,(-1,dim*cnn_sizes[-1]))
120.        for h in hidden_sizes[:-1]:
```

```python
121.        x = tf.layers.dense(x, units=h, activation=activation, name='fc_%d'%(j))
122.        j += 1
123.    return tf.layers.dense(x, units=hidden_sizes[-1], activation=output_activation,name='fc_%d'%(j))
124.
125.
126. def get_vars(scope):
127.    return [x for x in tf.global_variables() if scope in x.name]
128.
129. def count_vars(scope):
130.    v = get_vars(scope)
131.    return sum([np.prod(var.shape.as_list()) for var in v])
132.
133. def gaussian_likelihood(x, mu, log_std):
134.    pre_sum = -0.5 * (((x - mu) / (tf.exp(log_std) + EPS)) ** 2 + 2 * log_std + np.log(2 * np.pi))
135.    return tf.reduce_sum(pre_sum, axis=1)
136.
137. def get_target_update(main_name, target_name, polyak):
138.    '''
139.    Get a tensorflow op to update target variables based on main variables
140.    '''
141.    main_vars = {x.name: x for x in get_vars(main_name)}
142.    targ_vars = {x.name: x for x in get_vars(target_name)}
143.    assign_ops = []
144.    for v_targ in targ_vars:
145.        assert v_targ.startswith(target_name), f'bad var name {v_targ} for {target_name}'
146.        v_main = v_targ.replace(target_name, main_name, 1)
147.        assert v_main in main_vars, f'missing var name {v_main}'
148.        assign_op = tf.assign(targ_vars[v_targ], polyak * targ_vars[v_targ] + (1 - polyak) * main_vars[v_ma
    in]) # Soft update
149.        assign_ops.append(assign_op)
150.    return tf.group(assign_ops)
151.
152.
153. """
154. Policies
155. """
156.
157. LOG_STD_MAX = 2
158. LOG_STD_MIN = -20
159.
160.
161. def mlp_gaussian_policy(x, a, hidden_sizes, activation, output_activation, cnn_sizes):
162.    act_dim = a.shape.as_list()[-1]
163.    net = cnn_mlp(x, list(hidden_sizes), activation, activation,list(cnn_sizes))
```

```python
164.    mu = tf.layers.dense(net, act_dim, activation=output_activation,name='fc_mu')
165.    log_std = tf.layers.dense(net, act_dim, activation=None,name='fc_std')
166.    log_std = tf.clip_by_value(log_std, LOG_STD_MIN, LOG_STD_MAX)
167.
168.    std = tf.exp(log_std)
169.    pi = mu + tf.random_normal(tf.shape(mu)) * std
170.
171.    logp_pi = gaussian_likelihood(pi, mu, log_std)
172.    return mu, pi, logp_pi
173.
174.
175. def apply_squashing_func(mu, pi, logp_pi):
176.    # Adjustment to log prob
177.    logp_pi -= tf.reduce_sum(2 * (np.log(2) - pi - tf.nn.softplus(-2 * pi)), axis=1)
178.
179.    # Squash those unbounded actions!
180.    mu = tf.tanh(mu)
181.    pi = tf.tanh(pi)
182.    return mu, pi, logp_pi
183.
184.
185. """
186. Actors and Critics
187. """
188.
189. # Actor network
190. def mlp_actor(x, a, name='pi', hidden_sizes=(64, 64), activation=tf.nn.relu,
191.         output_activation=None, policy=mlp_gaussian_policy, action_space=None, cnn_sizes=[8,16]):
192.    # policy
193.    with tf.variable_scope(name):
194.        mu, pi, logp_pi = policy(x, a, hidden_sizes, activation, output_activation,cnn_sizes)
195.        mu, pi, logp_pi = apply_squashing_func(mu, pi, logp_pi)
196.    return mu, pi, logp_pi
197.
198. # Critic-var network
199. def mlp_var(x, a, pi, name, hidden_sizes=(64, 64), activation=tf.nn.relu,
200.         output_activation=None, action_space=None, cnn_sizes=[]):
201.    fn_mlp = lambda x: tf.squeeze(cnn_mlp(x=x,
202.                         hidden_sizes=list(hidden_sizes) + [1],
203.                         activation=activation,
204.                         output_activation=None,
205.                         cnn_sizes=cnn_sizes),
206.                    axis=1)
207.    with tf.variable_scope(name):
```

```python
208.        var = fn_mlp(tf.concat([x, a], axis=-1))
209.        var = tf.nn.softplus(var)
210.
211.    with tf.variable_scope(name, reuse=True):
212.        var_pi = fn_mlp(tf.concat([x, pi], axis=-1))
213.        var_pi = tf.nn.softplus(var_pi)
214.
215.    return var, var_pi
216.
217. # Critic network
218. def mlp_critic(x, a, pi, name, hidden_sizes=(64, 64), activation=tf.nn.relu,
219.              output_activation=None, action_space=None,cnn_sizes=[]):
220.    fn_mlp = lambda x: tf.squeeze(cnn_mlp(x=x,
221.                        hidden_sizes=list(hidden_sizes) + [1],
222.                        activation=activation,
223.                        output_activation=None,
224.                        cnn_sizes=cnn_sizes),
225.                    axis=1)
226.    with tf.variable_scope(name):
227.        critic = fn_mlp(tf.concat([x, a], axis=-1))
228.
229.    with tf.variable_scope(name, reuse=True):
230.        critic_pi = fn_mlp(tf.concat([x, pi], axis=-1))
231.
232.    return critic, critic_pi
233.
234. def Fig(logger_Dict, item='EpCost', c=None, Title=None, SaveFig=True, tsplot=True):
235.    if tsplot:
236.        try:
237.            Epoch1 = np.array(logger_Dict['Epoch'])
238.            Epoch2 = np.array(logger_Dict['Epoch'])
239.            Epoch3 = np.array(logger_Dict['Epoch'])
240.            item1 = np.array(logger_Dict['Average'+item])
241.            item2 = np.array(logger_Dict['Min'+item])
242.            item3 = np.array(logger_Dict['Max'+item])
243.            data = np.concatenate((item1,item2,item3))
244.            epoch = np.concatenate((Epoch1,Epoch2,Epoch3))
245.            sns.set(style="whitegrid")    # darkgrid whitegrid ticks
246.            sns.lineplot(x=epoch, y=data, color=c, lw=0.8)
247.            plt.xlabel("epoch")
248.            plt.ylabel(item)
249.            if Title:
250.                plt.title(Title)
251.            else:
```

```python
252.            plt.title('The Expectations of '+item+' of Each Epoch')
253.        if SaveFig:
254.            now = datetime.datetime.now()
255.            savefname = './Debug_record/%d.%d.%d/Figture'%(now.year,now.month,now.day)
256.            os.makedirs(savefname,exist_ok=True)
257.            plt.savefig(savefname+'/%d-%d-%s.png'%(now.hour,now.minute,item),dpi=500)
258.        plt.show()
259.    except:
260.        print('在 logger_Dict 中无法找到与" '+item+' "的对应数据或保存路径出错')
261.    else:
262.        try:
263.            Epoch = logger_Dict['Epoch']
264.            item_name = logger_Dict[item]
265.            plt.plot(Epoch, item_name)
266.            plt.xlabel('Epoch')
267.            plt.ylabel(item)
268.            if Title:
269.                plt.title(Title)
270.            else:
271.                plt.title('The Expectations of '+item+' of Each Epoch')
272.            if SaveFig:
273.                now = datetime.datetime.now()
274.                savefname = './Debug_record/%d.%d.%d/Figture'%(now.year,now.month,now.day)
275.                os.makedirs(savefname,exist_ok=True)
276.                plt.savefig(savefname+'/%d-%d-%s.png'%(now.hour,now.minute,item),dpi=500)
277.            plt.show()
278.        except:
279.            print('在 logger_Dict 中无法找到与" '+item+' "的对应数据或保存路径出错')
280.
281.
282.
283. class ReplayBuffer:
284.    """
285.    A simple FIFO experience replay buffer for SAC agents.
286.    """
287.
288.    def __init__(self, obs_dim, act_dim, size, load_buffer=True, Path_xls='./logger/Buffer.xlsx'):
289.        self.obs1_buf = np.zeros([size, obs_dim], dtype=np.float32)
290.        self.obs2_buf = np.zeros([size, obs_dim], dtype=np.float32)
291.        self.acts_buf = np.zeros([size, act_dim], dtype=np.float32)
292.        self.rews_buf = np.zeros(size, dtype=np.float32)
293.        self.costs_buf = np.zeros(size, dtype=np.float32)
294.        self.done_buf = np.zeros(size, dtype=np.float32)
295.        self.ptr, self.size, self.max_size = 0, 0, size
```

```python
296.
297.     if load_buffer:    # 加载已有经验池
298.         try:
299.             obs1_buf = np.array(pd.read_excel(Path_xls,sheet_name='obs1'))
300.             obs2_buf = np.array(pd.read_excel(Path_xls,sheet_name='obs2'))
301.             acts_buf = np.array(pd.read_excel(Path_xls,sheet_name='acts'))
302.             rews_buf = np.array(pd.read_excel(Path_xls,sheet_name='rews'))
303.             costs_buf = np.array(pd.read_excel(Path_xls,sheet_name='costs'))
304.             done_buf = np.array(pd.read_excel(Path_xls,sheet_name='done'))
305.
306.             u = np.min([size,rews_buf.shape[0]])
307.             self.obs1_buf[0:u,:] = obs1_buf[0:u,:]
308.             self.obs2_buf[0:u,:] = obs2_buf[0:u,:]
309.             self.acts_buf[0:u,:] = acts_buf[0:u,:]
310.             self.rews_buf[0:u] = rews_buf[0:u].reshape(-1)
311.             self.costs_buf[0:u] = costs_buf[0:u].reshape(-1)
312.             self.done_buf[0:u] = done_buf[0:u].reshape(-1)
313.
314.             for init_ptr in range(u):
315.                 if obs1_buf[init_ptr,obs_dim-1]==0:
316.                     self.ptr = init_ptr
317.                     break
318.             self.size = init_ptr
319.             print('经验池已加载\n')
320.         except:
321.             print('经验池加载失败，文件不存在或参数不对应')
322.
323.     def store(self, obs, act, rew, next_obs, done, cost):
324.         self.obs1_buf[self.ptr] = obs
325.         self.obs2_buf[self.ptr] = next_obs
326.         self.acts_buf[self.ptr] = act
327.         self.rews_buf[self.ptr] = rew
328.         self.costs_buf[self.ptr] = cost
329.         self.done_buf[self.ptr] = done
330.         self.ptr = (self.ptr + 1) % self.max_size
331.         self.size = min(self.size + 1, self.max_size)
332.
333.     def sample_batch(self, batch_size=32):
334.         idxs = np.random.randint(0, self.size, size=batch_size)
335.         return dict(obs1=self.obs1_buf[idxs],
336.                     obs2=self.obs2_buf[idxs],
337.                     acts=self.acts_buf[idxs],
338.                     rews=self.rews_buf[idxs],
339.                     costs=self.costs_buf[idxs],
```

```python
340.                    done=self.done_buf[idxs])
341.
342.        def savebuffer(self, Path_xls):
343.            data = dict(obs1=self.obs1_buf,
344.                       obs2=self.obs2_buf,
345.                       acts=self.acts_buf,
346.                       rews=self.rews_buf,
347.                       costs=self.costs_buf,
348.                       done=self.done_buf)
349.            title = list(data.keys())
350.            with pd.ExcelWriter(Path_xls) as writer:
351.                for i in title:
352.                    df = pd.DataFrame(data[i])
353.                    df.to_excel(writer, sheet_name=i, index=False)
354.
355.
356.
357. """
358. Embedded Physical Optimization Robust Soft Actor-Critic
359. """
360.
361. def EPO_RSAC(env_fn, actor_fn=mlp_actor, critic_fn=mlp_critic, var_fn=mlp_var,
362.        ac_kwargs_actor=dict(),ac_kwargs_critic=dict(),ac_kwargs_var=dict(),
363.        seed=0, steps_per_epoch=1200, epochs=101, replay_size=1200 * 10 * 24, gamma=0.99, cl=0.5,
364.        polyak=0.995, lr=1e-4, batch_size=1024, local_start_steps=600,
365.        max_ep_len=n_T, logger_kwargs=dict(), save_freq=50, local_update_after=int(1e3),
366.        update_freq=120, render=False,
367.        fixed_entropy_bonus=None, entropy_constraint=-1.0,
368.        fixed_cost_penalty=None, cost_constraint=None, cost_lim=None,
369.        reward_scale=1, lr_scale=1, damp_scale=0,Train=True,Continue_Training=False,
370.        Model_Portion=False,load_buffer=True
371.        ):
372.
373.    use_costs = fixed_cost_penalty or cost_constraint or cost_lim
374.
375.    # for computing cvar: CVaR=qc+pdf_cdf
376.    pdf_cdf = cl ** (-1) * norm.pdf(norm.ppf(cl))
377.
378.    logger = EpochLogger(**logger_kwargs)
379.    logger.save_config(locals())
380.
381.    # Env instantiation
382.    env, test_env = env_fn(), env_fn()
383.
```

```python
384.    #Setting seeds
385.    seed += 200 * proc_id()
386.    tf.set_random_seed(seed)
387.    np.random.seed(seed)
388.
389.    # Inputs to computation graph
390.    x_ph, a_ph, x2_ph, r_ph, d_ph, c_ph = placeholders(obs_dim, act_dim, obs_dim, None, None, None)
391.
392.    # Main outputs from computation graph
393.    with tf.variable_scope('main'):
394.        mu, pi, logp_pi = actor_fn(x_ph, a_ph, **ac_kwargs_actor)
395.        qr1, qr1_pi = critic_fn(x_ph, a_ph, pi, name='qr1', **ac_kwargs_critic)
396.        qr2, qr2_pi = critic_fn(x_ph, a_ph, pi, name='qr2', **ac_kwargs_critic)
397.        qc, qc_pi = critic_fn(x_ph, a_ph, pi, name='qc', **ac_kwargs_critic)
398.        qc_var, qc_pi_var = var_fn(x_ph, a_ph, pi, name='qc_var', **ac_kwargs_var)
399.
400.    with tf.variable_scope('main', reuse=True):
401.        # Additional policy output from a different observation placeholder
402.        # This lets us do separate optimization updates (actor, critics, etc)
403.        # in a single tensorflow op.
404.        _, pi2, logp_pi2 = actor_fn(x2_ph, a_ph, **ac_kwargs_actor)
405.
406.    # Target value network
407.    with tf.variable_scope('target'):
408.        _, qr1_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qr1', **ac_kwargs_critic)
409.        _, qr2_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qr2', **ac_kwargs_critic)
410.        _, qc_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qc', **ac_kwargs_critic)
411.        _, qc_pi_var_targ = var_fn(x2_ph, a_ph, pi2, name='qc_var', **ac_kwargs_var)
412.
413.    # Entropy bonus
414.    if fixed_entropy_bonus is None:
415.        with tf.variable_scope('entreg'):
416.            soft_alpha = tf.get_variable('soft_alpha',
417.                                         initializer=0.0,
418.                                         trainable=True,
419.                                         dtype=tf.float32)
420.        alpha = tf.nn.softplus(soft_alpha)
421.    else:
422.        alpha = tf.constant(fixed_entropy_bonus)
423.    log_alpha = tf.log(tf.clip_by_value(alpha, 1e-8, 1e8))
424.
425.    # Cost penalty
426.    if use_costs:
427.        if fixed_cost_penalty is None:
```

```python
428.            with tf.variable_scope('costpen'):
429.                soft_beta = tf.get_variable('soft_beta',
430.                                initializer=0.0,
431.                                trainable=True,
432.                                dtype=tf.float32)
433.            beta = tf.nn.softplus(soft_beta)
434.            log_beta = tf.log(tf.clip_by_value(beta, 1e-8, 1e8))
435.        else:
436.            beta = tf.constant(fixed_cost_penalty)
437.            log_beta = tf.log(tf.clip_by_value(beta, 1e-8, 1e8))
438.    else:
439.        beta = 0.0  # costs do not contribute to policy optimization
440.        print('Not using costs')
441.
442.    # Experience buffer
443.    replay_buffer = ReplayBuffer(obs_dim=obs_dim, act_dim=act_dim, size=replay_size, load_buffer=load_buffer)
444.
445.    # Count variables
446.    if proc_id() == 0:
447.        var_counts = tuple(count_vars(scope) for scope in
448.                    ['main/pi', 'main/qr1', 'main/qr2', 'main/qc', 'main/qc_var', 'main'])
449.        print((
450.            '\nNumber of parameters: \t pi: %d, \t qr1: %d, \t qr2: %d, \t qc: %d, \t qc_var: %d, \t total: %d\n') % var_counts)
451.
452.    # Min Double-Q:
453.    min_q_pi = tf.minimum(qr1_pi, qr2_pi)   # Double-DQN，
454.    min_q_pi_targ = tf.minimum(qr1_pi_targ, qr2_pi_targ)
455.
456.    qc_var = tf.clip_by_value(qc_var, 1e-8, 1e8)
457.    qc_pi_var = tf.clip_by_value(qc_pi_var, 1e-8, 1e8)
458.    qc_pi_var_targ = tf.clip_by_value(qc_pi_var_targ, 1e-8, 1e8)
459.
460.    # Targets for Q and V regression
461.    q_backup = tf.stop_gradient(r_ph + gamma * (1 - d_ph) * (min_q_pi_targ - alpha * logp_pi2))
462.    # qc_pi_targ
463.    qc_backup = tf.stop_gradient(c_ph + gamma * (1 - d_ph) * qc_pi_targ)
464.    qc_var_backup = tf.stop_gradient(
465.        c_ph ** 2 + 2 * gamma * c_ph * qc_pi_targ + gamma ** 2 * qc_pi_var_targ + gamma ** 2 * qc_pi_targ ** 2 - qc ** 2)
466.    qc_var_backup = tf.clip_by_value(qc_var_backup, 1e-8, 1e8)
467.
468.    cost_constraint = cost_lim * (1 - gamma ** max_ep_len) / (1 - gamma) / max_ep_len
```

```python
469.    damp = damp_scale * tf.reduce_mean(cost_constraint - qc - pdf_cdf * tf.sqrt(qc_var))
470.
471.    # Soft actor-critic losses
472.    pi_loss = tf.reduce_mean(alpha * logp_pi - min_q_pi + (beta - damp) * (qc_pi + pdf_cdf * (qc_pi_var
        ** 0.5)))
473.    qr1_loss = 0.5 * tf.reduce_mean((q_backup - qr1) ** 2)
474.    qr2_loss = 0.5 * tf.reduce_mean((q_backup - qr2) ** 2)
475.    qc_loss = 0.5 * tf.reduce_mean((qc_backup - qc) ** 2)
476.    qc_var_loss = 0.5 * tf.reduce_mean(qc_var + qc_var_backup - 2 * ((qc_var * qc_var_backup) ** 0.5))

477.    q_loss = qr1_loss + qr2_loss + qc_loss + qc_var_loss
478.
479.    # Loss for alpha
480.    entropy_constraint *= act_dim
481.    pi_entropy = -tf.reduce_mean(logp_pi)
482.    alpha_loss = - alpha * (entropy_constraint - pi_entropy)
483.    print('using entropy constraint', entropy_constraint)
484.
485.    # Loss for beta
486.    if use_costs:
487.        if cost_constraint is None:
488.
489.            '''
490.            Convert assuming equal cost accumulated each step
491.            Note this isn't the case, since the early in episode doesn't usually have cost,
492.            but since our algorithm optimizes the discounted infinite horizon from each entry
493.            in the replay buffer, we should be approximately correct here.
494.            It's worth checking empirical total undiscounted costs to see if they match.
495.            '''
496.
497.            cost_constraint = cost_lim * (1 - gamma ** max_ep_len) / (1 - gamma) / max_ep_len
498.        print('using cost constraint', cost_constraint)
499.        beta_loss = beta * (cost_constraint - qc - pdf_cdf * tf.sqrt(qc_var))    # qc + pdf_cdf * tf.sqrt(qc_var)

500.
501.    # Policy train op
502.    # (has to be separate from value train op, because qr1_pi appears in pi_loss)
503.    train_pi_op = MpiAdamOptimizer(learning_rate=lr).minimize(pi_loss, var_list=get_vars('main/pi'), na
        me='train_pi')
504.
505.    # Value train op
506.    with tf.control_dependencies([train_pi_op]):
507.        train_q_op = MpiAdamOptimizer(learning_rate=lr).minimize(q_loss, var_list=get_vars('main/q'), na
            me='train_q')
```

```python
508.
509.    if fixed_entropy_bonus is None:
510.        entreg_optimizer = MpiAdamOptimizer(learning_rate=lr)
511.        with tf.control_dependencies([train_q_op]):
512.            train_entreg_op = entreg_optimizer.minimize(alpha_loss, var_list=get_vars('entreg'))
513.
514.    if use_costs and fixed_cost_penalty is None:
515.        costpen_optimizer = MpiAdamOptimizer(learning_rate=lr * lr_scale)
516.
517.        if fixed_entropy_bonus is None:
518.            with tf.control_dependencies([train_entreg_op]):
519.                train_costpen_op = costpen_optimizer.minimize(beta_loss, var_list=get_vars('costpen'))
520.        else:
521.            with tf.control_dependencies([train_q_op]):
522.                train_costpen_op = costpen_optimizer.minimize(beta_loss, var_list=get_vars('costpen'))
523.
524.    # Polyak averaging for target variables
525.    target_update = get_target_update('main', 'target', polyak)
526.
527.    # Single monolithic update with explicit control dependencies
528.    with tf.control_dependencies([train_pi_op]):
529.        with tf.control_dependencies([train_q_op]):
530.            grouped_update = tf.group([target_update])
531.
532.    if fixed_entropy_bonus is None:
533.        grouped_update = tf.group([grouped_update, train_entreg_op])
534.    if use_costs and fixed_cost_penalty is None:
535.        grouped_update = tf.group([grouped_update, train_costpen_op])
536.
537.    def get_action(o, deterministic=False):
538.        act_op = mu if deterministic else pi
539.        return sess.run(act_op, feed_dict={x_ph: o.reshape(1, -1)})[0]
540.
541.    def test_agent(num_wind=0, deterministic=True, n_test_sce=100):
542.        o = test_env.reset(n = num_wind,Train = Train)
543.        ep_ret, ep_cost, ep_len, ep_rU, ep_rH, ep_rP, ep_cV, ep_cD, ep_cH =\
544.            0, 0, 0, 0, 0, 0, 0, 0, 0
545.        u_g =   np.zeros((n_gen,n_T))
546.        ROCOF_t = np.zeros(n_T)
547.        f_Nadir_t = np.zeros(n_T)
548.        f_Qss_t = np.zeros(n_T)
549.        curtailment_t = np.zeros(n_T)
550.        shedding_t = np.zeros(n_T)
551.        for i in range(max_ep_len):
```

```python
552.            # Take deterministic actions at test time
553.            o2, r, c, _, reward_u, reward_HESS, reward_p, cost_vio, cost_deta, cost_hess,\
554.                ROCOF, f_Nadir, f_Qss, curtailment, shedding, u_state =\
555.                test_env.step(get_action(o, deterministic),o,ep_len,Train = Train,n_test_sce=n_test_sce)
556.
557.            ep_ret += r
558.            ep_cost += c
559.            ep_len += 1
560.            ep_rU += reward_u
561.            ep_rH += reward_HESS
562.            ep_rP += reward_p
563.            ep_cV += cost_vio
564.            ep_cD += cost_deta
565.            ep_cH += cost_hess
566.            u_g[:,i] = u_state
567.            ROCOF_t[i] = ROCOF
568.            f_Nadir_t[i] = f_Nadir
569.            f_Qss_t[i] = f_Qss
570.            curtailment_t[i] = curtailment
571.            shedding_t[i] = shedding
572.            o = o2
573.        return ep_ret, ep_cost, ep_len, ep_rU, ep_rH, ep_rP, ep_cV, ep_cD, ep_cH,\
574.            u_g, ROCOF_t, f_Nadir_t, f_Qss_t,curtailment_t, shedding_t
575.
576.
577.    config = tf.ConfigProto()
578.    config.gpu_options.per_process_gpu_memory_fraction = 0.8
579.    # config.allow_soft_placement = True
580.    # config.log_device_placement = True
581.    # config.gpu_options.allow_growth = True
582.    sess = tf.Session(config=config)
583.
584.
585.    if not Train:
586.        saver = tf.train.Saver()
587.        # try:
588.        saver.restore(sess,tf.train.latest_checkpoint('./saved_models/Case118_%d'%args.Model_version))
589.        '''
590.        var_list = tf.global_variables()
591.        for var in var_list:
592.            if var.name.startswith("main"):
593.                print(var.name, var.shape)
594.        '''
595.        num_wind = 0
```

```python
596.    n_test_sce = 1000
597.    start = time.time()
598.
599.    start = time.time()
600.    ep_ret, ep_cost, ep_len, ep_rU, ep_rH, ep_rP, ep_cV, ep_cD, ep_cH,\
601.        u_g, ROCOF_t, f_Nadir_t, f_Qss_t,curtailment_t, shedding_t = \
602.        test_agent(num_wind = num_wind,deterministic=True,n_test_sce=n_test_sce)
603.    end = time.time()
604.    n_ROCOF, n_f_Nadir, n_f_Qss = 0, 0, 0
605.    for i in range(n_T):
606.        if ROCOF_t[i]>ROCOFmax:
607.            n_ROCOF += 1
608.        if 50-f_Nadir_t[i]>delt_fmax:
609.            n_f_Nadir += 1
610.        if f_Qss_t[i]>0:
611.            n_f_Qss += 1
612.
613.    print("求解时间:%.4f 秒"%(end-start))
614.
615.    print('\n==================Test Result==================')
616.    print('        The scenario of wind        |      %d'%num_wind)
617.    print('        The value of Ep_ret         |      %.2f'%ep_ret)
618.    print('        The value of Ep_cost        |      %.2f'%ep_cost)
619.    print('----------------------              --------------')
620.    print('          The Start_up cost         |      %.2f'%ep_rU)
621.    print('          The operating cost        |      %.2f'%(ep_rU+ep_rP))
622.    print('          Total cost for HESS       |      %.2f'%(ep_rU+ep_rP+ep_rH))
623.    print('   Expected energy curtailment |     %.2f'%(sum(curtailment_t))+' MWh')
624.    print('        Expected Load Shedding     |      %.2f'%(sum(shedding_t))+' MWh')
625.    print('            Maximum RoCoF         |      %.2f'%(max(ROCOF_t))+' Hz/s')
626.    print('        Minimum Frequency Nadir    |     %.2f'%(min(f_Nadir_t))+' Hz')
627.    print('Violation Probability of RoCoF|    %.2f'%(n_ROCOF/n_T*100)+' %')
628.    print('Violation Probability of Nadir|    %.2f'%(n_f_Nadir/n_T*100)+' %')
629.    print(' Violation Probability of Qss |    %.2f'%(n_f_Qss/n_T*100)+' %')
630.    print('==================================================')
631.
632.    t_range = range(n_T)
633.    fig, (ax1, ax2, ax3) = plt.subplots(3)
634.
635.    ax1.plot(t_range, ROCOF_t)
636.    ax1.set_title('ROCOF')
637.
638.    ax2.plot(t_range, f_Nadir_t)
639.    ax2.set_title('f_Nadir')
```

```python
640.
641.        ax3=sns.heatmap(u_g, linewidths = 0.05, linecolor= 'red', annot=True)
642.        plt.show
643.
644.        if args.SaveFig:
645.            now = datetime.datetime.now()
646.            savefname = './Debug_record/%d.%d.%d/Figture'%(now.year,now.month,now.day)
647.            os.makedirs(savefname,exist_ok=True)
648.            plt.savefig(savefname+'/Output result_%d_%d.png'%(num_wind,args.Model_version),dpi=500)
649.
650.        plt.show()
651.
652.   else:
653.        writer = tf.summary.FileWriter(".logs/",sess.graph)
654.        if Continue_Training:
655.            if Model_Portion:
656.                target_init = get_target_update('main', 'target', 0.0)
657.                sess.run(tf.global_variables_initializer())
658.                sess.run(target_init)
659.                sess.run(sync_all_params())
660.                Loading_model = [get_vars('main/q'),get_vars('target'),get_vars('costpen'),get_vars('entreg')]
661.                saver = tf.train.Saver(Loading_model)
662.                saver.restore(sess,tf.train.latest_checkpoint('./saved_models/Case118_%d'%args.Model_versio
    n))
663.            else:
664.                saver = tf.train.Saver()
665.                saver.restore(sess,tf.train.latest_checkpoint('./saved_models/Case118_%d'%args.Model_versio
    n))
666.        else:
667.            # Initializing targets to match main variables
668.            # As a shortcut, use our exponential moving average update w/ coefficient zero
669.            target_init = get_target_update('main', 'target', 0.0)
670.            sess.run(tf.global_variables_initializer())
671.            sess.run(target_init)
672.            sess.run(sync_all_params())
673.        # Setup model saving  保存模型
674.        logger.setup_tf_saver(sess, inputs={'x': x_ph, 'a': a_ph},
675.                      outputs={'mu': mu, 'pi': pi, 'qr1': qr1, 'qr2': qr2, 'qc': qc})
676.
677.        start_time = time.time()
678.        o, r, d, ep_ret, ep_cost, ep_len= env.reset(), 0, False, 0, 0, 0
679.        total_steps = steps_per_epoch * epochs * max_ep_len
680.
681.        # variables to measure in an update
```

```python
682.        vars_to_get = dict(LossPi=pi_loss, LossQR1=qr1_loss, LossQR2=qr2_loss, LossQC=qc_loss, Loss
      QCVar=qc_var_loss,
683.                       QR1Vals=qr1, QR2Vals=qr2, QCVals=qc, QCVar=qc_var, LogPi=logp_pi, PiEntropy=
      pi_entropy,
684.                       Alpha=alpha, LogAlpha=log_alpha, LossAlpha=alpha_loss)
685.        if use_costs:
686.            vars_to_get.update(dict(Beta=beta, LogBeta=log_beta, LossBeta=beta_loss))
687.
688.        print('starting training', proc_id())
689.
690.        # Main loop: collect experience in env and update/log each epoch
691.        cum_cost = 0
692.        local_steps = 0
693.        local_steps_per_epoch = steps_per_epoch // num_procs()
694.        local_batch_size = batch_size // num_procs()
695.        epoch_start_time = time.time()
696.        for t in range(total_steps // num_procs()):
697.            """
698.            Until local_start_steps have elapsed, randomly sample actions
699.            from a uniform distribution for better exploration. Afterwards,
700.            use the learned policy.
701.            """
702.            if t > local_start_steps:
703.                a = get_action(o)
704.            else:
705.                a = env.action_sample()
706.
707.            # Step the env
708.            o2, r, c, d = env.step(a,o,ep_len)
709.            r *= reward_scale    # yee-haw
710.            ep_ret += r
711.            ep_cost += c
712.            ep_len += 1
713.            local_steps += 1
714.
715.            # Track cumulative cost over training
716.            cum_cost += c
717.
718.            # Store experience to replay buffer
719.            replay_buffer.store(o, a, r, o2, d, c)
720.
721.            # Super critical, easy to overlook step: make sure to update
722.            # most recent observation!
723.            o = o2
```

```python
724.
725.         if d or (ep_len == max_ep_len):
726.             logger.store(EpRet=ep_ret, EpCost=ep_cost)
727.             o, r, d, ep_ret, ep_cost, ep_len = env.reset(n = np.random.randint(n_wind_data)),\
728.                 0, False, 0, 0, 0
729.
730.         if t > 0 and t % update_freq == 0:
731.
732.             for j in range(update_freq):
733.                 batch = replay_buffer.sample_batch(local_batch_size)
734.                 feed_dict = {x_ph: batch['obs1'],
735.                              x2_ph: batch['obs2'],
736.                              a_ph: batch['acts'],
737.                              r_ph: batch['rews'],
738.                              c_ph: batch['costs'],
739.                              d_ph: batch['done'],
740.                              }
741.
742.                 if t < local_update_after:     # 600
743.                     logger.store(**sess.run(vars_to_get, feed_dict))
744.
745.                 else:
746.                     values, _ = sess.run([vars_to_get, grouped_update], feed_dict)
747.                     logger.store(**values)
748.
749.             ETA = (time.time()-epoch_start_time)*(total_steps//update_freq-t//update_freq)
750.             print(('Training: [epoch:%d|%d] [batch:%d|%d] ETA %d:%d:%d')\
751.                 % (t // (local_steps_per_epoch * max_ep_len),epochs-1,\
752.                    (t%(local_steps_per_epoch * max_ep_len)//max_ep_len),steps_per_epoch-1,\
753.                    ETA//3600, ETA%3600//60, ETA%60))
754.             epoch_start_time = time.time()
755.
756.
757.         # End of epoch wrap-up
758.         if t > 0 and t % (local_steps_per_epoch * max_ep_len) == 0:
759.             epoch = t // (local_steps_per_epoch * max_ep_len)
760.             cumulative_cost = mpi_sum(cum_cost)
761.             cost_rate = cumulative_cost / ((epoch + 1) * steps_per_epoch)
762.
763.
764.             # Save model
765.             if (epoch % save_freq == 0) or (epoch == epochs - 1):
766.                 saver = tf.train.Saver()
767.                 saver.save(sess,'./saved_models/Case118_%d/Model'%epoch)
```

```python
768.            try:
769.                now_time = datetime.datetime.now()
770.                os.makedirs('./Debug_record/%d.%d.%d'%(now_time.year,now_time.month,now_time.day),exist_ok=True)
771.                saver.save(sess,'./Debug_record/%d.%d.%d/Case118_%d/Model'%(now_time.year,now_time.month,now_time.day,epoch))
772.                Path_xls = './Debug_record/%d.%d.%d/ReplayBuffer'%(now_time.year,now_time.month,now_time.day)
773.                os.makedirs(Path_xls,exist_ok=True)
774.                replay_buffer.savebuffer('./logger/Buffer.xlsx')
775.                replay_buffer.savebuffer(Path_xls=Path_xls+'/Buffer_%d.xlsx'%now_time.hour)
776.            except:
777.                print('路径出错')
778.
779.            logger.log_tabular('Epoch', epoch)
780.            logger.log_tabular('EpRet', with_min_and_max=True)
781.            logger.log_tabular('EpCost', with_min_and_max=True)
782.            logger.log_tabular('CumulativeCost', cumulative_cost)
783.            logger.log_tabular('CostRate', cost_rate)
784.            logger.log_tabular('LossPi', with_min_and_max=True)
785.            logger.log_tabular('LossQR1', with_min_and_max=True)
786.            logger.log_tabular('LossQR2', average_only=True)
787.            logger.log_tabular('LossQC', with_min_and_max=True)
788.            logger.log_tabular('LossQCVar', with_min_and_max=True)
789.            logger.log_tabular('LossAlpha', average_only=True)
790.            logger.log_tabular('LogAlpha', average_only=True)
791.            logger.log_tabular('Alpha', average_only=True)
792.            if use_costs:
793.                logger.log_tabular('LossBeta', average_only=True)
794.                logger.log_tabular('LogBeta', average_only=True)
795.                logger.log_tabular('Beta', average_only=True)
796.            logger.log_tabular('PiEntropy', with_min_and_max=True)
797.            logger.log_tabular('TotalTime', time.time() - start_time)
798.            logger.dump_tabular()
799.    writer.close()
800.
801.
802.
803. if __name__ == '__main__':
804.    mpi_fork(args.cpu)
805.
806.    logger_kwargs = setup_logger_kwargs(args.exp_name, args.seed)
807.    logger_kwargs = setup_logger_kwargs(args.exp_name)
808.    logger_kwargs = args.logger_kwargs_str
```

```
809.
810.    EPO_RSAC(lambda: unPower(uncertain=True, n_sce=5), actor_fn=mlp_actor, critic_fn=mlp_critic,
811.        ac_kwargs_actor=dict(hidden_sizes=args.hidden_sizes_actor, cnn_sizes=args.cnn_actor),
812.        ac_kwargs_critic=dict(hidden_sizes=args.hidden_sizes_critic, cnn_sizes=args.cnn_critic),
813.        ac_kwargs_var=dict(hidden_sizes=args.hidden_sizes_var, cnn_sizes=args.cnn_var),
814.        gamma=args.gamma, cl=args.cl, seed=args.seed, epochs=args.epochs, replay_size=24000, batch_size=args.batch_size,
815.        logger_kwargs=logger_kwargs, steps_per_epoch=args.steps_per_epoch,
816.        update_freq=args.update_freq, lr=args.lr, render=args.render,
817.        local_start_steps=args.local_start_steps, save_freq=args.save_freq, local_update_after=args.local_update_after,
818.        fixed_entropy_bonus=args.fixed_entropy_bonus, entropy_constraint=args.entropy_constraint,
819.        fixed_cost_penalty=args.fixed_cost_penalty, cost_constraint=args.cost_constraint, cost_lim=args.cost_lim,
820.        lr_scale=args.lr_s, damp_scale=args.damp_s, Train=args.Train, Continue_Training=args.Continue,
821.        Model_Portion=args.Portion,load_buffer=args.load_buffer
822.        )
```