

# HW5

Geoffrey Woollard

My code lives in the repo [https://github.com/geoffwoollard/prob\\_prog](https://github.com/geoffwoollard/prob_prog)

## Acknowledgments

For this assignment I primarily acknowledge the patient discussions with Jordan Lovrod, where she held my hand and walked me through the recursive structure of declaring lambda fn's, and the disrinctions between a return from calling something from the `Procedure` class and a `Procedure` object itself. She also supplied code for the classes `Env`, and `Procedure`.

- Ilias Karimalis on the address structure and how it dynamically tracks the evaluations of the dynamic computational graph
- Alan Milligan, Gaurav Bhatt, Masoud Mokhtari and everyone else on Slack for what to do when hitting snags, especially on `empty?`, and `hash-map`, `put` and `get` for handling `str` keys.
- Kim Dinh's comments on the interpretation of problem 1 as a Geometric distribution
- the baseline results provided by Lironne Kurzman
- hw6 starter code for some parts of the `eval_hoppl`, especially the double quotes, and some `primitives`

In [1]:

```
#!/bash
python evaluator.py

FOPPL test 1 passed
FOPPL test 2 passed
FOPPL test 3 passed
FOPPL test 4 passed
FOPPL test 5 passed
FOPPL test 6 passed
FOPPL test 7 passed
FOPPL test 8 passed
FOPPL test 9 passed
FOPPL test 10 passed
FOPPL test 11 passed
FOPPL test 12 passed
FOPPL test 13 passed
HOPPL test 1 passed
HOPPL test 2 passed
HOPPL test 3 passed
HOPPL test 4 passed
HOPPL test 5 passed
HOPPL test 6 passed
HOPPL test 7 passed
HOPPL test 8 passed
HOPPL test 9 passed
HOPPL test 10 passed
HOPPL test 11 passed
HOPPL test 12 passed
All deterministic tests passed
('normal', 5, 1.4142136)
p value 0.9521132603290829
('beta', 2.0, 5.0)
p value 0.8268444534006023
('exponential', 0.0, 5.0)
p value 0.3859297323773355
('normal', 5, 3.2)
p value 0.6247795543285082
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.8588495916821821
('normal', 0, 1.44)
p value 0.6629150112938403
All probabilistic tests passed
1

Sample of prior of program 1:
tensor(56)
2

Sample of prior of program 2:
tensor(-1.5379)
3

Sample of prior of program 3:
tensor([1, 2, 1, 0, 0, 2, 1, 1, 1, 0, 0, 0, 2, 0, 2, 1, 2])
```

## Code snippets

In [2]:

```
from dill.source import getsource, getsourcelines
```

In [3]:

```
from evaluator import evaluate, Env, Procedure, standard_env, eval_hoppl

list_of_programs = [evaluate, Env, Procedure, standard_env, eval_hoppl]

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 evaluate(exp, env=None, do_log=False): #TODO: add sigma, or something
1
2     if env is None:
3         env = standard_env()
4
5     fn, _ = eval_hoppl(exp, env, sigma=None, do_log=do_log)
6     ret, sigma = fn(())
7     if do_log: print('return', ret)
8     return ret

0 class Env():
1     "An environment: a dict of {'var': val} pairs, with an outer Env."
2     def __init__(self, params=(), args=(), outer=None):
3         self.data = pmap(zip(params, args))
4         self.outer = outer
5         if outer is None:
6             self.level = 0
7         else:
8             self.level = outer.level+1
9     def __getitem__(self, item):
10         return self.data[item]
11     def find(self, var):
12         "Find the innermost Env where var appears."
13         if (var in self.data):
14             return self
15         else:
16             if self.outer is not None:
17                 return self.outer.find(var)
18             else:
19                 raise RuntimeError('var "{}" not found in outermost scope'.format(var))
20
21     def print_env(self, print_lowest=False):
22         print_line = 1 if print_lowest == False else 0
23         outer = self
24         while outer is not None:
25             if outer.level >= print_limit:
26                 print('Scope on level {}'.format(outer.level))
27                 if 'e' in outer:
28                     print('Found e, ')
29                     print(outer['f'].body)
30                     print(outer['f'].params)
31                     print(outer['f'].env)
32                 print(outer, '\n')
33                 outer = outer.outer

0 class Procedure(object):
1     "A user-defined Scheme procedure."
2     def __init__(self, params, body, env, do_log):
3         self.params = self.body, self.env = params, body, env
4         self.do_log = do_log
5     def __call__(self, *args):
6         new_env = copy.deepcopy(self.env)
7         return eval_hoppl(self.body, Env(self.params, args, new_env), do_log=self.do_log)

0 def standard_env():
1     "An environment with some Scheme standard procedures."
2     env = Env(penv.keys(),, penv.values())
3     return env

0 def eval_hoppl(x, env=standard_env(), sigma=None, do_log=False):
1     # TODO: remove default env=standard_env()
2
3     if do_log: print('x', x)
4
5     if isinstance(x, list):
6         op, param, *args = x
7
8         if op == 'if':
9             assert len(x) == 4
10             test, consequ, alt = x[1:4]
11             if do_log: print('case if: x', x)
12             exp = (conseq if eval_hoppl(test, env, sigma, do_log=do_log)[0] else alt) # be careful to get ret
13             if do_log: print('case if: exp', exp)
14             return eval_hoppl(exp, env, sigma, do_log=do_log)
15
16         if op == 'sample':
17             if do_log: print('case sample: x', x)
18             _, address, exp, distribution = x
19             distribution, sigma = eval_hoppl(exp, distribution, env, sigma, do_log=do_log)
20             if do_log: print('case sample: distribution', distribution)
21             evaluated_sample = distribution.sample()
22             if do_log: print('case sample: evaluated_sample', evaluated_sample)
23             return evaluated_sample, sigma
24
25         elif op == 'observe':
26             if do_log: print('case observe: (pass)')
27             return 'observed', sigma
28         elif op == 'push-address':
29             return '', sigma
30         elif op == 'fn':
31             if do_log: print('case fn: args', args)
32             body = args[0]
33             return Procedure(param, body, env, sigma, do_log=do_log), sigma # has eval in it
34             # param ['alpha', 'x']
35             # body [['', ['push-address', 'alpha', 'addr2']], 'x', 'x']
36             # env
37
38         else:
39             if do_log: print('case else: x', x)
40             proc, _ = eval_hoppl(op, env, sigma, do_log=do_log)
41             vals = []
42             if do_log: print('case else: args', args)
43             args.extend([eval_hoppl(arg, env, sigma, do_log=do_log)[0] for arg in args])
44             if do_log: print('case else: vals', vals)
45             if do_log: print('case Procedure:', proc, vals)
46
47             if isinstance(proc, Procedure): # lambdas, not primitives
48                 r, _ = proc('vals')
49                 if do_log: print('case Procedure: r', r)
50             else:
51                 if do_log: print('case primitives: vals[1:]', vals[1:])
52                 r = proc(vals[1:]) # primitives
53                 if do_log: print('case primitives: r', r)
54
55             return r, sigma
56
57     # base cases
58     def is_instance(x, str):
59         if x[0] == '"': # daphne output: strings have double, double quotes
60             return x[1:-1], sigma
61
62         lowest_env = env.find(x)
63         return lowest_env[x], sigma
64
65     elif isinstance(x, (float, int, bool)):
66         return torch.tensor(x), sigma
67
68     else:
69         raise ValueError('unknown expression case')
```

In [4]:

```
import primitives

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

0 """primitives (and distributions) used to evaluate algorithm 6 in
1 van de Meent, J.-W., Paige, B., Yang, H., & Wood, F. (2018).
2 An Introduction to Probabilistic Programming, XX(Xx), 1-221.
3 http://doi.org/10.1561/XXXXXXX
4
5 Acknowledgements to Yuan T https://github.com/yuant95/CPSC532W/blob/master/HW2/primitives.py
6 and
7 Masoud Mokhtari https://github.com/MasoudMo/CPSC-532W/blob/master/HW2/primitives.py
8 and
9 RM6 starter code https://www.cs.ubc.ca/~fwood/CS532W-539W/homework/6.html
10 """
11
12 import torch
13 from torch import tensor
14
15 number = (float, int)
16 distribution_types = (
17     torch.distributions.Normal,
18     torch.distributions.Beta,
19     torch.distributions.Uniform,
20     torch.distributions.Exponential,
21     torch.distributions.Categorical,
22     torch.distributions.bernoulli.Bernoulli,
23     torch.distributions.dirichlet.Dirichlet,
24     torch.distributions.gamma.Gamma,
25 )
26
27
28 def two_arg_op_primitive(op, arg1, arg2):
29     arg1, arg2 = arg1, arg2
30     return op(arg1, arg2)
31
32
33 def add_primitive(arg1, arg2):
34     return two_arg_op_primitive(torch.add, arg1, arg2)
35
36
37 def subtract_primitive(arg1, arg2):
38     return two_arg_op_primitive(torch.subtract, arg1, arg2)
39
40
41 def multiply_primitive(arg1, arg2):
42     return two_arg_op_primitive(torch.multiply, arg1, arg2)
43
44
45 def divide_primitive(arg1, arg2):
46     return two_arg_op_primitive(torch.divide, arg1, arg2)
47
48
49 def one_arg_op_primitive(op, arg):
50     arg0 = arg[0] # because list of len one passed, i.e. [arg0]
51     return op(arg0)
52
53
54 def sqrt_primitive(arg):
55     return one_arg_op_primitive(torch.sqrt, arg)
56
57
58 def get_primitive(vector_and_index):
59     vector, index = vector_and_index
60
61     if isinstance(index, str):
62         index_safe = index
63     elif torch.is_tensor(index):
64         index_safe = index.item()
65     else:
66         assert False, 'index type {} case not implemented'.format(type(index))
67
68     if isinstance(vector, dict):
69         return vector[index_safe]
70     elif torch.is_tensor(vector):
71         assert not isinstance(index, str)
72         return vector[index.long()]
73     elif isinstance(vector, list):
74         index_int = torch.tensor(int(index))
75         assert torch.isclose(index_int, index) # TODO: use native pytorch
76         return vector[index_int]
77     else:
78         assert False, 'vector type {} case not implemented'.format(type(vector))
79
80
81 def put_primitive(vector, index, overwritevalue):
82     vector, index, overwritevalue = vector, index, overwritevalue
83
84     if isinstance(index, str):
85         index_safe = index
86     elif torch.is_tensor(index):
87         index_safe = index.item()
88     else:
89         assert False, 'index type {} case not implemented'.format(type(index))
90
91     if isinstance(vector, dict):
92         vector[index_safe] = overwritevalue
93     elif torch.is_tensor(vector):
94         assert not isinstance(index, str)
95         vector[index.long()] = overwritevalue
96     else:
97         assert False, 'vector type {} case not implemented'.format(type(vector))
98     return vector
99
100
101 def return_idx_primitive(vector, idx, idx_f):
102     return vector[0][idx, idx_f]
103
104
105 def first_primitive(vector):
106     return return_idx_primitive(vector, idx=0, idx_f=1)
107
108
109 def second_primitive(vector):
110     return return_idx_primitive(vector, idx=1, idx_f=2)
111
112
113 def last_primitive(vector):
114     return return_idx_primitive(vector, idx=-1, idx_f=None)
115
116
117 def nth_primitive(vector, nth):
118     vector, nth = vector, nth
119     return return_idx_primitive(vector, idx=nth, idx_f=nth+1)
120
121
122 def hash_map_primitive(hash_pairs):
123     keys = hash_pairs[1:-2]
124     for idx, key in enumerate(keys):
125         if torch.is_tensor(key):
126             tensor_key = key
127             keys[idx] = tensor_key.item() # dict keys as tensors problematic. can make but lookup fails
128             # on fresh but equivalent tensor (bc memory look up?)
129             elif isinstance(key, str):
130                 keys[idx] = key # if key string, just keep as is
131
132     # keys = [tensor_key.item() for tensor_key in keys]
133     vals = hash_pairs[1:-2]
134     return dict(zip(keys, vals))
135
136
137 def gt_primitive(consequent_alternative):
138     return two_arg_op_primitive(torch.gt, consequent_alternative)
139
140
141 def lt_primitive(consequent_alternative):
142     return two_arg_op_primitive(torch.lt, consequent_alternative)
143
144
145 def ge_primitive(consequent_alternative):
146     return two_arg_op_primitive(torch.ge, consequent_alternative)
147
148
149 def le_primitive(consequent_alternative):
150     return two_arg_op_primitive(torch.le, consequent_alternative)
151
152
153 def eq_primitive(consequent_alternative):
154     return two_arg_op_primitive(torch.eq, consequent_alternative)
155
156
157 def rest_primitive(vector):
158     return vector[0][1:]
159
160
161 def freshvar_primitive(arg):
162     return None
163
164
165 def vector_primitive(vector):
166     ret = list()
167     for e in vector:
168         try:
169             ret.append(e.tolist())
170         except:
171             ret.append(e)
172     try:
173         return torch.tensor(ret)
174     except:
175         return ret
176
177
178 def append_primitive(vector, element):
179     vector, element = vector, element
180     return torch.cat((vector, torch.Tensor([element])), 0)
181
182
183 def tanh_primitive(arg):
184     return one_arg_op_primitive(torch.tanh, arg)
185
186
187 def and_primitive(arg1, arg2):
188     return two_arg_op_primitive(torch.logical_and, arg1, arg2)
189
190
191 def or_primitive(arg1, arg2):
192     return two_arg_op_primitive(torch.logical_or, arg1, arg2)
193
194
195 def abs_primitive(arg):
196     return one_arg_op_primitive(torch.abs, arg)
197
198
199 def empty_primitive(args):
200     vector = args[0]
201     if torch.is_tensor(vector) or isinstance(vector, list):
202         return len(vector) == 0
203     else:
204         assert False, 'length for non list or non tensor not implemented'
205
206
207 def cons_primitive(args):
208     """https://bfontaine.net/blog/2014/05/25/how-to-remember-the-difference-between-conj-and-cons-in-clojur
209     """
210     item, vector = args
211     if torch.is_tensor(item) and torch.is_tensor(vector):
212         return torch.cat((torch.tensor(item), vector), dim=0)
213     elif isinstance(vector, list):
214         return [item] + vector
215     else:
216         assert False, 'not implemented'
217
218
219 def prepend_primitive(args):
220     """https://bfontaine.net/blog/2014/05/25/how-to-remember-the-difference-between-conj-and-cons-in-clojur
221     """
222     vector, item = args
223     if torch.is_tensor(item) and torch.is_tensor(vector):
224         if item.dim() == 0:
225             item = item.reshape(1,)
226         elif item.dim() == 1:
227             pass
228         else:
229             assert False, 'not implemented'
230     return torch.cat((item, vector), dim=0)
231     elif isinstance(vector, list):
232         return vector + [item]
233     else:
234         assert False, 'not implemented'
235
236
237 def conj_primitive(args):
238     """https://bfontaine.net/blog/2014/05/25/how-to-remember-the-difference-between-conj-and-cons-in-clojur
239     """
240     vector, item = args
241     if torch.is_tensor(item) and torch.is_tensor(vector):
242         assert item.dim() == 0
243         return torch.cat((vector, item.reshape(1,)), dim=0)
244     elif isinstance(vector, list):
245         return vector + [item]
246     else:
247         assert False, 'not implemented'
248
249
250 def log_primitive(arg):
251     return one_arg_op_primitive(torch.log, arg)
252
253
254 def peek_primitive(vector):
255     vector = vector[0]
256     # TODO: assert only defined for vectors
257     return vector[0]
258
259
260 # NB: these functions take a list [c0] or [c0, c1, ..., cn]
261 # rely on user to not write something non-sensical that will fail (e.g. ["+", 1, 2, 3])
262 primitives_d = {
263     '+': add_primitive,
264     '-': subtract_primitive,
265     '/': divide_primitive,
266     '*': multiply_primitive,
267     'sqrt': sqrt_primitive,
268     'vector': vector_primitive,
269     'get': get_primitive,
270     'put': put_primitive,
271     'first': first_primitive,
272     'second': second_primitive,
273     'last': last_primitive,
274     'nth': nth_primitive,
275     'append': append_primitive,
276     'hash-map': hash_map_primitive,
277     '>': gt_primitive,
278     '<': lt_primitive,
279     '>=': ge_primitive,
280     '<=': le_primitive,
281     'rest': rest_primitive,
282     'mat-transpose': lambda a: a[0].T,
283     'mat-tanh': tanh_primitive,
284     'mat-mul': lambda a: torch.matmul(a[0], a[1]),
285     'mat-add': add_primitive,
286     'mat-repeat': lambda a: a[0].repeat((int(a[1].item()), int(a[2].item()))),
287     'and': and_primitive,
288     'or': or_primitive,
289     'abs': abs_primitive,
290     'empty?': empty_primitive,
291     'cons': cons_primitive,
292     'conj': conj_primitive,
293     'log': log_primitive,
294     'peek': peek_primitive,
295     'prepend': prepend_primitive,
296 }
297
298
299 def normal(mean_std):
300     return two_arg_op_primitive(torch.distributions.Normal, mean_std)
301
302
303 def beta(alpha_beta):
304     return two_arg_op_primitive(torch.distributions.Beta, alpha_beta)
305
306
307 def exponential(lam):
308     return one_arg_op_primitive(torch.distributions.Exponential, lam)
309
310
311 def uniform(low_hi):
312     return two_arg_op_primitive(torch.distributions.Uniform, low_hi)
313
314
315 def discrete(prob_vector):
316     return one_arg_op_primitive(torch.distributions.Categorical, prob_vector)
317
318
319 def flip(prob):
320     return one_arg_op_primitive(torch.distributions.bernoulli.Bernoulli, prob)
321
322
323 def dirichlet(concentration):
324     return one_arg_op_primitive(torch.distributions.dirichlet.Dirichlet, concentration)
325
326
327 def gamma(concentration_rate):
328     return two_arg_op_primitive(torch.distributions.gamma.Gamma, concentration_rate)
329
330
331 distributions_d = {
332     'normal': normal,
333     'beta': beta,
334     'exponential': exponential,
335     'uniform-continuous': uniform,
336     'discrete': discrete,
337     'flip': flip,
338     'dirichlet': dirichlet,
339     'gamma': gamma,
340 }
341
342
343 env = {'distributions_d': distributions_d, 'primitives_d': primitives_d}
```