HW6

Geoffrey Woollard

My code lives in the repo https://github.com/geoffwoollard/prob_prog

Acknowledgments

- discussions with Jordan Lovrod, Ilias Karimalis, Gaurav Bhatt
- starter code for `smc.resample_particles` and `smc.SMC` from Masoud Mokhtari

In [1]:
```
from dill.source import getsource, getsourcelines
```

In [2]:
```
from smc import resample_particles, SMC

list_of_programs = [resample_particles, SMC]

for program in list_of_programs:
    for line_number, function_line in enumerate(getsourcelines(program)[0]):
        print(line_number, function_line,end='')
    print()
```

```
0 def resample_particles(particles, log_weights):
1     """
2     Eq. 4.24 in course textbook (https://arxiv.org/abs/1809.10756v2, pp. 122)
3     See Algorithm 15 in the course textbook, section 6.7 Sequantial Monte Carlo, p. 176
4     """
5     log_weights = tensor(log_weights)
6     n_particles = log_weights.size().numel()
7 
8     unnormalized_particle_weights = torch.exp(log_weights).detach().numpy()
9 
10     particle_idxs = np.random.choice(
11         a=range(n_particles),
12         size=n_particles,
13         p=unnormalized_particle_weights/unnormalized_particle_weights.sum(),
14         replace=True,
15         )
16     #print('particle_idxs',particle_idxs)
17 
18     new_particles = []
19     for idx in range(n_particles):
20         new_particles.append(particles[particle_idxs[idx]]) # TODO: copy?
21 
22     log_Z = np.log(np.sum(unnormalized_particle_weights)/n_particles)
23     return log_Z, new_particles

0 def SMC(n_particles, exp,do_log=False):
1 
2     particles = []
3     weights = []
4     logZs = []
5     output = lambda x: x
6 
7     for i in range(n_particles):
8 
9         res = evaluate(exp, env=None)('addr_start', output)
10         logW = 0.
11 
12 
13         particles.append(res)
14         weights.append(logW)
15 
16     #can't be done after the first step, under the address transform, so this should be fine:
17     done = False
18     smc_cnter = 0
19     while not done:
20         if do_log: print('In SMC step {}, Zs: '.format(smc_cnter), logZs)
21         for i in range(n_particles): #Even though this can be parallelized, we run it serially
22             res = run_until_observe_or_end(particles[i]) # particle i at next breakbpoint
23             if 'done' in res[2]: #this checks if the calculation is done
24                 particles[i] = res[0]
25                 if i == 0:
26                     done = True  #and enforces everything to be the same as the first particle
27                     address = ''
28                 else:
29                     if not done: # triggered when i=0 was not done and i>0 was done
30                         raise RuntimeError('Failed SMC, finished one calculation before the other')
31             else:
32                 #TODO: check particle addresses, and get weights and continuations
33                 particles[i] = res
34                 cont, args, sigma = res
35                 assert 'observe' == sigma['type']
36                 weights[i] = sigma['distribution'].log_prob(sigma['observed_constant'])
37 
38                  # check particle addresses
39                 if i == 0:
40                     break_point_address = sigma['address']
41                 else:
42                     if sigma['address'] != break_point_address:
43                         assert False, 'particles at different break points'
44 
45 
46 
47         if not done:
48             #resample and keep track of logZs
49             logZn, particles = resample_particles(particles, weights)
50             logZs.append(logZn)
51         smc_cnter += 1
52     logZ = sum(logZs)
53     return logZ, particles
```