HW2

Geoffrey Woollard

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

Acknowledgments

I acknowledge helpful discussions with Justice Sefas, Masoud Mokhatari, Yuan Tian, Dylan Green, and Jordan Lovrod.

I incorporated code snippets from Yuan Tian and Masoud Mokhatari, and the `vector` primitive from from Yuan Tian was especially helpful. See notes in the source itself.

Pass all units tests

I put this upfront to show that my code passes the tests

In [10]:
```
%%bash
python evaluation_based_sampling.py
```

```
Test 1 passed
Test 2 passed
Test 3 passed
Test 4 passed
Test 5 passed
Test 6 passed
Test 7 passed
Test 8 passed
Test 9 passed
Test 10 passed
Test 11 passed
Test 12 passed
Test 13 passed
All 13 deterministic tests passed
('normal', 5, 1.4142136)
p value 0.8007084502688623
('beta', 2.0, 5.0)
p value 0.570232967759639
('exponential', 0.0, 5.0)
p value 0.11315320065439927
('normal', 5.3, 3.2)
p value 0.9547243874435392
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.7550768772041979
('normal', 0, 1.44)
p value 0.6280392825189822
All probabilistic tests passed



Sample of prior of program 1:
tensor(3.8318)



Sample of prior of program 2:
tensor([-11.5714,  -6.0625])



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



Sample of prior of program 4:
[[[-1.7676916122436523], [-1.118196725845337], [-0.9354032278060913], [0.4580332338809967], [1.04443728923797
6], [-0.4065367579460144], [0.07462392747402191], [-1.051155686378479], [-0.1819164752960205], [-0.801722526550
293]], [[2.6741867065429688], [0.512474536895752], [-1.0142426490783691], [0.1489662528038025], [-0.22332051396
369934], [0.4326988458633423], [0.24860680103302002], [0.9658144116401672], [-0.41809406876564026], [-0.5750256
180763245]], [[0.7798222899436951, -0.42983824014663696, -1.0312782526016235, -0.1406644582748413, -0.899638652
8015137, -0.5114136338233948, 0.3236844837665558, -0.855675220489502, 1.986088514328003, 0.45031386613845825],
[0.7211995124816895, 0.21626058220863342, 1.1248375177383423, 1.2797229290008545, -1.5293806791305542, 1.641720
6525802612, -0.9853824377059937, 1.2712563276290894, 0.0842912346124649, -0.12786734104156494], [-0.05732435360
5508804, -0.3076135516166687, -0.22628335654735565, 0.308715283870697, 0.9706620573997498, 0.6126998066902161,
1.466538906097412, 0.5463335514068604, -0.6245884299278259, 0.36614343523979187], [2.1768877506256104, 0.566818
4161186218, -0.3649443984031677, -0.484836608171463, -0.5604891777038574, -0.1092633306980133, -0.8711755871772
766, -0.9893564581871033, -2.622530460357666, -0.19729599356651306], [0.5502117872238159, 0.6832883358001709,
0.521416425704956, 0.9656248092651367, -1.3003575801849365, -0.9540558457374573, -0.17749276757240295, -1.07263
93461227417, 0.3236338794231415, -1.5408300161361694], [0.43400198221206665, 2.2279958724975586, 0.632498264312
7441, 0.6176543831825256, -0.803342878818512, -0.9014406204223633, -0.4077558219432831, -0.4219468832015991, 0.
6329726576805115, 0.4033464789390564], [0.925809919834137, 1.1987119913101196, 0.4856245815753937, 1.4616057872
772217, 0.8021788597106934, 0.7660475969314575, 0.42356815934181213, -1.2062325477600098, -1.037696123123169, -
0.32846102118492126], [-0.8269684910774231, -0.16437765955924988, 1.0802171230316162, -0.11656741052865982, -1.
0195673704147339, -0.6772551536560059, -0.2532135844230652, -0.04122402146458626, -5.8011133660329506e-05, 0.90
99297523498535], [-0.6462174654006958, 2.7595667839050293, -0.07100299745798111, 1.001941204071045, -0.37857598
066329956, -0.389217346906662, -0.409883975982666, -0.18588533997535706, 1.079423189163208, -0.763119399547576
9], [-0.05784069374203682, 1.102362871170044, -1.8171186447143555, -0.1802964061498642, 1.4344890117645264, 0.5
055438876152039, 0.7119135856628418, -1.6287842988967896, 0.3141670525074005, 0.12383020669221878]], [[-1.50190
57989120483], [0.5333874225616455], [1.367606282234192], [-0.8230386972427368], [0.3667728304862976], [-0.17649
102210998535], [-0.9033983945846558], [-0.6441863179206848], [0.3718051016330719], [0.5993713140487671]]]
```

In [1]:
```
%%bash
python graph_based_sampling.py
```

```
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
All 12 deterministic tests passed
('normal', 5, 1.4142136)
p value 0.9750933029255907
('beta', 2.0, 5.0)
p value 0.19567166298785177
('exponential', 0.0, 5.0)
p value 0.5077823820542168
('normal', 5.3, 3.2)
p value 0.03229248986313449
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.22134508762391436
('normal', 0, 1.44)
p value 0.06801567419824306
All probabilistic tests passed



Sample of prior of program 1:
tensor(-1.9531)



Sample of prior of program 2:
tensor([ 0.0142, -2.6072])



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



Sample of prior of program 4:
[[[0.3404915928840637], [0.07253124564886093], [1.9410412311553955], [1.595449686050415], [0.7876811027526855],
[0.7403945922851562], [-0.7994391322135925], [0.1699640154838562], [0.8591791987419128], [-0.4378859996795654
3]], [[-0.1290707141160965], [0.9287136793136597], [0.13374224305152893], [-0.7232226729393005], [-0.1657858937
9787445], [0.5664358735084534], [1.3867830038070679], [1.5518771409988403], [-1.3518511056900024], [3.018144130
706787]], [[1.2903233766555786, -0.7906650304794312, 0.26601752638816833, 0.7309240698814392, -0.62711685895919
8, 1.0325922966003418, -1.8757492303848267, -1.059775710105896, -1.3485506772994995, -0.49490001797676086], [-
0.9938995838165283, -0.5072317123413086, -0.19611066579818726, 1.3307677507400513, -1.315219521522522, 0.798445
3439712524, 0.07256823033094406, -0.5548135638237, -0.007056010887026787, -0.012109244242310524], [0.4829832613
46817, -0.4422144293785095, -0.7381874918937683, -2.085496187210083, 0.12616091966629028, 0.3482881188392639,
0.867943286895752, 0.5284357666969299, -0.9863168597221375, 0.31645524501800537], [1.0765364170074463, -1.17782
78350830078, 0.15887458622455597, -2.273386001586914, 0.49036675691604614, 0.3065895736217499, 1.85581767559051
51, 0.14977042376995087, -0.14015142619609833, -1.1439459323883057], [0.8989789485931396, 0.7787042856216431, -
0.8660594820976257, -1.717496633529663, 1.5413812398910522, 0.36692044138908386, -0.765255868434906, -1.9025458
097457886, 1.4255683422088623, 1.1816129684448242], [1.1142038106918335, -1.6864469051361084, 0.320096850395202
64, -0.3467515707015991, -0.8627139925956726, -0.6969993114471436, 0.8607492446899414, 0.6708496809005737, 0.01
2494700029492378, 0.5289558172225952], [1.3242874145507812, 0.02315308153629303, -1.1985429525375366, -0.571601
0928153992, -1.2426120042800903, -1.1988112926483154, 0.8473421931266785, -1.5871920585632324, -1.7627055644989
014, 0.0475890152156353], [0.9945656061172485, 1.7476205825805664, -1.2077726125717163, 1.4424941539764404, -0.
5761687159538269, 1.848507285118103, 0.6472342014312744, -0.6800799369812012, -0.1219860091805458, -0.623577117
9199219], [0.6625446677207947, 1.2416499853134155, 1.3970935344696045, 0.021771788597106934, -0.207233056426048
28, -2.1815247535705566, 1.529159426689148, -0.027197113260626793, -1.6421653032302856, 1.9766781330108643],
[0.6615936160087585, 1.0952072143554688, -2.07956862449646, -0.33996936678886414, 0.9241219758987427, 0.3169116
675853729, 0.6953616142272949, 0.020686496049165726, 1.178383231163025, 1.8796063661575317]], [[-1.675215601921
0815], [0.5269629955291748], [-1.063934087753296], [2.432159185409546], [0.39528951048851013], [1.5675228834152
222], [0.24734075367450714], [1.1506731510162354], [-0.1022210642695427], [-1.1905789375305176]]]
```

code snippets

In [4]:
```
from dill.source import getsource, getsourcelines
```

`evaluation_based_sampling`

Algorithm 6 evaluator

- I called it `evaluator` and not `eval`, because `eval` is already taken in Python
- This recursive function works by evaluating back to a few base cases. These can be seen because they return things directly without calling the `evaluate` recursively
  - `return torch.tensor(float(e))` in the case of `(float,int)`
  - `return e` in the cases of a `list`, primitive operations `primitives_d`, distributions `distributions_d`, tensor,
  - `return local_env[e]` (ie the bound variable) in the case of a key in the local context `local_env`
  - `return distributions_d[cs[0]](cs[1:])` in the case of `cs[0]` being a distribution. ie return the primitive type of distribution
  - `return local_env[e ]` in the case of something in the local context `local_env`
  - `return primitives_d[cs[0]](cs[1:])`, ie do the primitive operation

In [8]:
```
from evaluation_based_sampling import evaluate
```

In [9]:
```
for line_number, function_line in enumerate(getsourcelines(evaluate)[0]):
    print(line_number, function_line,end='')
```

```
0 def evaluate(e,local_env={},defn_d={},do_log=False):
1     # TODO: get local_env to evaluate values to tensors, not regular floats
2     # remember to return evaluate (recursive)
3     # everytime we call evaluate, we have to use local_env, otherwise it gets overwritten with the default {}
4     if do_log: logger.info('local_env {}'.format(local_env))
5     # get first expression out of list or list of one
6     if not isinstance(e,list) or len(e) == 1:
7         if isinstance(e,list):
8             e = e[0]
9 
10         if isinstance(e, number):
11             if do_log: logger.info('match case: number {}'.format(e))
12             return torch.tensor(float(e))
13         elif isinstance(e,list):
14             if do_log: logger.info('match case: list {}'.format(e))
15             return e
16         elif e in list(primitives_d.keys()):
17             if do_log: logger.info('match case: primitives_d {}'.format(e))
18             return e
19         elif e in list(distributions_d.keys()):
20             if do_log: logger.info('match case: distributions_d {}'.format(e))
21             return e
22         elif torch.is_tensor(e) and not len(list(e.shape)) == 0:
23             if do_log: logger.info('match case: is_tensor {}'.format(e))
24             return e
25         elif e in local_env.keys():
26             if do_log: logger.info('match case: local_env, e {}'.format(e))
27             if do_log: logger.info('match case: local_env, local_env[e] {}'.format(local_env[e]))
28             return local_env[e] # TODO return evaluate?
29         elif e in list(defn_d.keys()):
30             if do_log: logger.info('match case: defn_d {}'.format(e))
31             return e
32         elif isinstance(e,distribution_types):
33             if do_log: logger.info('match case: distribution {}'.format(e))
34             return e
35         else:
36             assert False
37     elif e[0] == 'sample':
38         if do_log: logger.info('match case: sample {}'.format(e))
39         distribution = evaluate(e[1],local_env,defn_d,do_log=do_log)
40         return distribution.sample() # match shape in number base case
41     elif e[0] == 'observe':
42         return None
43     elif e[0] == 'let':
44         if do_log: logger.info('match case: let {}'.format(e))
45         # let [v1 e1] e0
46         # here
47             # e[0] : "let"
48             # e[1] : [v1, e1]
49             # e[2] : e0
50         # evaluates e1 to c1 and binds this value to e0
51         # this means we update the context with old context plus {v1:c1}
52         c1 = evaluate(e[1][1],local_env,defn_d,do_log=do_log) # evaluates e1 to c1
53         v1 = e[1][0]
54         return evaluate(e[2], local_env = {**local_env,v1:c1},defn_d=defn_d,do_log=do_log)
55     elif e[0] == 'if': # if e0 e1 e2
56         if do_log: logger.info('match case: if {}'.format(e))
57         e0 = e[1]
58         e1 = e[2]
59         e2 = e[3]
60         if evaluate(e0,local_env,defn_d,do_log=do_log):
61             return evaluate(e1,local_env,defn_d,do_log=do_log)
62         else:
63             return evaluate(e2,local_env,defn_d,do_log=do_log)
64 
65     else:
66         cs = []
67         for ei in e:
68             if do_log: logger.info('cycling through expressions. ei {}'.format(ei))
69             c = evaluate(ei,local_env,defn_d,do_log=do_log)
70             cs.append(c)
71         if cs[0] in primitives_d:
72             if do_log: logger.info('do case primitives_d: cs0 {}'.format(cs[0]))
73             if do_log: logger.info('do case primitives_d: cs1 {}'.format(cs[1:]))
74             if do_log: logger.info('do case primitives_d: primitives_d[cs[0]] {}'.format(primitives_d[cs
[0]]))
75             return primitives_d[cs[0]](cs[1:])
76         elif cs[0] in distributions_d:
77             if do_log: logger.info('do case distributions_d: cs0 {}'.format(cs[0]))
78             return distributions_d[cs[0]](cs[1:])
79         elif cs[0] in defn_d:
80             if do_log: logger.info('do case defn: cs0  {}'.format(cs[0]))
81             defn_function_li = defn_d[cs[0]]
82             defn_function_args, defn_function_body = defn_function_li
83             local_env_update = {key:value for key,value in zip(defn_function_args, cs[1:])}
84             if do_log: logger.info('do case defn: update to local_env from defn_d {}'.format(local_env_updat
e))
85             return evaluate(defn_function_body,local_env = {**local_env, **local_env_update},defn_d=defn_d,d
o_log=do_log)
86         else:
87             assert False
```

For the primitives and distributions, I made the design choice that they would take in a parsed `list`, `cs[1:]`. I used the name of the input list of the function as documentation of what is expected, e.g. `normal(mean_std)`. I also modularized some parsing in `one_arg_op_primitive` and `two_arg_op_primitive`, and used these as internal helper functions.

One challenge I encountered at the very end of the assignment was generalizing my `vector` primitive to work properly to construct multidimensional arrays. I was getting shape mismatch errors in the Neural Network example `4.daphne`, and try as I might, I couldn't write a `vector` primitive that could properly get all the different cases working. Sometimes `vector` yielded a list of distributions. Sometimes a `torch.tensor([float_1,...,float_n])` of shape `(n,)`, or a tensor of what `(n,1)` other way: `torch.tensor([[float_1],...,[float_n]])`. It got to the point where I had a vector primitive that was dozens of cases, and very hacky, and uninterpretable. The root cause of all this pain was that I had made the design choice early on to return `torch.tensor.([float])` instead of `torch.tensor.(float)`. Big mistake. As soon as I changed this, I could write a much simpler `vector` primitive to break into a few cases of returning a `list` (e.g. of distributions), `torch.tensor` and `torch.stack`, and things came together.

I had originally made this choice to get `append` working with `torch.cat`, because its second argument needs to be `torch.tensor.([float])` not `torch.tensor.(float)`. So I just needed to include the append `primitive` to include this case by reworking the second argument.

I trouble shooted by running `evaluate(parsed_4_daphne,do_log=True)` and looking at the local context, noting shapes, and seeing what was causing things to fail. The evaluation based sampling is interpretable step by step, and the shapes of what things should be was seen from the code (e.g. vector of 1x1 vectors, so column vector instead of row vector). Lesson: read ahead a bit to how primitives are used, because making design choices that get burned into the code base.

Primitives

In [6]:
```
from primitives import primitives_d
for key in primitives_d.keys() :
    print(key,':')
    for line_number, function_line in enumerate(getsourcelines(primitives_d[key])[0]):
        print(line_number, function_line,end='')
    print()
```

```
+ :
0 def add_primitive(arg1_arg2):
1     return two_arg_op_primitive(torch.add,arg1_arg2)

- :
0 def subtract_primitive(arg1_arg2):
1     return two_arg_op_primitive(torch.subtract,arg1_arg2)

/ :
0 def divide_primitive(arg1_arg2):
1     return two_arg_op_primitive(torch.divide,arg1_arg2)

* :
0 def multiply_primitive(arg1_arg2):
1     return two_arg_op_primitive(torch.multiply,arg1_arg2)

sqrt :
0 def sqrt_primitive(arg):
1     return one_arg_op_primitive(torch.sqrt,arg)

vector :
0 def vector_primitive(vector):
1     ret = list()
2     for e in vector:
3         try:
4             ret.append(e.tolist())
5         except:
6             ret.append(e)
7     try:
8         return torch.FloatTensor(ret)
9     except:
10         return ret

get :
0 def get_primitive(vector_and_index):
1     vector, index = vector_and_index
2     if isinstance(vector,dict):
3         return vector[index.item()]
4     elif torch.is_tensor(vector):
5         return vector[index.long()]
6     elif isinstance(vector,list):
7         index_int = int(index)
8         assert np.isclose(index_int,index) # TODO: use native pytorch
9         return vector[index_int]
10     else:
11         assert False,  'vector type {} case not implemented'.format(type(vector))

put :
0 def put_primitive(vector_index_overwritevalue):
1     vector, index, overwritevalue = vector_index_overwritevalue
2     if isinstance(vector,dict):
3         vector[index.item()] = overwritevalue
4     elif torch.is_tensor(vector):
5         vector[index.long()] = overwritevalue
6     else:
7         assert False,  'vector type {} case not implemented'.format(type(vector))
8     return vector

first :
0 def first_primitive(vector):
1     return return_idx_primitive(vector,idx_i=0,idx_f=1)

second :
0 def second_primitive(vector):
1     return return_idx_primitive(vector,idx_i=1,idx_f=2)

last :
0 def last_primitive(vector):
1     return return_idx_primitive(vector,idx_i=-1,idx_f=None)

nth :
0 def nth_primitive(vector_nth):
1     vector, nth = vector_nth
2     return return_idx_primitive(vector,idx_i=nth,idx_f=nth+1)

append :
0 def append_primitive(vector_element):
1         vector, element = vector_element
2         # arg2 must be torch.tensor([float]), not torch.tensor(float) otherwise torch.cat fails
3         return torch.cat((vector,torch.Tensor([element])), 0)

hash-map :
0 def hash_map_primitive(hash_pairs):
1     keys = hash_pairs[::2]
2     # dict keys as tensors problematic. can make but lookup fails on fresh but equivalent tensor (bc memory l
ook up?)
3     keys = [tensor_key.item() for tensor_key in keys]
4     vals = hash_pairs[1::2]
5     return dict(zip(keys, vals))

> :
0 def gt_primitive(consequent_alternative):
1     return two_arg_op_primitive(torch.gt,consequent_alternative)

< :
0 def lt_primitive(consequent_alternative):
1     return two_arg_op_primitive(torch.lt,consequent_alternative)

>= :
0 def ge_primitive(consequent_alternative):
1     return two_arg_op_primitive(torch.ge,consequent_alternative)

<= :
0 def le_primitive(consequent_alternative):
1     return two_arg_op_primitive(torch.le,consequent_alternative)

== :
0 def eq_primitive(consequent_alternative):
1     return two_arg_op_primitive(torch.eq,consequent_alternative)

rest :
0 def rest_primative(vector):
1     return vector[0][1:]

mat-transpose :
0     'mat-transpose': lambda a: a[0].T,

mat-tanh :
0 def tanh_primitive(arg):
1     return one_arg_op_primitive(torch.tanh,arg)

mat-mul :
0     'mat-mul': lambda a: torch.matmul(a[0],a[1]),

mat-add :
0 def add_primitive(arg1_arg2):
1     return two_arg_op_primitive(torch.add,arg1_arg2)

mat-repmat :
0     'mat-repmat': lambda a: a[0].repeat((int(a[1].item()), int(a[2].item()))),
```

Distributions

I separated out the primitive operations from the distributions. This helped debug while developing.

In [500…
```
from primitives import distributions_d

for key in distributions_d.keys() :
    print(key,':')
    for line_number, function_line in enumerate(getsourcelines(distributions_d[key])[0]):
        print(line_number, function_line,end='')
    print()
```

```
normal :
0 def tanh_primitive(arg):
1     return one_arg_op_primitive(torch.tanh,arg)

beta :
0     'mat-mul': lambda a: torch.matmul(a[0],a[1]),

exponential :
0     'mat-repmat': lambda a: a[0].repeat((int(a[1].item()), int(a[2].item()))),

uniform :
0 def normal(mean_std):
1     return two_arg_op_primitive(torch.distributions.Normal,mean_std)

discrete :
0 def beta(alpha_beta):
1     return two_arg_op_primitive(torch.distributions.Beta,alpha_beta)
```

`defn` and `evaluate_program`

I implemented the defn in `evaluate_program`, which is a layer above the recursive `evaluate`, beause the programs had the functional definitions at the beginning of the program. I simply parsed this in `evaluate_program`, and made a binding of the defn function name string to the args and expression body : `defn_d[defn_function_name] = [defn_function_args,defn_function_body]`.

In [5]:
```
from evaluation_based_sampling import evaluate_program
for line_number, function_line in enumerate(getsourcelines(evaluate_program)[0]):
    print(line_number, function_line,end='')
```

```
0 def evaluate_program(ast,sig=None,do_log=False):
1     """Evaluate a program as desugared by daphne, generate a sample from the prior
2     Args:
3         ast: json FOPPL program
4     Returns: sample from the prior of ast
5     """
6     defn_d ={}
7     ast0 = ast[0]
8 
9     if ast0[0] == 'defn':
10         defn_function_name = ast0[1]
11         defn_function_args = ast0[2]
12         defn_function_body = ast0[3]
13         defn_d[defn_function_name] = [defn_function_args,defn_function_body]
14         ast1 = ast[1]
15         res = evaluate(ast1,defn_d=defn_d,do_log=do_log)
16     elif len(ast) == 1:
17         res = evaluate(ast0,defn_d=defn_d,do_log=do_log)
18     else:
19         assert False
20     return res, sig
```

The `evaluate` function handles `denf` s by updating the local context dictionary to bind the args to the evaluation of the args (eventually reduces to constants) that were passed to it when it was called. The return is the evaluation of the function body, under the updated local context.

There are some logger functionality to help debugging, and also `assert False` to catch unforseen cases.

I added in some functinoality to write the daphne parsed abstract syntax trees and graphs to jsons, and check if they exist. This avoids re-doing parsing redundantly, and speeds up the development cycle.

`graph_based_sampling`

The `deterministic_eval` in `graph_based_sampling` is using the exact same `evaluate` helper function as in `evalute_based_sampling`

In [501…
```
from graph_based_sampling import deterministic_eval
print(getsource(deterministic_eval))
```

```
def deterministic_eval(exp):
    return evaluate(exp)
```

The ancestral sampling is done with `sample_from_joint`, which parses the daphne output json graph structure, and then traverses the graph, calling the `evaluate` internally on the expressions.

The ancestral part is ensured by

- performing a topological sort on the verteces
  - using the information of the arcs to construct a graph, and then using a topological sort routine I found online, and slighlt massaged to work in this case.
- walking thorugh each top sorted vertex (the ancestral part in ancestral sampling)
- evaluating the expressions as they arise into a distribution
- taking a sample (the sampling part in ancestral sampling), which evaluates to a constant
- updating the local context of the evaluator with this sampling variable name bound to its sampled constant value, and
- passing by observes and doing nothing
- At the end of the graph traversal, the local context bindings we have accumulated have all the information we need for the evaluaton of the expression in the return / meaning of the program. We simply call `evaluate` one more time with these bindings used in the local context (they only have sample variables in them).

In [502…
```
from graph_based_sampling import sample_from_joint
for line_number, function_line in enumerate(getsourcelines(sample_from_joint)[0]):
    print(line_number, function_line,end='')
```

```
0 def sample_from_joint(graph,do_log=False):
1     """This function does ancestral sampling starting from the prior.
2 
3     graph output from `daphne graph -i sugared.daphne`
4     * list of length 3
5       * first entry is defn dict
6         * {"string-defn-function-name":["fn", ["var_1", ..., "var_n"], e_function_body], ...}
7       * second entry is graph: {V,A,P,Y}
8         * "V","A","P","Y" are keys in dict
9         * "V" : ["string_name_vertex_1", ..., "string_name_vertex_n"] # list of string names of vertices
10          * "A" : {"string_name_vertex_1" : [..., "string_name_vertex_i", ...] # dict of arc pairs (u,v) with
u a string key in the dict, and the value a list of string names of the vertices. note that the keys can be thi
ngs like "uniform" and don't have to be vetex name strings
11          * "P" : "string_name_vertex_i" : ["sample*", e_i] # dict. keys vertex name strings and value a reste
d list with a linking function in it. typically e_i is a distribution object.
12          * "Y" : observes
13       * third entry is return
14         * name of return rv, or constant
15 
16     """
17     G = graph[1]
18     verteces = G['V']
19     arcs = G['A']
20     verteces_topsorted = topsort(verteces, arcs)
21     P = G['P']
22     Y = G['Y']
23     sampled_graph = {}
24     local_env = {}
25     for vertex in verteces_topsorted:
26         link_function = P[vertex]
27         if link_function[0] == 'sample*':
28             assert len(link_function) == 2
29             e = link_function[1]
30     #         print('e in as',e)
31             distribution = evaluate(e,local_env = local_env, do_log=do_log)
32             E = distribution.sample() # now have concrete value. need to pass it as var to evaluate
33             update_local_env = {vertex:E}
34             local_env.update(update_local_env)
35         elif link_function[0] == 'observe*':
36             pass
37             # assert len(link_function) == 3
38             # E = Y[vertex]
39         else:
40             assert False
41         # sampled_graph[vertex] = E
42     sampled_graph = local_env
43     return_of_graph = graph[2] # meaning of program, but need to evaluate
44     # if do_log: print('sample_from_joint local_env',local_env)
45     # if do_log: print('sample_from_joint sampled_graph',sampled_graph)
46     return evaluate(return_of_graph,local_env = sampled_graph, do_log=do_log)
```

Programs 1-4

1000 sample marginal plots and expectations

I understand this question to be asking to run the 1.daphne ... 4.daphne programs 1000 times, and then look at the distribution of each quantity independently. So if a program returned two tensors, one of size (a,b,c) and the other of size (x,y,z), we would plot the 6 1-dimensional distributions, for (a,b,c,x,y,z), along with the 6 expectations of each of these distributions. The term "marginal" makes sense in this context, because its independent of what the other dimensions were doing. For this reason it's a necessary condition, but not sufficient for corectness.

The evaluation based vs graph based results are shown beside each other

In [308…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from graph_based_sampling import get_stream as get_stream_graph
from graph_based_sampling import sample_from_joint

from evaluation_based_sampling import get_stream as get_stream_evaluation
from evaluation_based_sampling import evaluate_program
```

In [249…
```
import os, json
from daphne import daphne

def load_graph(i):
    os.chdir('/Users/gw/repos/prob_prog/hw/hw2/CS532-HW2/')
    sugared_fname = '../prob_prog/hw/hw2/CS532-HW2/programs/{}.daphne'.format(i)
    graph_json_fname = '/Users/gw/repos/prob_prog/' + sugared_fname.replace('.daphne','_graph.json')
    if os.path.isfile(graph_json_fname):
        with open(graph_json_fname) as f:
            graph = json.load(f)
    else:
        #note: the sugared path that goes into daphne desugar should be with respect to the daphne path!
        graph = daphne(['graph', '-i', sugared_fname])
        with open(graph_json_fname, 'w') as f:
            json.dump(graph, f)
    return graph

def load_ast(i):
    os.chdir('/Users/gw/repos/prob_prog/hw/hw2/CS532-HW2/')
    sugared_fname = '../prob_prog/hw/hw2/CS532-HW2/programs/{}.daphne'.format(i)
    desugared_ast_json_fname = '/Users/gw/repos/prob_prog/' + sugared_fname.replace('.daphne','.json')
    if os.path.isfile(desugared_ast_json_fname):
        with open(desugared_ast_json_fname) as f:
            ast = json.load(f)
    else:
        #note: the sugared path that goes into daphne desugar should be with respect to the daphne path!
        ast = daphne(['desugar', '-i', sugared_fname])
        with open(desugared_ast_json_fname, 'w') as f:
            json.dump(ast, f)
    return ast
```

`1.daphne`: Gaussian unknown mean

Prior of `mu` given by $\mathcal{N}[\mu=1, \sigma=\sqrt{5}]$

Nota Bene: `mu` and μ, actually refer to something different here. See the source code of the problem for how the mean of a normal is itself normally distributed.

In [457…
```
graph = load_graph(i=1)
graph[-1]
```

Out[457…
```
'sample2'
```

In [458…
```
ast = load_ast(i=1)
ast
```

Out[458…
```
[['let',
  ['mu', ['sample', ['normal', 1, ['sqrt', 5]]]],
  ['let',
   ['sigma', ['sqrt', 2]],
   ['let',
    ['lik', ['normal', 'mu', 'sigma']],
    ['let',
     ['dontcare0', ['observe', 'lik', 8]],
     ['let', ['dontcare1', ['observe', 'lik', 9]], 'mu']]]]]]
```

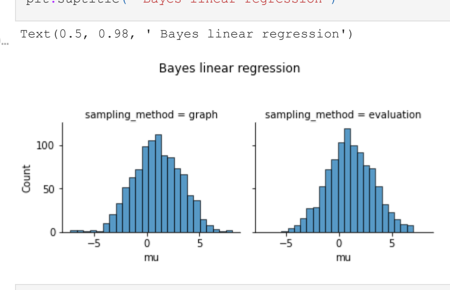
In [459…
```
graph_samples,evaluation_samples=[],[]
num_samples=1000
for _ in range(num_samples):
    graph_samples.append(next(get_stream_graph(graph)))
    evaluation_samples.append(next(get_stream_evaluation(ast)))


df = pd.DataFrame({'graph':graph_samples,'evaluation':evaluation_samples} )
df = pd.melt(df).rename(columns={'variable':'sampling_method','value':'mu'})
```

In [460…
```
g = sns.FacetGrid(df, col="sampling_method")
g.map(sns.histplot, "mu")

plt.subplots_adjust(top=0.7)
plt.suptitle(' Bayes linear regression')
```

Out[460…
```
Text(0.5, 0.98, ' Bayes linear regression')
```



In [462…
```
print(df.groupby('sampling_method').mean().rename(columns={'mu':'expectation_mu'}).to_string())
```

```
                 expectation_mu
sampling_method
evaluation             1.050941
graph                  0.963071
```

In [465…
```
print(df.groupby('sampling_method').var().rename(columns={'mu':'var_mu'}).to_string())
```

```
                   var_mu
sampling_method
evaluation       4.776112
graph            5.112996
```

The sample mean and var of `mu` are near their expected values

`2.daphne` Bayesian Linear Regression

returns slope and bias in a 2-vector

`slope` $\sim \mathcal{N}[\mu=0, \sigma=10]$

`bias` $\sim \mathcal{N}[\mu=0, \sigma=10]$

In [436…
```
graph = load_graph(i=2)
graph[-1]
```

Out[436…
```
['vector', 'sample1', 'sample2']
```

In [437…
```
ast = load_ast(i=2)
ast
```

Out[437…
```
[['defn',
  'observe-data',
  ['_', 'data', 'slope', 'bias'],
  ['let',
   ['xn', ['first', 'data']],
   ['let',
    ['yn', ['second', 'data']],
    ['let',
     ['zn', ['+', ['*', 'slope', 'xn'], 'bias']],
     ['let',
      ['dontcare0', ['observe', ['normal', 'zn', 1.0], 'yn']],
      ['rest', ['rest', 'data']]]]]]],
 ['let',
  ['slope', ['sample', ['normal', 0.0, 10.0]]],
  ['let',
   ['bias', ['sample', ['normal', 0.0, 10.0]]],
   ['let',
    ['data',
     ['vector', 1.0, 2.1, 2.0, 3.9, 3.0, 5.3, 4.0, 7.7, 5.0, 10.2, 6.0, 12.9]],
    ['let',
     ['dontcare1',
      ['let',
       ['a2', 'slope'],
       ['let',
        ['a3', 'bias'],
        ['let',
         ['acc4', ['observe-data', 0, 'data', 'a2', 'a3']],
         ['let',
          ['acc5', ['observe-data', 1, 'acc4', 'a2', 'a3']],
          ['let',
           ['acc6', ['observe-data', 2, 'acc5', 'a2', 'a3']],
           ['let',
            ['acc7', ['observe-data', 3, 'acc6', 'a2', 'a3']],
            ['let',
             ['acc8', ['observe-data', 4, 'acc7', 'a2', 'a3']],
             ['let',
              ['acc9', ['observe-data', 5, 'acc8', 'a2', 'a3']],
              'acc9']]]]]]]],
     ['vector', 'slope', 'bias']]]]]]
```

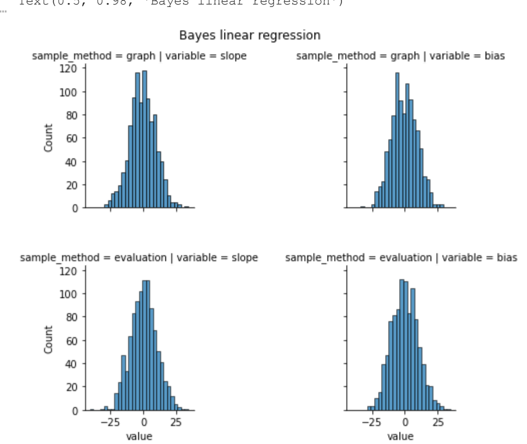
In [438…
```
graph_samples,evaluation_samples=[],[]
num_samples=1000
for _ in range(num_samples):
    graph_samples.append(sample_from_joint(graph).tolist())
    evaluation_samples.append(evaluate_program(ast)[0].tolist())
```

In [439…
```
df_list=[]
for samples, sample_method in zip([graph_samples,evaluation_samples],['graph','evaluation']):
    df = pd.DataFrame(samples)
    df = pd.melt(df)
    df['variable'] = df['variable'].map({0:'slope',1:'bias'})
    df['sample_method'] = sample_method
    df_list.append(df)
df = pd.concat(df_list)
```

In [440…
```
g = sns.FacetGrid(df, col="variable",row='sample_method')
g.map(sns.histplot, "value")
plt.subplots_adjust(hspace=0.4, wspace=1.4)

plt.subplots_adjust(top=0.9)
plt.suptitle('Bayes linear regression')
```

Out[440… Text(0.5, 0.98, 'Bayes linear regression')



In [442…

```python
print(df.groupby(['variable','sample_method']).mean().rename(columns={'value':'expectation'}).to_string())
```

```
                        expectation
variable sample_method
bias     evaluation       -0.290544
         graph            -0.625519
slope    evaluation        0.072113
         graph            -0.510666
```

In [446…

```python
print(df.groupby(['variable','sample_method']).std().rename(columns={'value':'std'}).to_string())
```

```
                              std
variable sample_method
bias     evaluation     10.024271
         graph           9.693775
slope    evaluation     10.013176
         graph           9.825065
```

The sample mean and var of the bias and slope are near their expected values

`3.daphne` Hidden Markov Model

return is vector of length 17, from 16 steps after the initial distribution.

In [466…

```python
graph = load_graph(i=3)
graph[-1]
```

Out[466…

```
['vector',
 'sample0',
 'sample1',
 'sample3',
 'sample5',
 'sample7',
 'sample9',
 'sample11',
 'sample13',
 'sample15',
 'sample17',
 'sample19',
 'sample21',
 'sample23',
 'sample25',
 'sample27',
 'sample29',
 'sample31']
```

In [467…

```python
ast = load_ast(i=3)
ast
```

Out[467…

```
[['defn',
  'hmm-step',
  ['t', 'states', 'data', 'trans-dists', 'likes'],
  ['let',
   ['z', ['sample', ['get', 'trans-dists', ['last', 'states']]]],
   ['let',
    ['dontcare0', ['observe', ['get', 'likes', 'z'], ['get', 'data', 't']]],
    ['append', 'states', 'z']]]],
 ['let',
  ['data',
   ['vector',
    0.9,
    0.8,
    0.7,
    0.0,
    -0.025,
    -5.0,
    -2.0,
    -0.1,
    0.0,
    0.13,
    0.45,
    6,
    0.2,
    0.3,
    -1,
    -1]],
  ['let',
   ['trans-dists',
    ['vector',
     ['discrete', ['vector', 0.1, 0.5, 0.4]],
     ['discrete', ['vector', 0.2, 0.2, 0.6]],
     ['discrete', ['vector', 0.15, 0.15, 0.7]]]],
   ['let',
    ['likes',
     ['vector',
      ['normal', -1.0, 1.0],
      ['normal', 1.0, 1.0],
      ['normal', 0.0, 1.0]]],
    ['let',
     ['states',
      ['vector', ['sample', ['discrete', ['vector', 0.33, 0.33, 0.34]]]]],
     ['let',
      ['a1', 'data'],
      ['let',
       ['a2', 'trans-dists'],
       ['let',
        ['a3', 'likes'],
        ['let',
         ['acc4', ['hmm-step', 0, 'states', 'a1', 'a2', 'a3']],
         ['let',
          ['acc5', ['hmm-step', 1, 'acc4', 'a1', 'a2', 'a3']],
          ['let',
           ['acc6', ['hmm-step', 2, 'acc5', 'a1', 'a2', 'a3']],
           ['let',
            ['acc7', ['hmm-step', 3, 'acc6', 'a1', 'a2', 'a3']],
            ['let',
             ['acc8', ['hmm-step', 4, 'acc7', 'a1', 'a2', 'a3']],
             ['let',
              ['acc9', ['hmm-step', 5, 'acc8', 'a1', 'a2', 'a3']],
              ['let',
               ['acc10', ['hmm-step', 6, 'acc9', 'a1', 'a2', 'a3']],
               ['let',
                ['acc11', ['hmm-step', 7, 'acc10', 'a1', 'a2', 'a3']],
                ['let',
                 ['acc12', ['hmm-step', 8, 'acc11', 'a1', 'a2', 'a3']],
                 ['let',
                  ['acc13', ['hmm-step', 9, 'acc12', 'a1', 'a2', 'a3']],
                  ['let',
                   ['acc14', ['hmm-step', 10, 'acc13', 'a1', 'a2', 'a3']],
                   ['let',
                    ['acc15', ['hmm-step', 11, 'acc14', 'a1', 'a2', 'a3']],
                    ['let',
                     ['acc16', ['hmm-step', 12, 'acc15', 'a1', 'a2', 'a3']],
                     ['let',
                      ['acc17', ['hmm-step', 13, 'acc16', 'a1', 'a2', 'a3']],
                      ['let',
                       ['acc18', ['hmm-step', 14, 'acc17', 'a1', 'a2', 'a3']],
                       ['let',
                        ['acc19', ['hmm-step', 15, 'acc18', 'a1', 'a2', 'a3']],
                        'acc19']]]]]]]]]]]]]]]]]]]]]]]
```

In [468…

```python
graph_samples,evaluation_samples=[],[]
num_samples=1000
for _ in range(num_samples):
    graph_samples.append(sample_from_joint(graph).tolist())
    evaluation_samples.append(evaluate_program(ast)[0].tolist())
```

In [469…

```python
df_list=[]
for samples, sample_method in zip([graph_samples,evaluation_samples],['graph','evaluation']):
    df = pd.DataFrame(samples)
    df = pd.melt(df)
    df['sample_method'] = sample_method
    df_list.append(df)
df = pd.concat(df_list)
```

In [470…

```python
g = sns.FacetGrid(df, row="variable",col='sample_method')
g.map(sns.histplot, "value")
plt.subplots_adjust(wspace=1)

plt.subplots_adjust(top=.97)
plt.suptitle('HMM')
```

Out[470… Text(0.5, 0.98, 'HMM')

In [471…

```python
print(df.groupby(['variable','sample_method']).mean().rename(columns={'value':'expectation'}).to_string())
```

```
                        expectation
variable sample_method
0        evaluation           1.054
         graph                1.026
1        evaluation           1.390
         graph                1.429
2        evaluation           1.466
         graph                1.452
3        evaluation           1.453
         graph                1.450
4        evaluation           1.446
         graph                1.451
5        evaluation           1.472
         graph                1.441
6        evaluation           1.461
         graph                1.467
7        evaluation           1.460
         graph                1.477
8        evaluation           1.486
         graph                1.467
9        evaluation           1.495
         graph                1.440
10       evaluation           1.481
         graph                1.491
11       evaluation           1.502
         graph                1.468
12       evaluation           1.426
         graph                1.459
13       evaluation           1.472
         graph                1.500
14       evaluation           1.500
         graph                1.469
15       evaluation           1.488
         graph                1.484
16       evaluation           1.468
         graph                1.487
```

On the first step the distribution is given by the initial state `[0.33 0.33 0.34]` and approaches an equilibrium distribution, given by the eigenvectors of the transition matrix is v=(15/62,21/62,1), and normalized into a probablity and this agrees with the histograms (so we reach the equilibium distribution very quickly and don't need to take many steps).

In [482…

```python
stationary_dist = np.array([15/62,21/62,1])
stationary_dist / stationary_dist.sum()*num_samples
```

Out[482… array([153.06122449, 214.28571429, 632.65306122])

`4.daphne` : Bayesian Neural Network Learning

return is list of `[W_0, b_0, W_1, b_1]` with shapes (10,1),(10,1),(10,10),(10,1), and all params are distributed as $\mathcal{N}[\mu=0,\sigma=1]$

I flattened $W1$ so that $W1_{ij}$ is labelled `W_1_{i x j}`

In [484…

```python
graph = load_graph(i=4)
graph[-1]
```

Out[484…

```
['vector',
 ['vector',
  ['vector', 'sample1'],
  ['vector', 'sample2'],
  ['vector', 'sample3'],
  ['vector', 'sample4'],
  ['vector', 'sample5'],
  ['vector', 'sample6'],
  ['vector', 'sample7'],
  ['vector', 'sample8'],
  ['vector', 'sample9'],
  ['vector', 'sample10']],
 ['vector',
  ['vector', 'sample121'],
  ['vector', 'sample122'],
  ['vector', 'sample123'],
  ['vector', 'sample124'],
  ['vector', 'sample125'],
  ['vector', 'sample126'],
  ['vector', 'sample127'],
  ['vector', 'sample128'],
  ['vector', 'sample129'],
  ['vector', 'sample130']],
 ['vector',
  ['vector',
   'sample11',
   'sample12',
   'sample13',
   'sample14',
   'sample15',
   'sample16',
   'sample17',
   'sample18',
   'sample19',
   'sample20'],
  ['vector',
   'sample21',
   'sample22',
   'sample23',
   'sample24',
   'sample25',
   'sample26',
   'sample27',
   'sample28',
   'sample29',
   'sample30'],
  ['vector',
   'sample31',
   'sample32',
   'sample33',
   'sample34',
   'sample35',
   'sample36',
   'sample37',
   'sample38',
   'sample39',
   'sample40'],
  ['vector',
   'sample41',
   'sample42',
   'sample43',
   'sample44',
   'sample45',
   'sample46',
   'sample47',
   'sample48',
   'sample49',
   'sample50'],
  ['vector',
   'sample51',
   'sample52',
   'sample53',
   'sample54',
   'sample55',
   'sample56',
   'sample57',
   'sample58',
   'sample59',
   'sample60'],
  ['vector',
   'sample61',
   'sample62',
   'sample63',
   'sample64',
   'sample65',
   'sample66',
   'sample67',
   'sample68',
   'sample69',
   'sample70'],
  ['vector',
   'sample71',
   'sample72',
   'sample73',
   'sample74',
   'sample75',
   'sample76',
   'sample77',
   'sample78',
   'sample79',
   'sample80'],
  ['vector',
   'sample81',
   'sample82',
   'sample83',
   'sample84',
   'sample85',
   'sample86',
   'sample87',
   'sample88',
   'sample89',
   'sample90'],
  ['vector',
   'sample91',
   'sample92',
   'sample93',
   'sample94',
   'sample95',
   'sample96',
   'sample97',
   'sample98',
   'sample99',
   'sample100'],
  ['vector',
   'sample101',
   'sample102',
   'sample103',
   'sample104',
   'sample105',
   'sample106',
   'sample107',
   'sample108',
   'sample109',
   'sample110']],
 ['vector',
  ['vector', 'sample131'],
  ['vector', 'sample132'],
  ['vector', 'sample133'],
  ['vector', 'sample134'],
  ['vector', 'sample135'],
  ['vector', 'sample136'],
  ['vector', 'sample137'],
  ['vector', 'sample138'],
  ['vector', 'sample139'],
  ['vector', 'sample140']]]
```

In [485…

```python
ast = load_ast(i=4)
ast
```

Out[485…

```
[['let',
  ['weight-prior', ['normal', 0, 1]],
  ['let',
   ['W_0',
    ['vector',
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']],
     ['vector', ['sample', 'weight-prior']]]],
   ['let',
    ['W_1',
     ['vector',
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']],
      ['vector',
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior'],
       ['sample', 'weight-prior']]]],
    ['let',
     ['W_2',
      ['vector',
       ['vector',
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior'],
        ['sample', 'weight-prior']]]],
     ['let',
      ['b_0',
       ['vector',
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']],
        ['vector', ['sample', 'weight-prior']]]],
      ['let',
       ['b_1',
        ['vector',
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']],
         ['vector', ['sample', 'weight-prior']]]],
       ['let',
        ['b_2', ['vector', ['vector', ['sample', 'weight-prior']]]],
        ['let',
         ['x',
          ['mat-transpose',
           ['vector',
            ['vector', 1],
            ['vector', 2],
            ['vector', 3],
            ['vector', 4],
            ['vector', 5]]]],
         ['let',
          ['y',
           ['vector',
            ['vector', 1],
            ['vector', 4],
            ['vector', 9],
            ['vector', 16],
            ['vector', 25]]],
          ['let',
           ['h_0',
            ['mat-tanh',
             ['mat-add',
              ['mat-mul', 'W_0', 'x'],
              ['mat-repmat', 'b_0', 1, 5]]]],
           ['let',
            ['h_1',
             ['mat-tanh',
              ['mat-add',
               ['mat-mul', 'W_1', 'h_0'],
               ['mat-repmat', 'b_1', 1, 5]]]],
            ['let',
             ['mu',
              ['mat-transpose',
               ['mat-tanh',
                ['mat-add',
                 ['mat-mul', 'W_2', 'h_1'],
                 ['mat-repmat', 'b_2', 1, 5]]]]],
             ['let',
              ['dontcare0',
               ['vector',
                ['let',
                 ['y_r', ['get', 'y', 0]],
                 ['let',
                  ['mu_r', ['get', 'mu', 0]],
                  ['vector',
                   ['let',
                    ['y_rc', ['get', 'y_r', 0]],
                    ['let',
                     ['mu_rc', ['get', 'mu_r', 0]],
                     ['observe', ['normal', 'mu_rc', 1], 'y_rc']]]]]],
                ['let',
                 ['y_r', ['get', 'y', 1]],
                 ['let',
                  ['mu_r', ['get', 'mu', 1]],
                  ['vector',
                   ['let',
                    ['y_rc', ['get', 'y_r', 0]],
                    ['let',
                     ['mu_rc', ['get', 'mu_r', 0]],
                     ['observe', ['normal', 'mu_rc', 1], 'y_rc']]]]]],
                ['let',
                 ['y_r', ['get', 'y', 2]],
                 ['let',
                  ['mu_r', ['get', 'mu', 2]],
                  ['vector',
                   ['let',
                    ['y_rc', ['get', 'y_r', 0]],
                    ['let',
                     ['mu_rc', ['get', 'mu_r', 0]],
                     ['observe', ['normal', 'mu_rc', 1], 'y_rc']]]]]],
                ['let',
                 ['y_r', ['get', 'y', 3]],
                 ['let',
                  ['mu_r', ['get', 'mu', 3]],
                  ['vector',
                   ['let',
                    ['y_rc', ['get', 'y_r', 0]],
                    ['let',
                     ['mu_rc', ['get', 'mu_r', 0]],
                     ['observe', ['normal', 'mu_rc', 1], 'y_rc']]]]]],
                ['let',
                 ['y_r', ['get', 'y', 4]],
                 ['let',
                  ['mu_r', ['get', 'mu', 4]],
                  ['vector',
                   ['let',
                    ['y_rc', ['get', 'y_r', 0]],
                    ['let',
                     ['mu_rc', ['get', 'mu_r', 0]],
                     ['observe', ['normal', 'mu_rc', 1], 'y_rc']]]]]]]],
              ['vector', 'W_0', 'b_0', 'W_1', 'b_1']]]]]]]]]]]]]]]
```

In [474…

```python
label_stem_li = []
values_li = []
variables_li = []
sample_method_li=[]

def parse_helper(arr,label_stem):
    arr = np.array(arr).flatten()
    labels = [label_stem+str(i) for i in range(arr.size)]
    return arr, labels

num_samples=1000
for _ in range(num_samples):
    for (W_0, b_0, W_1, b_1), sample_method in zip([sample_from_joint(graph), evaluate_program(ast)[0]], ['grap
        for arr_not_flattened, label_stem in zip([W_0, b_0, W_1, b_1],['W_0','b_0','W_1','b_1']):
            arr, labels = parse_helper(arr_not_flattened,label_stem+'_')
            label_stem_li.extend([label_stem]*arr.size)
            values_li.extend(arr.tolist())
            variables_li.extend(labels)
            sample_method_li.extend([sample_method]*arr.size)
```

In [488…

```python
df = pd.DataFrame({'value':values_li,'variable':variables_li,'return_object':label_stem_li,'sample_method':samp
```

In [499…

```python
g = sns.FacetGrid(df, row="variable",col='sample_method')
g.map(sns.histplot, "value")

plt.subplots_adjust(top=.978, wspace=1)
plt.suptitle('Bayes NN')
```

Out[499… Text(0.5, 0.98, 'Bayes NN')

In [487…

```
print(df.groupby(['variable','sample_method']).mean().rename(columns={'value':'expectation'}).to_string())
```

```
                      expectation
variable sample_method
W_0_0    evaluation     -0.043096
         graph          -0.002899
W_0_1    evaluation     -0.011218
         graph          -0.033593
W_0_2    evaluation      0.023124
         graph          -0.007882
W_0_3    evaluation     -0.007883
         graph          -0.023510
W_0_4    evaluation      0.007085
         graph           0.050954
W_0_5    evaluation      0.023969
         graph           0.042748
W_0_6    evaluation      0.042733
         graph           0.033198
W_0_7    evaluation      0.014269
         graph           0.014705
W_0_8    evaluation      0.068341
         graph          -0.003389
W_0_9    evaluation     -0.002856
         graph           0.058859
W_1_0    evaluation      0.001055
         graph           0.020127
W_1_1    evaluation      0.013470
         graph          -0.043318
W_1_10   evaluation      0.023726
         graph           0.017620
W_1_11   evaluation     -0.026489
         graph          -0.018569
W_1_12   evaluation     -0.013365
         graph           0.055238
W_1_13   evaluation      0.059385
         graph          -0.037465
W_1_14   evaluation     -0.027377
         graph          -0.029978
W_1_15   evaluation      0.053660
         graph           0.034047
W_1_16   evaluation      0.074670
         graph           0.006812
W_1_17   evaluation      0.030654
         graph           0.003054
W_1_18   evaluation      0.021280
         graph          -0.005831
W_1_19   evaluation     -0.023708
         graph           0.005762
W_1_2    evaluation      0.018799
         graph          -0.003303
W_1_20   evaluation     -0.013383
         graph           0.011148
W_1_21   evaluation     -0.004988
         graph          -0.032956
W_1_22   evaluation      0.027611
         graph          -0.008863
W_1_23   evaluation     -0.035528
         graph           0.001008
W_1_24   evaluation      0.014503
         graph           0.012526
W_1_25   evaluation      0.015993
         graph           0.020441
W_1_26   evaluation     -0.033211
         graph          -0.066631
W_1_27   evaluation     -0.032335
         graph           0.008881
W_1_28   evaluation      0.037912
         graph           0.078209
W_1_29   evaluation      0.003086
         graph          -0.028768
W_1_3    evaluation      0.013786
         graph          -0.032283
W_1_30   evaluation     -0.052127
         graph          -0.030480
W_1_31   evaluation      0.043241
         graph          -0.051870
W_1_32   evaluation     -0.034865
         graph          -0.015598
W_1_33   evaluation      0.025566
         graph           0.015691
W_1_34   evaluation      0.031062
         graph           0.045557
W_1_35   evaluation      0.001692
         graph          -0.014125
W_1_36   evaluation      0.031407
         graph          -0.008885
W_1_37   evaluation     -0.015215
         graph           0.038950
W_1_38   evaluation      0.007364
         graph           0.024350
W_1_39   evaluation      0.024892
         graph           0.000538
W_1_4    evaluation     -0.008642
         graph           0.042645
W_1_40   evaluation      0.019825
         graph          -0.012079
W_1_41   evaluation     -0.029104
         graph          -0.053125
W_1_42   evaluation     -0.015567
         graph          -0.022837
W_1_43   evaluation      0.003420
         graph          -0.021354
W_1_44   evaluation      0.008602
         graph          -0.022150
W_1_45   evaluation      0.026408
         graph          -0.035884
W_1_46   evaluation      0.008283
         graph          -0.000285
W_1_47   evaluation     -0.015262
         graph          -0.052496
W_1_48   evaluation      0.023248
         graph           0.016385
W_1_49   evaluation     -0.068306
         graph          -0.030757
W_1_5    evaluation      0.029876
         graph          -0.049021
W_1_50   evaluation      0.032010
         graph          -0.037616
W_1_51   evaluation      0.033658
         graph           0.045790
W_1_52   evaluation     -0.064597
         graph          -0.041400
W_1_53   evaluation     -0.000932
         graph           0.025142
W_1_54   evaluation     -0.021481
         graph          -0.032912
W_1_55   evaluation      0.008208
         graph          -0.030879
W_1_56   evaluation      0.004547
         graph          -0.028533
W_1_57   evaluation     -0.004387
         graph          -0.006601
W_1_58   evaluation     -0.020878
         graph           0.018207
W_1_59   evaluation      0.021497
         graph           0.077369
W_1_6    evaluation      0.001002
         graph           0.073025
W_1_60   evaluation      0.045339
         graph           0.048856
W_1_61   evaluation     -0.053190
         graph           0.001706
W_1_62   evaluation      0.009432
         graph           0.043858
W_1_63   evaluation      0.041799
         graph          -0.036877
W_1_64   evaluation     -0.025844
         graph           0.025104
W_1_65   evaluation     -0.002227
         graph           0.033115
W_1_66   evaluation      0.036209
         graph          -0.001529
W_1_67   evaluation      0.021623
         graph           0.011335
W_1_68   evaluation     -0.004372
         graph           0.009701
W_1_69   evaluation      0.024645
         graph           0.002574
W_1_7    evaluation     -0.084780
         graph           0.032760
W_1_70   evaluation     -0.008048
         graph          -0.000919
W_1_71   evaluation     -0.013220
         graph          -0.000034
W_1_72   evaluation      0.046843
         graph           0.036138
W_1_73   evaluation     -0.013116
         graph           0.028072
W_1_74   evaluation     -0.032797
         graph          -0.014004
W_1_75   evaluation     -0.022093
         graph          -0.004522
W_1_76   evaluation     -0.016256
         graph           0.020249
W_1_77   evaluation      0.039264
         graph           0.008742
W_1_78   evaluation     -0.037972
         graph          -0.030033
W_1_79   evaluation     -0.033672
         graph           0.031142
W_1_8    evaluation     -0.016617
         graph           0.051728
W_1_80   evaluation     -0.078374
         graph          -0.005936
W_1_81   evaluation     -0.015525
         graph          -0.047583
W_1_82   evaluation      0.000009
         graph           0.001156
W_1_83   evaluation     -0.032259
         graph          -0.060867
W_1_84   evaluation     -0.032813
         graph          -0.079700
W_1_85   evaluation     -0.011039
         graph          -0.026324
W_1_86   evaluation     -0.017718
         graph          -0.015276
W_1_87   evaluation     -0.031048
         graph           0.044101
W_1_88   evaluation     -0.019579
         graph          -0.032261
W_1_89   evaluation     -0.039685
         graph           0.001333
W_1_9    evaluation     -0.006032
         graph           0.027811
W_1_90   evaluation      0.011231
         graph          -0.026312
W_1_91   evaluation      0.017094
         graph          -0.002617
W_1_92   evaluation     -0.030046
         graph          -0.017592
W_1_93   evaluation      0.024885
         graph          -0.057459
W_1_94   evaluation      0.040548
         graph           0.051746
W_1_95   evaluation     -0.025853
         graph          -0.050530
W_1_96   evaluation     -0.001166
         graph          -0.039160
W_1_97   evaluation      0.002232
         graph           0.031179
W_1_98   evaluation     -0.049502
         graph          -0.025896
W_1_99   evaluation      0.025903
         graph           0.048308
b_0_0    evaluation     -0.042520
         graph          -0.032526
b_0_1    evaluation     -0.009018
         graph          -0.000719
b_0_2    evaluation      0.021713
         graph           0.010405
b_0_3    evaluation     -0.009456
         graph           0.009236
b_0_4    evaluation     -0.024035
         graph          -0.038964
b_0_5    evaluation     -0.025644
         graph          -0.010503
b_0_6    evaluation     -0.073194
         graph           0.069146
b_0_7    evaluation      0.041229
         graph           0.011530
b_0_8    evaluation      0.045325
         graph          -0.018643
b_0_9    evaluation     -0.032414
         graph           0.030348
b_1_0    evaluation     -0.014699
         graph           0.004404
b_1_1    evaluation      0.000337
         graph          -0.018796
b_1_2    evaluation      0.009761
         graph           0.004372
b_1_3    evaluation      0.028725
         graph          -0.035376
b_1_4    evaluation     -0.038660
         graph          -0.041216
b_1_5    evaluation      0.009500
         graph          -0.035018
b_1_6    evaluation     -0.002683
         graph          -0.009070
b_1_7    evaluation      0.005556
         graph           0.034422
b_1_8    evaluation      0.000002
         graph          -0.049465
b_1_9    evaluation      0.063307
         graph          -0.006531
```

In [477…

```
print(df.groupby(['variable','sample_method']).std().rename(columns={'value':'std'}).to_string())
```

```
                            std
variable sample_method
W_0_0    evaluation     1.046078
         graph          1.004075
W_0_1    evaluation     1.006705
         graph          0.997485
W_0_2    evaluation     1.022333
         graph          0.961551
W_0_3    evaluation     1.010893
         graph          1.022728
W_0_4    evaluation     0.992392
         graph          1.035903
W_0_5    evaluation     1.016629
         graph          1.036223
W_0_6    evaluation     1.010043
         graph          0.972032
W_0_7    evaluation     1.024951
         graph          1.006529
W_0_8    evaluation     0.992875
         graph          0.990808
W_0_9    evaluation     0.993283
         graph          0.981686
W_1_0    evaluation     0.982045
         graph          0.988132
W_1_1    evaluation     1.021965
         graph          1.003351
W_1_10   evaluation     0.990999
         graph          0.992141
W_1_11   evaluation     1.024080
         graph          0.988539
W_1_12   evaluation     1.012395
         graph          1.019573
W_1_13   evaluation     1.010986
         graph          1.009732
W_1_14   evaluation     1.031095
         graph          1.010569
W_1_15   evaluation     0.974541
         graph          0.970903
W_1_16   evaluation     0.969177
         graph          0.977258
W_1_17   evaluation     1.014994
         graph          0.991231
W_1_18   evaluation     0.954958
         graph          1.020073
W_1_19   evaluation     0.987844
         graph          1.044341
W_1_2    evaluation     1.003396
         graph          1.004501
W_1_20   evaluation     1.013204
         graph          1.043814
W_1_21   evaluation     0.997192
         graph          0.997449
W_1_22   evaluation     0.983802
         graph          1.022212
W_1_23   evaluation     1.034029
         graph          1.028797
W_1_24   evaluation     1.017770
         graph          1.014058
W_1_25   evaluation     0.967539
         graph          0.978966
W_1_26   evaluation     1.023091
         graph          1.009189
W_1_27   evaluation     0.991643
         graph          1.030557
W_1_28   evaluation     1.025955
         graph          1.007776
W_1_29   evaluation     1.024531
         graph          0.978430
W_1_3    evaluation     1.031433
         graph          0.988780
W_1_30   evaluation     0.954005
         graph          1.016484
W_1_31   evaluation     0.996182
         graph          0.983187
W_1_32   evaluation     0.968425
         graph          0.983832
W_1_33   evaluation     1.015795
         graph          0.987713
W_1_34   evaluation     1.011315
         graph          0.982028
W_1_35   evaluation     1.011677
         graph          0.976697
W_1_36   evaluation     1.009834
         graph          0.998594
W_1_37   evaluation     1.051446
         graph          1.002189
W_1_38   evaluation     0.950666
         graph          1.033302
W_1_39   evaluation     1.035725
         graph          1.026313
W_1_4    evaluation     1.016198
         graph          1.008538
W_1_40   evaluation     1.037525
         graph          1.008861
W_1_41   evaluation     0.994191
         graph          0.987032
W_1_42   evaluation     1.008163
         graph          1.036582
W_1_43   evaluation     0.969050
         graph          0.979858
W_1_44   evaluation     1.007335
         graph          1.011384
W_1_45   evaluation     0.998572
         graph          0.984518
W_1_46   evaluation     1.046581
         graph          1.042685
W_1_47   evaluation     1.003107
         graph          1.008302
W_1_48   evaluation     0.990776
         graph          0.964298
W_1_49   evaluation     1.002440
         graph          1.007511
W_1_5    evaluation     0.993141
         graph          1.056269
W_1_50   evaluation     0.980257
         graph          1.018925
W_1_51   evaluation     1.010919
         graph          1.002943
W_1_52   evaluation     1.000330
         graph          0.977014
W_1_53   evaluation     0.996010
         graph          1.010730
W_1_54   evaluation     0.983698
         graph          1.016061
W_1_55   evaluation     0.973616
         graph          0.990622
W_1_56   evaluation     0.998233
         graph          1.021180
W_1_57   evaluation     0.996141
         graph          1.014539
W_1_58   evaluation     1.025822
         graph          1.022339
W_1_59   evaluation     0.977194
         graph          1.007726
W_1_6    evaluation     1.012846
         graph          0.990668
W_1_60   evaluation     1.004889
         graph          0.991146
W_1_61   evaluation     0.987333
         graph          1.016238
W_1_62   evaluation     1.022363
         graph          1.021559
W_1_63   evaluation     0.990392
         graph          0.981150
W_1_64   evaluation     0.995447
         graph          0.973179
W_1_65   evaluation     1.029878
         graph          1.004797
W_1_66   evaluation     0.970379
         graph          0.999182
W_1_67   evaluation     1.024228
         graph          1.008667
W_1_68   evaluation     0.974206
         graph          0.950650
W_1_69   evaluation     1.002789
         graph          1.012599
W_1_7    evaluation     0.994694
         graph          0.989780
W_1_70   evaluation     1.008912
         graph          0.996592
W_1_71   evaluation     0.955991
         graph          1.000786
W_1_72   evaluation     1.010155
         graph          0.979043
W_1_73   evaluation     0.994611
         graph          1.007648
W_1_74   evaluation     0.974380
         graph          1.012879
W_1_75   evaluation     0.963558
         graph          0.973101
W_1_76   evaluation     1.029099
         graph          0.992586
W_1_77   evaluation     1.040258
         graph          1.008761
W_1_78   evaluation     1.031041
         graph          1.003226
W_1_79   evaluation     0.974365
         graph          1.016809
W_1_8    evaluation     0.991550
         graph          1.001867
W_1_80   evaluation     1.025138
         graph          0.985729
W_1_81   evaluation     0.992682
         graph          0.985119
W_1_82   evaluation     1.015214
         graph          0.965865
W_1_83   evaluation     0.984493
         graph          0.971942
W_1_84   evaluation     0.992801
         graph          1.009771
W_1_85   evaluation     1.016418
         graph          1.030283
W_1_86   evaluation     1.002029
         graph          1.036929
W_1_87   evaluation     1.041908
         graph          0.986610
W_1_88   evaluation     1.008084
         graph          1.001333
W_1_89   evaluation     0.992658
         graph          1.038454
W_1_9    evaluation     1.016651
         graph          0.995542
W_1_90   evaluation     0.988477
         graph          0.990091
W_1_91   evaluation     1.010830
         graph          0.976391
W_1_92   evaluation     1.045746
         graph          1.015307
W_1_93   evaluation     0.995196
         graph          0.969272
W_1_94   evaluation     1.015215
         graph          0.985100
W_1_95   evaluation     0.983829
         graph          0.979103
W_1_96   evaluation     1.025608
         graph          0.968248
W_1_97   evaluation     0.973028
         graph          1.023131
W_1_98   evaluation     0.994668
         graph          0.993061
W_1_99   evaluation     1.014412
         graph          0.975008
b_0_0    evaluation     1.033262
         graph          1.030300
b_0_1    evaluation     0.995352
         graph          0.964775
b_0_2    evaluation     0.981766
         graph          0.990254
b_0_3    evaluation     1.016171
         graph          0.950821
b_0_4    evaluation     0.954396
         graph          0.973914
b_0_5    evaluation     0.982124
         graph          1.053420
b_0_6    evaluation     1.000191
         graph          0.987757
b_0_7    evaluation     0.994068
         graph          1.005113
b_0_8    evaluation     1.004745
         graph          1.021522
b_0_9    evaluation     1.003728
         graph          1.017562
b_1_0    evaluation     1.017138
         graph          0.974304
b_1_1    evaluation     1.003807
         graph          1.009297
b_1_2    evaluation     0.961312
         graph          0.936347
b_1_3    evaluation     1.014101
         graph          0.991930
b_1_4    evaluation     0.995649
         graph          0.989586
b_1_5    evaluation     0.982801
         graph          0.977707
b_1_6    evaluation     0.975103
         graph          1.037723
b_1_7    evaluation     1.014075
         graph          1.002985
b_1_8    evaluation     1.021193
         graph          0.987395
b_1_9    evaluation     0.976115
         graph          1.007316
```

The sample mean and std of each parameters is near its expected value