CPSC532W Homework 5

Justin Reiher
Student ID: 37291151
CWL: reiher

Link to public repository for homework 5:

`https://github.com/justinreiher/probProg_Fall2021/tree/main/CS532-HW5`

The HOPPL is implemented following Peter Norvig's tutorial `https://norvig.com/lispy.html` very closely. Particularly the `Procedure` and `Environment` classes:

```python
class Env(dict):
    "An environment: a dictionary of ('var':val) paris with and outer Env."
    def __init__(self,params=(),args=(),outer=None):
        self.update(zip(params,args))
        self.outer = outer
    def get(self,var):
        "Find the innermost Env where var appears."
        return self[var] if (var in self) else self.outer.get(var)

class Procedure(object):
    "A user-defined FOPPL procedure."
    def __init__(self,params,body,env):
        self.params, self.body, self.env = params,body,env
    def __call__(self,*args):
        return evaluate(self.body, Env(self.params, args, self.env))
```

The evaluator itself likewise follows the format and style in the tutorial augmented with `sample` and `observe` where `observe` in this case does nothing interesting other than return the observed value:

```python
def evaluate(exp, env=None): #TODO: add sigma, or something
    # if the environment is not set, then get the standard environment, and add
    # sigma to this environment
    if env is None:
        env = standard_env()
        env = env.update({'sig':''})

    if isinstance(exp,Symbol):  #variable reference
        e = env.get(exp)
        if e == None:
            e = exp
        return e
    elif not isinstance(exp,List): #constant case
        return torch.tensor(float(exp))

    op, *args = exp
    if op == 'if':
        (test,conseq,alt) = args
        exp = (conseq if evaluate(test,env) else alt)
        return evaluate(exp,env)
    elif op == 'fn': #procedure definition
        (params,body) = args
        return Procedure(params,body,env)
    elif op == 'sample':
        v = evaluate(args[0],env)
        d = evaluate(args[1],env)
        return d.sample()
    elif op == 'observe':
        v = evaluate(args[0],env)
        d = evaluate(args[1],env)
        c = evaluate(args[2],env)
        return c
    else:
        proc = evaluate(op,env)
        vals = [evaluate(arg,env) for arg in args]
        return proc(*vals)

    return
```

1 Program 1: Deterministic and Probabilistic Tests

Output demonstrating all tests pass:

```
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
FOPPL Tests passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
/home/justin/Research/Research/ProbProg/CS532-HW5/primitives.py:244: UserWarning: To copy co
return torch.cat((torch.tensor([val]),torch.tensor(l)),0)
Test passed
All deterministic tests passed
('normal', 5, 1.4142136)
p value 0.4392251209556768
('beta', 2.0, 5.0)
p value 0.20362142284502927
('exponential', 0.0, 5.0)
p value 0.4026319736237799
('normal', 5.3, 3.2)
p value 0.6117760761512494
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.42402962493527685
('normal', 0, 1.44)
p value 0.018257659736088516
All probabilistic tests passed
```

The warning in the HOPPL test 12:

```
/home/justin/Research/Research/ProbProg/CS532-HW5/primitives.py:244: UserWarning: To copy co
return torch.cat((torch.tensor([val]),torch.tensor(l)),0)
```

is telling me that I should not call `torch.tensor(l)` on a tensor object that already exists. However the behaviour is correct, which is to say that if the list is not a `torch.tensor(l)` it will create one, if it already exists then it returns the same list.

2 Running Programs

All programs are run with 10k samples and the results are shown below

2.1 Program 1

Output from running program 1:

```
Sample of prior of program 1:
Elapsed time for program  1 .daphne is:  0:05:11.798638  seconds
Mean of samples:  tensor(99.1019)
Variance of samples:  tensor(9923.6123)
```

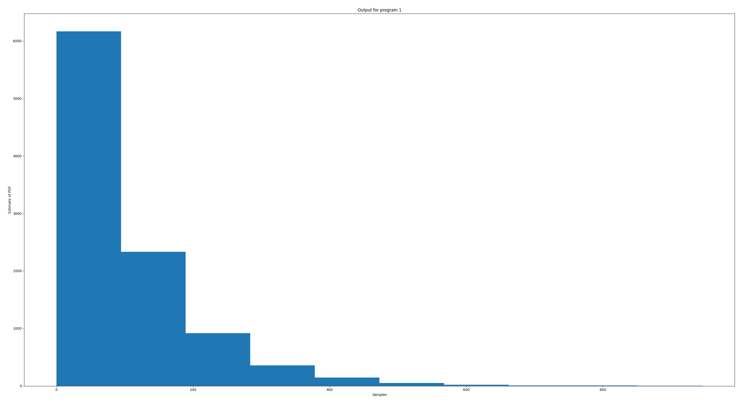

Figure 1: Histogram for Program 1

2.2 Program 2

Output from running program 2:

```
Sample of prior of program 2:
Elapsed time for program  2 .daphne is:  0:00:29.480438  seconds
Mean of samples:  tensor(0.9736)
Variance of samples:  tensor(5.0724)
```

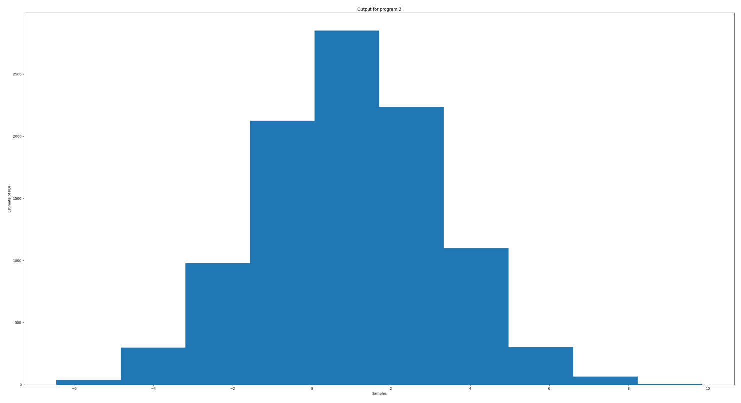

Figure 2: Histogram for Program 2

2.3 Program 3

Output from running program 3:

```
Sample of prior of program 3:
Elapsed time for program  3 .daphne is:  0:02:03.474374  seconds
```

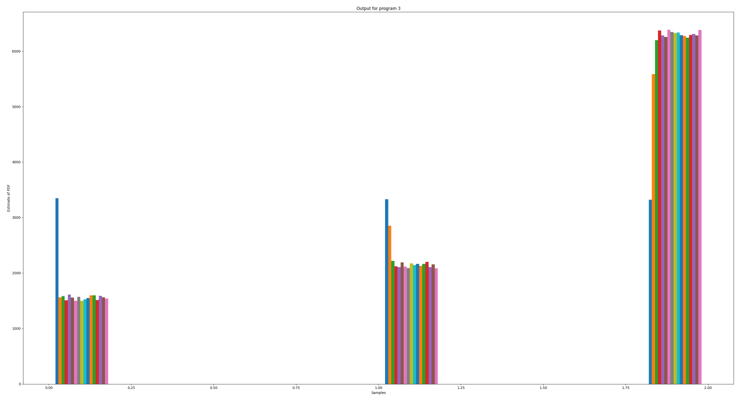

Figure 3: Histogram for Program 3