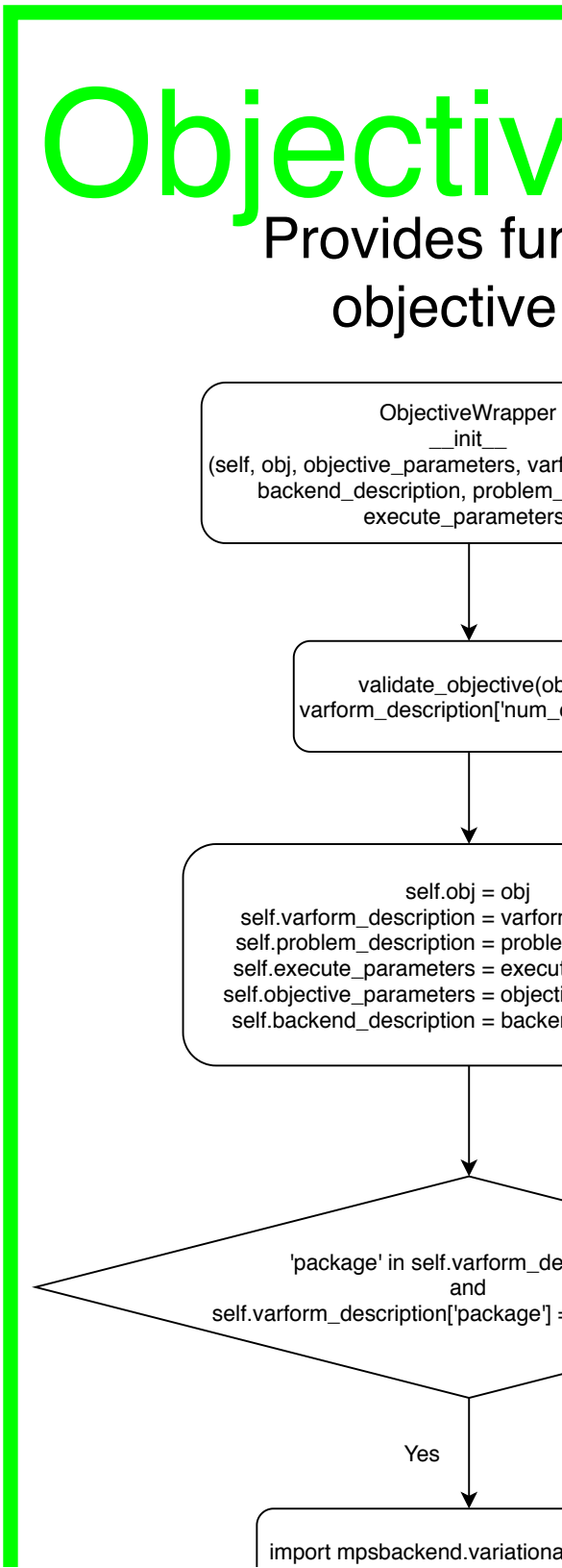
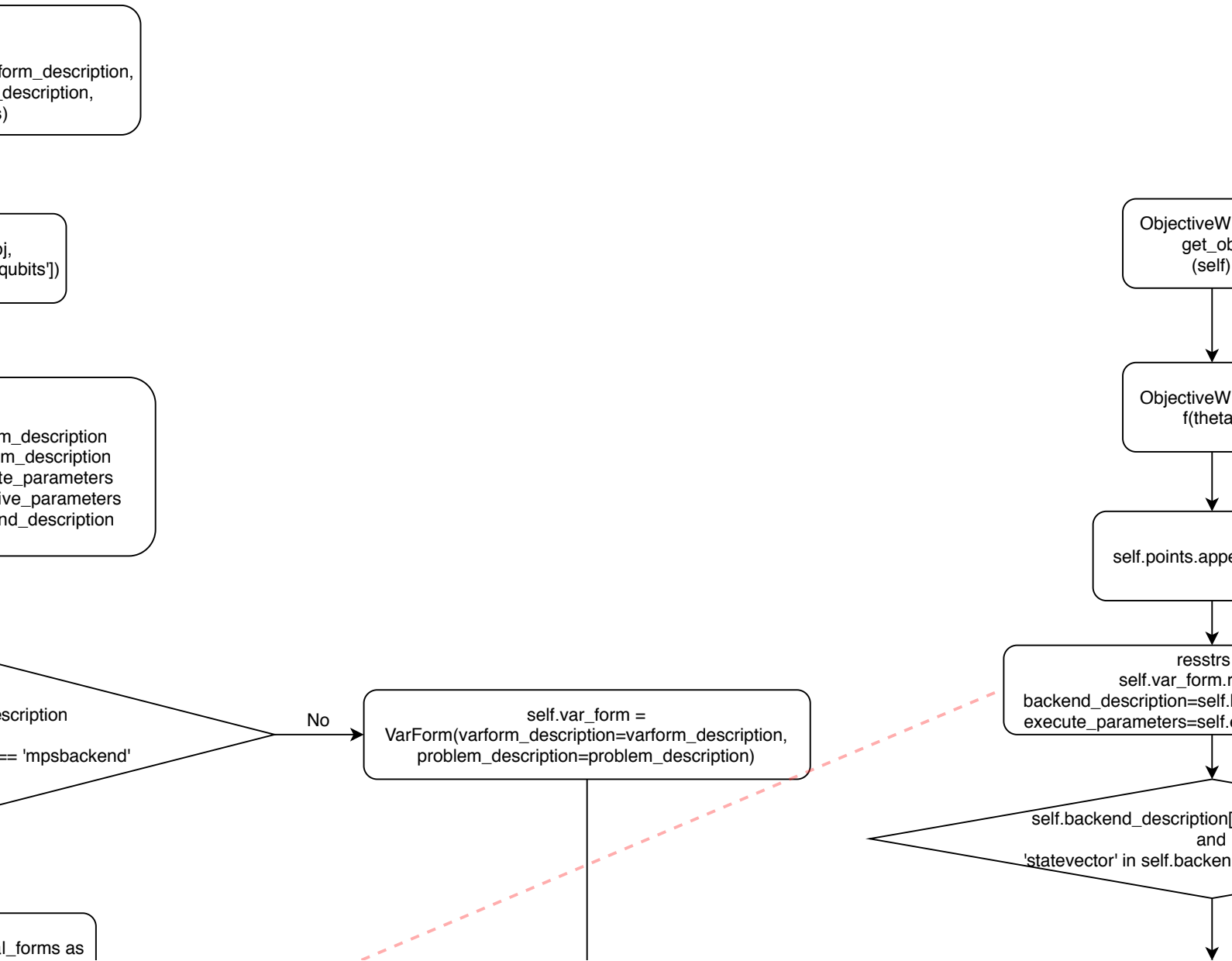
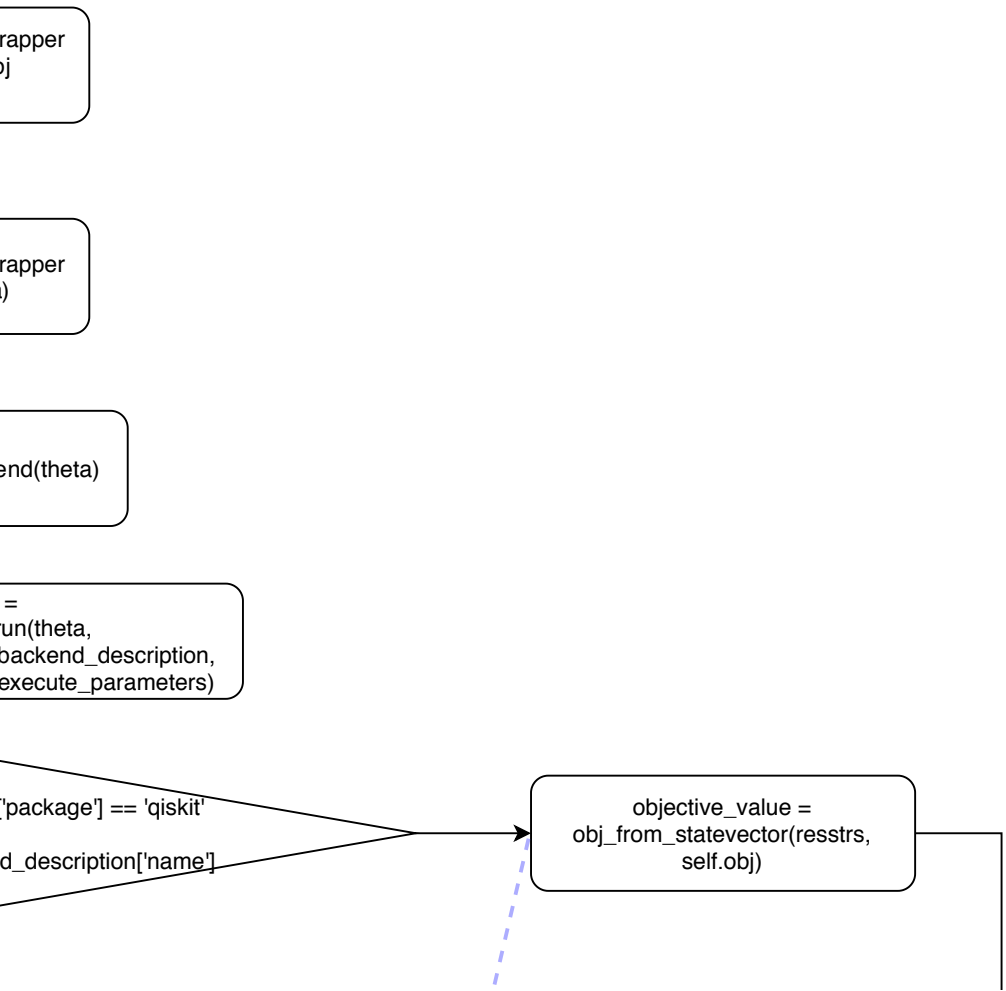
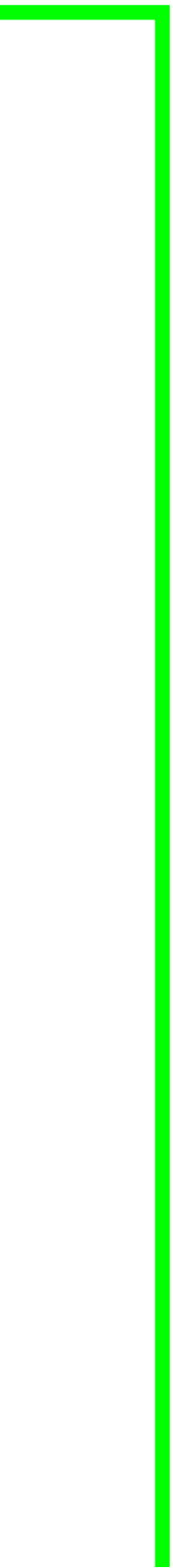

Objectiv
Provides fur
objective
ObjectiveWrapper
__init__
(self, obj, objective_parameters, varf
backend_description, problem_
execute_parameters
validate_objective(ob
varform_description['num_
self.obj = obj
self.varform_description = varform
self.problem_description = proble
self.execute_parameters = execut
self.objective_parameters = objecti
self.backend_description = backe
'package' in self.varform_de
and
self.varform_description['package'] =
Yes
import mpsbackend.variationa

veWrapper.py

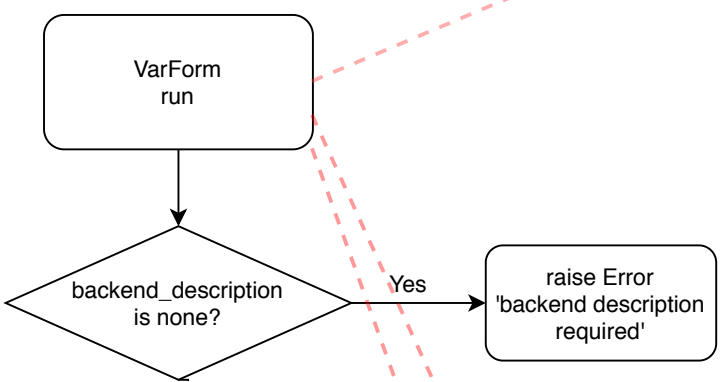
nctions for creating the
function to be used



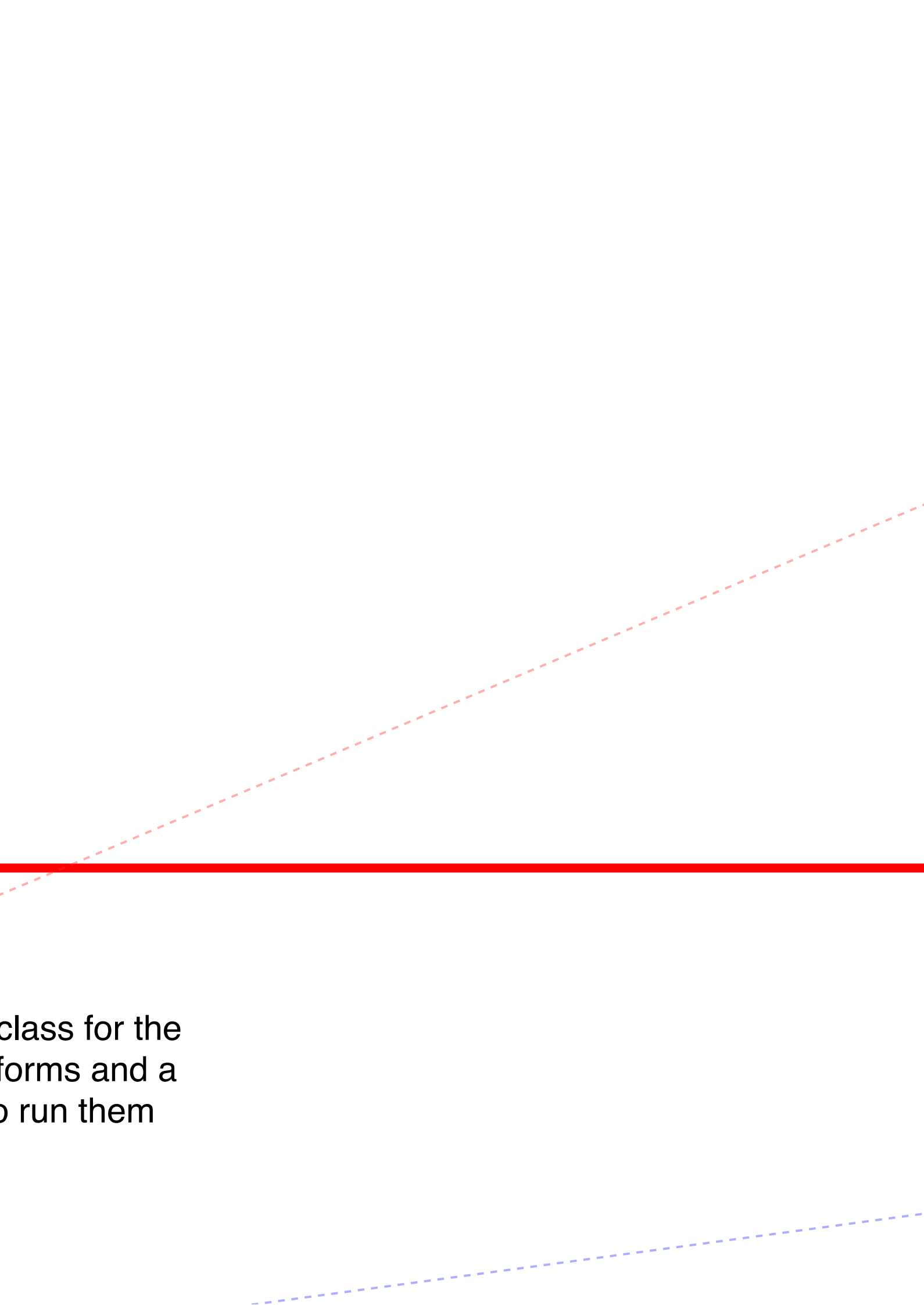

rapper
j
rapper
)
end(theta)
=
un(theta,
backend_description,
execute_parameters)
'package'] == 'qiskit'
d_description['name']
objective_value =
obj_from_statevector(resstrs,
self.obj)



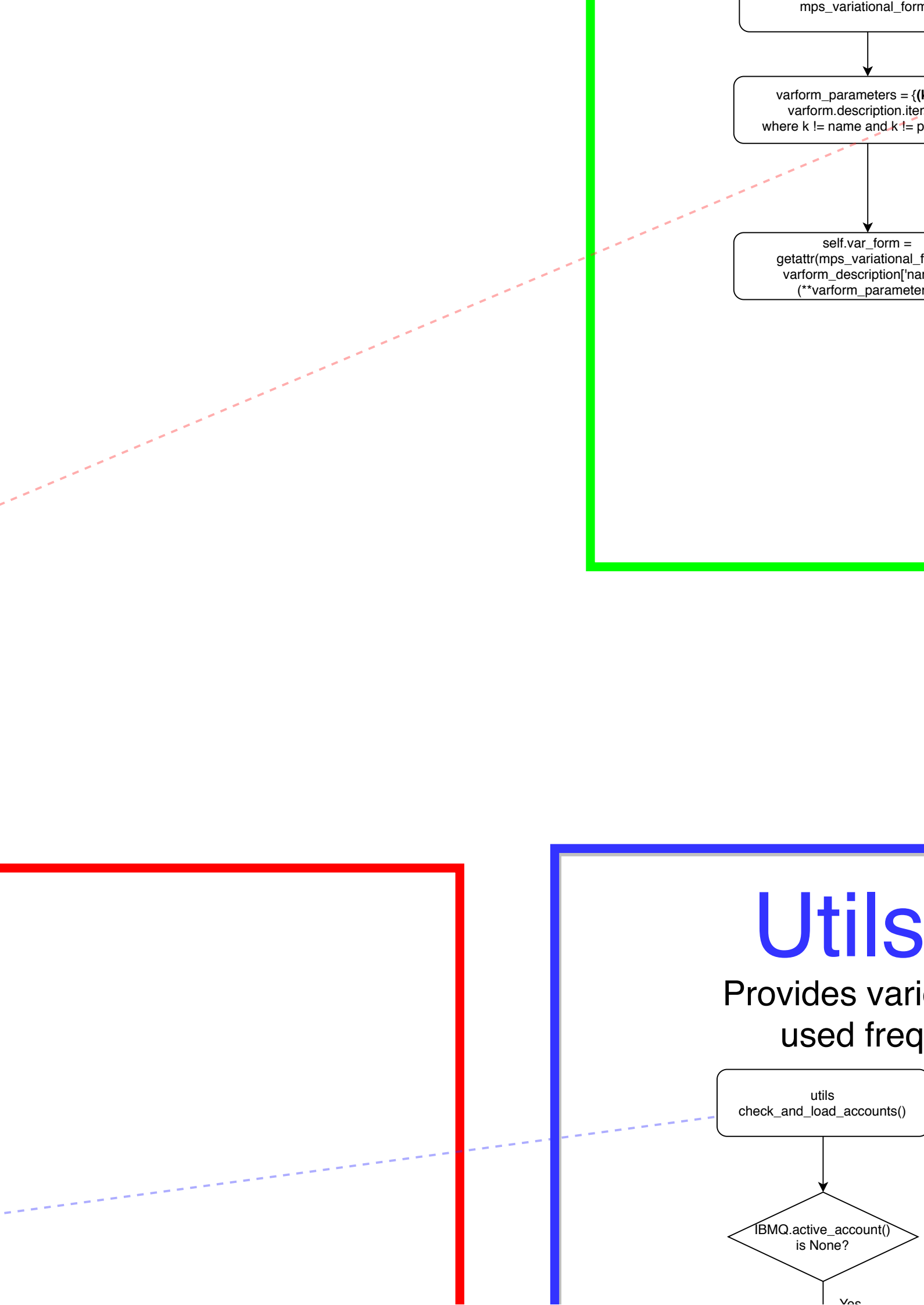
VarForm.py

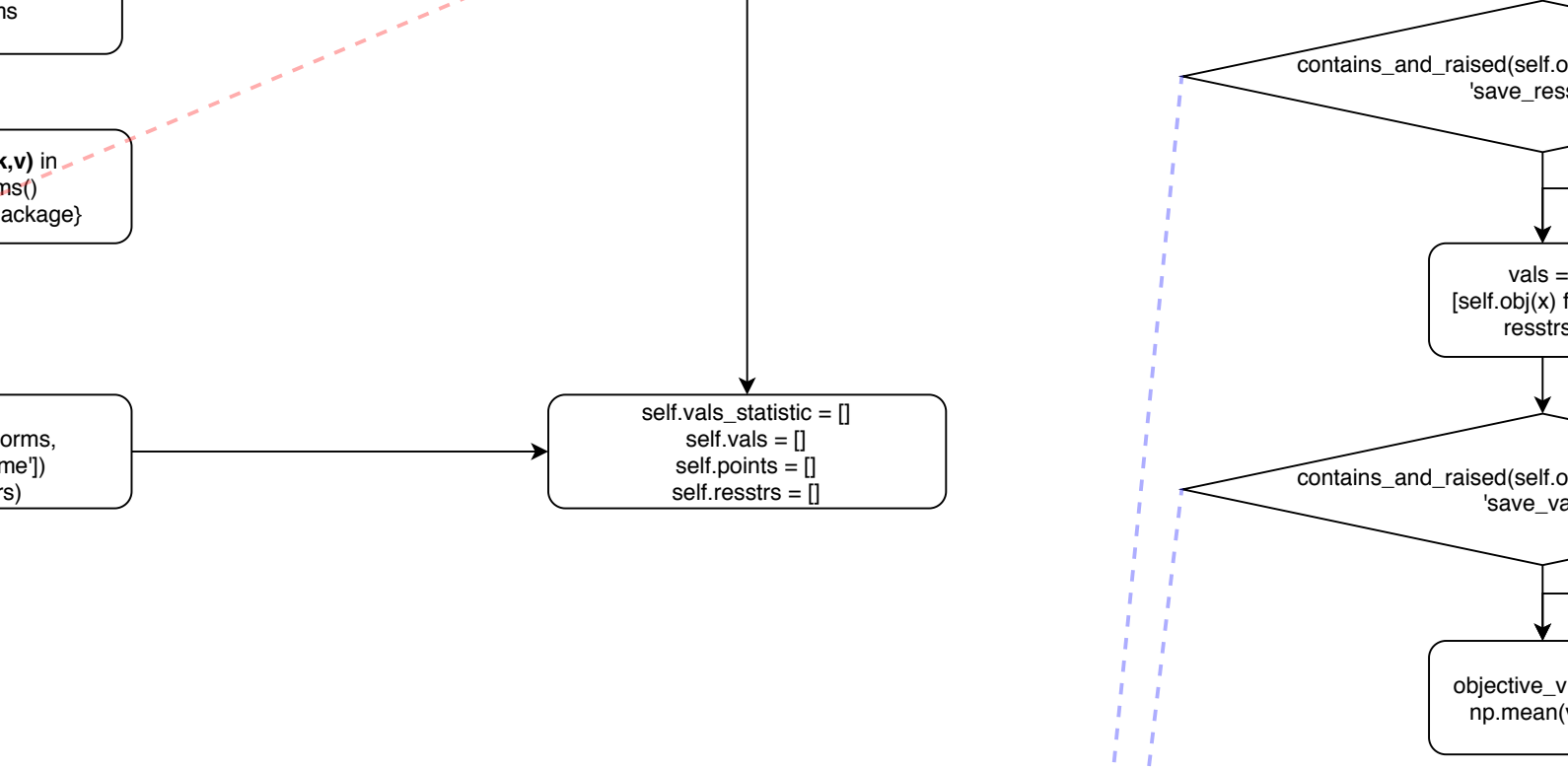


Provides a
variational f
function to



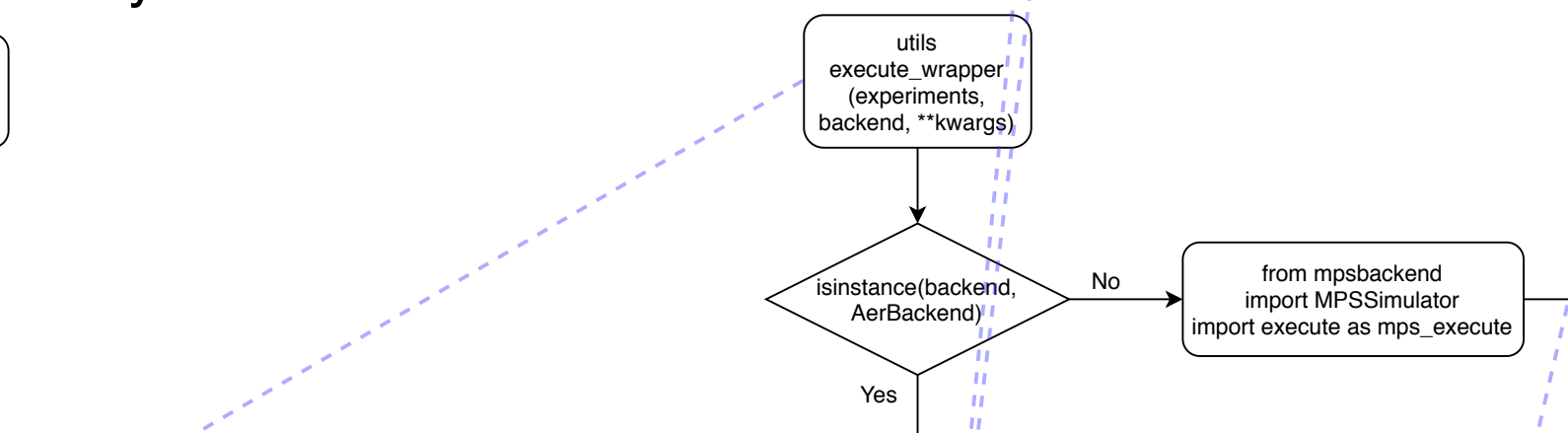
class for the
forms and a
o run them

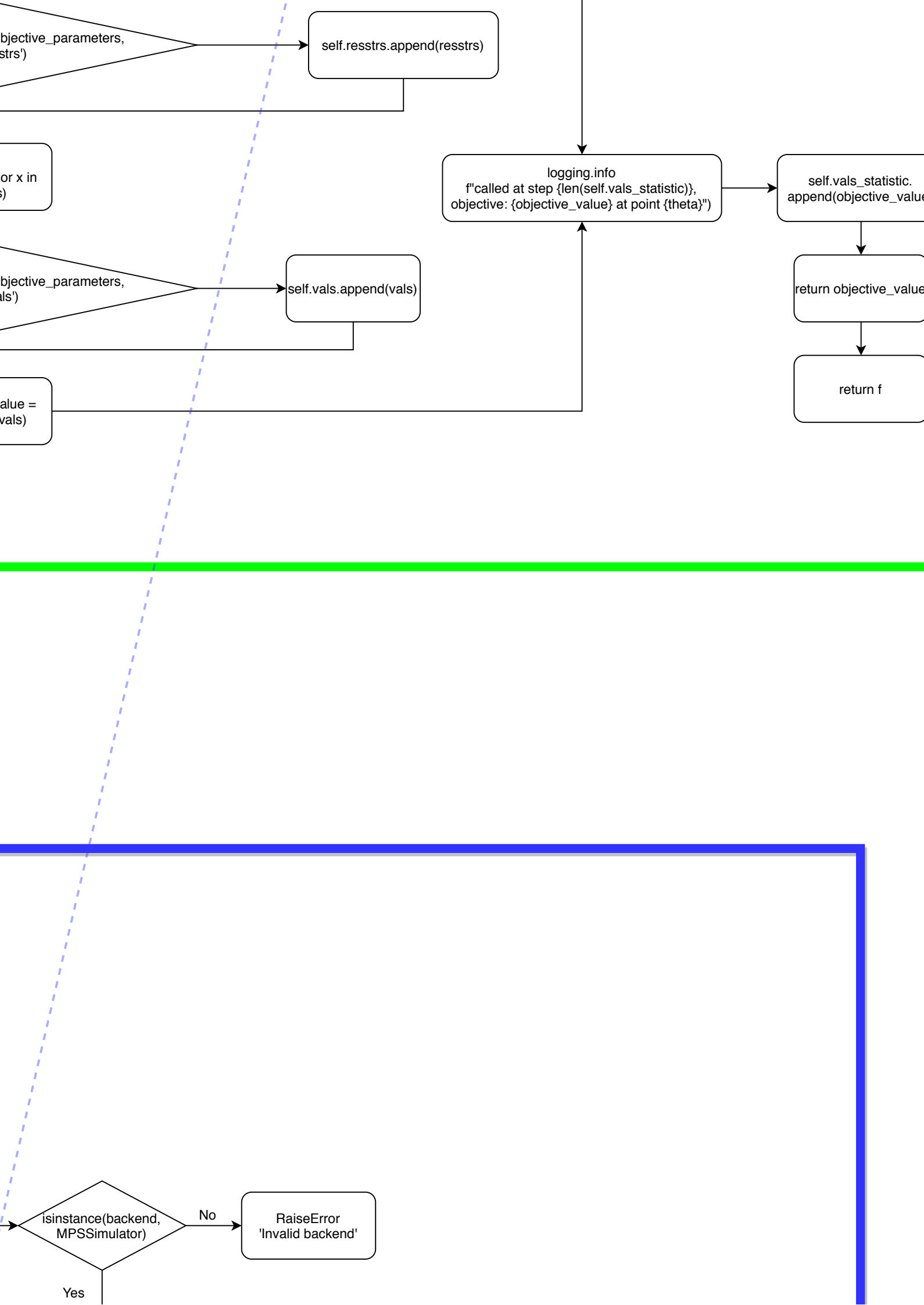

mps_variational_form
varform_parameters = {(k
varform.description.iter
where k != name and k != p
self.var_form =
getattr(mps_variational_f
varform_description['nar
(**varform_parameter
Utils
Provides vari
used freq
utils
check_and_load_accounts()
IBMQ.active_account()
is None?
Yes

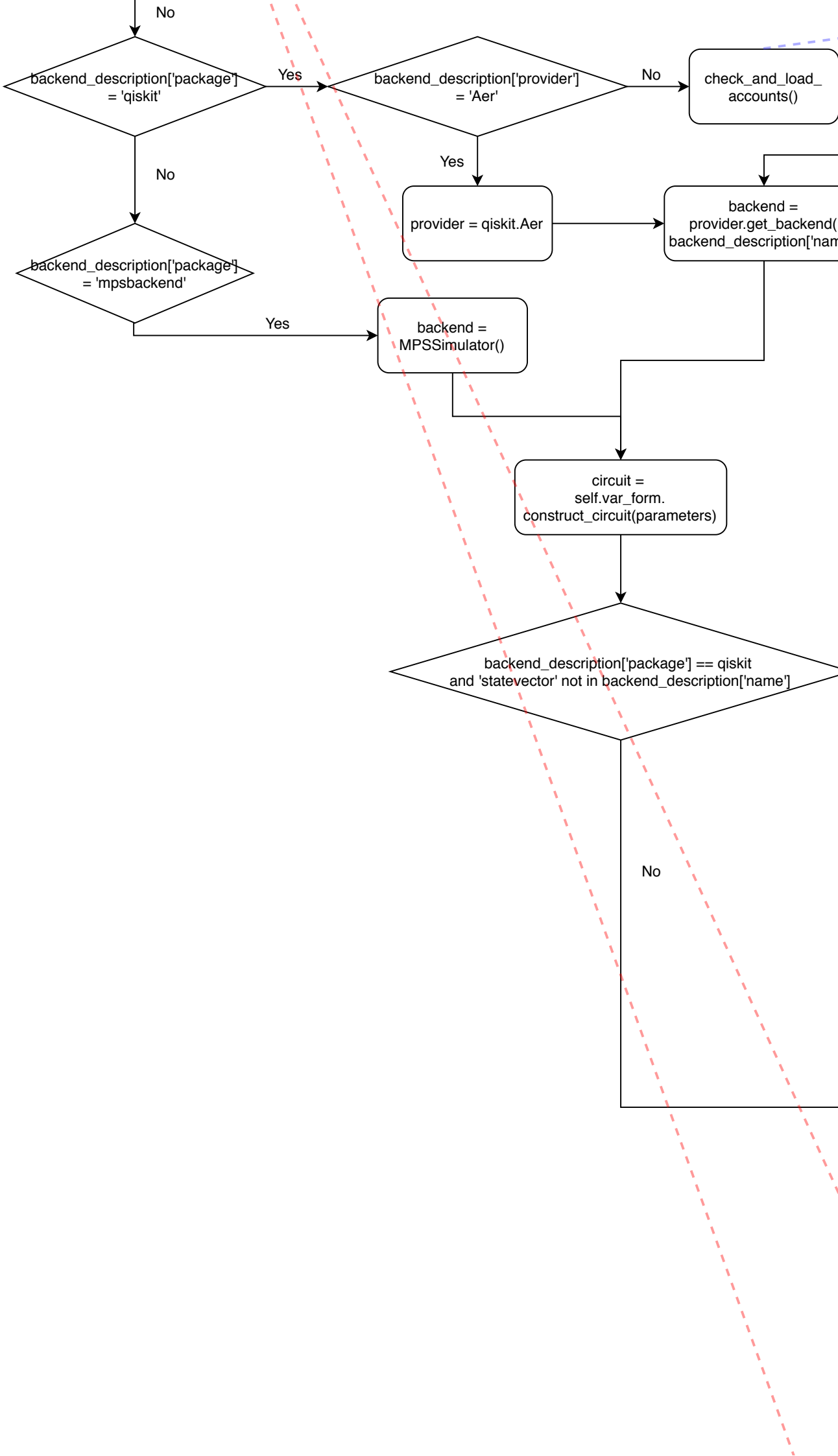


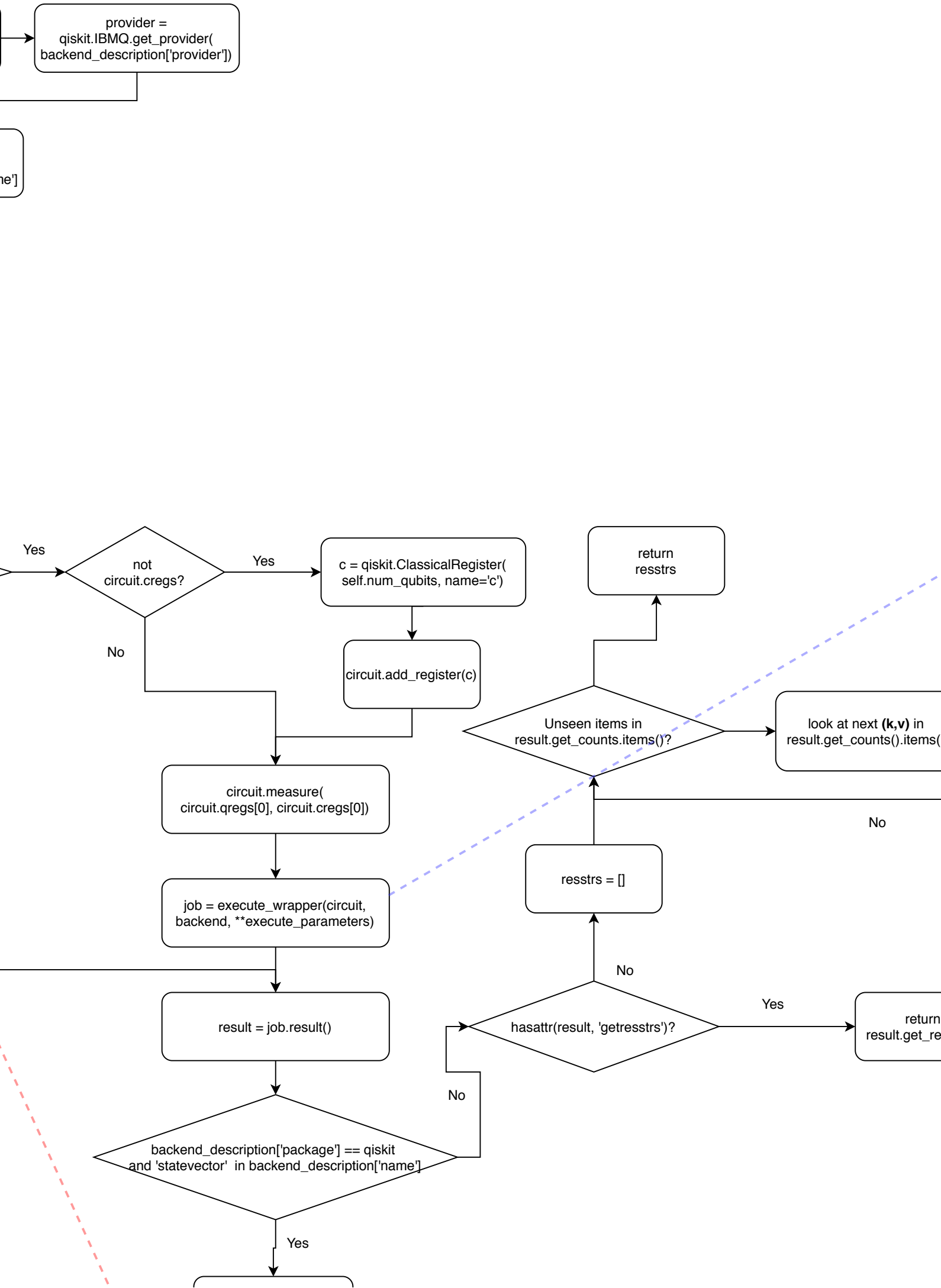
.py

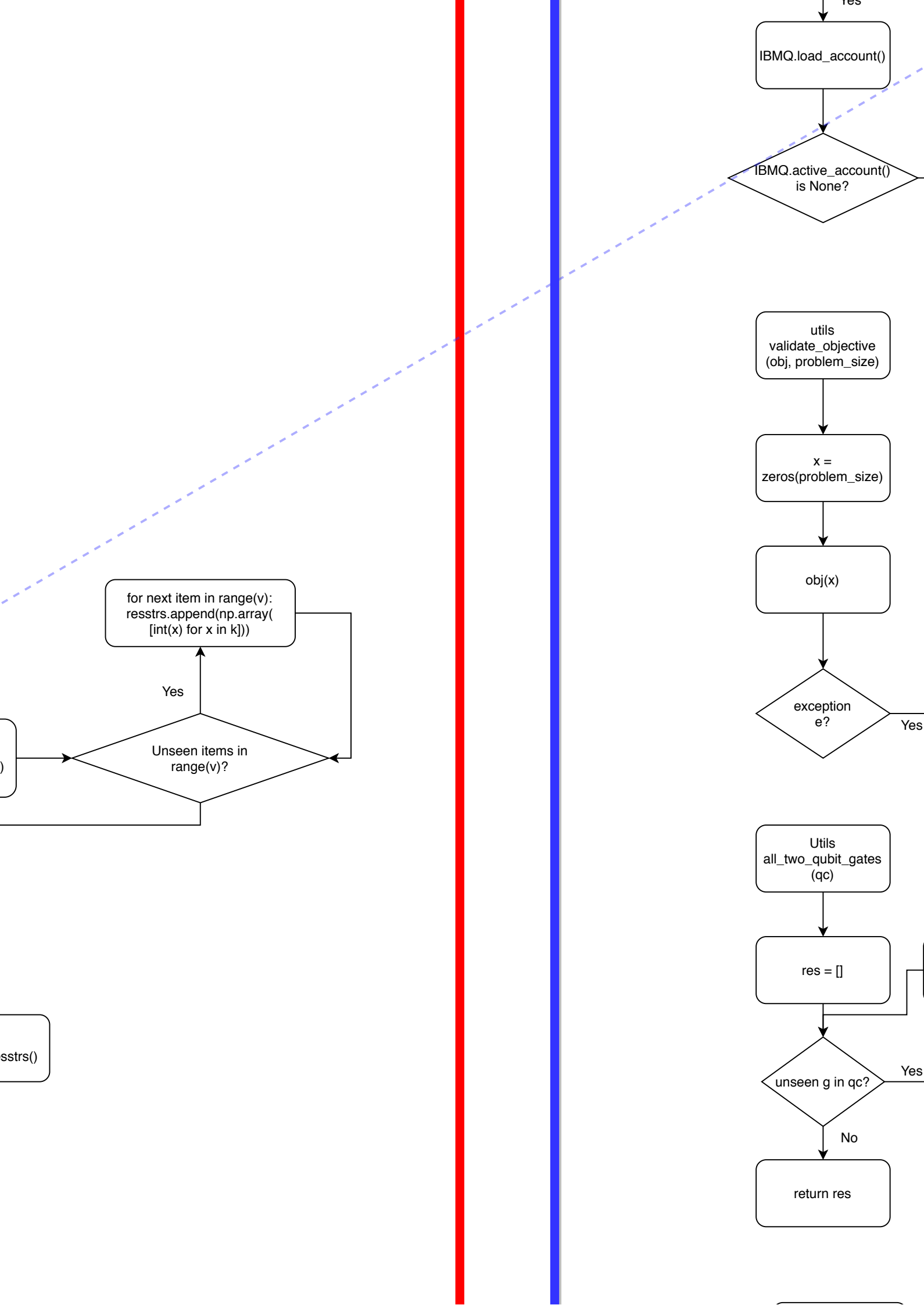
ous utility function that are
uently in other classes

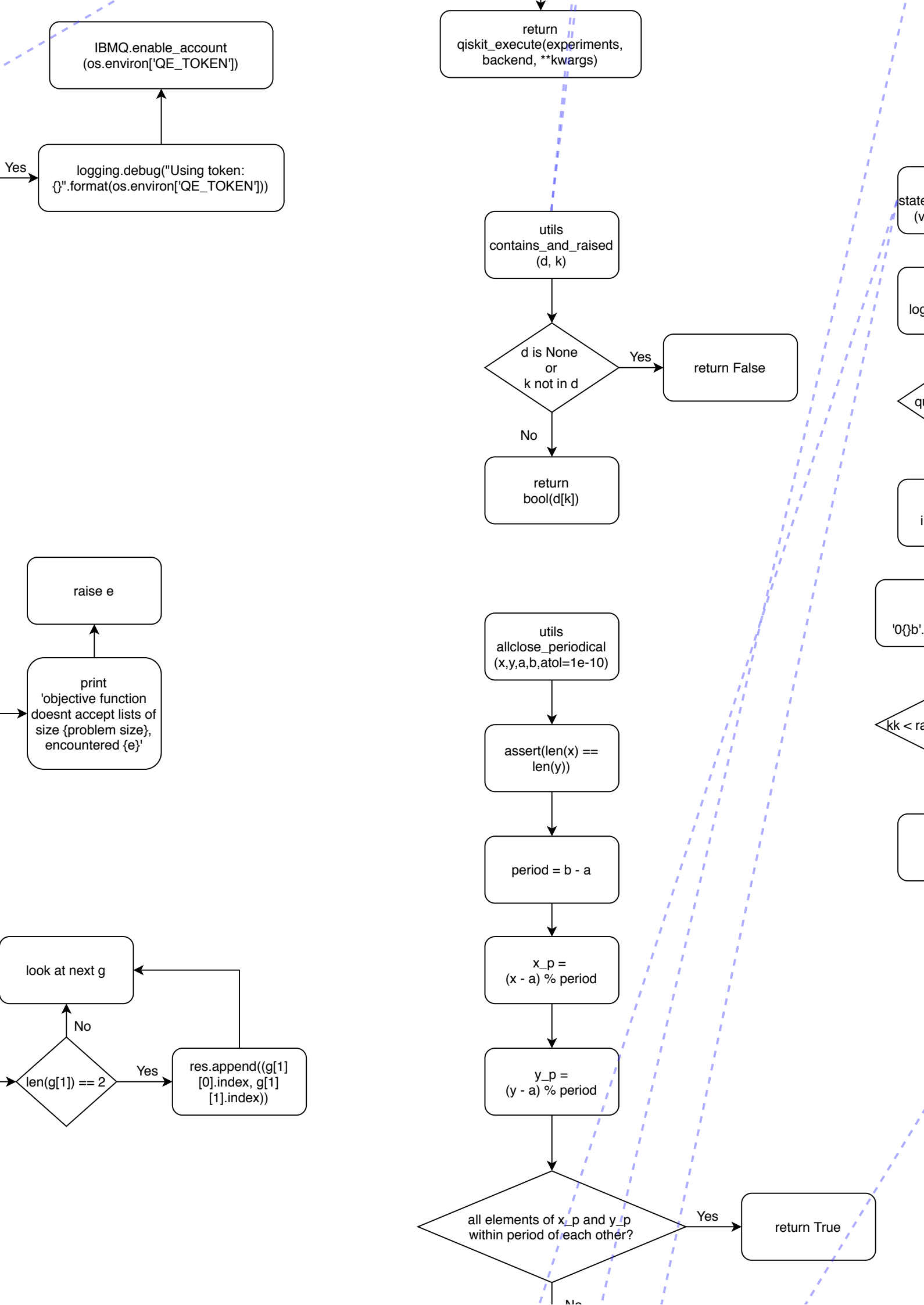


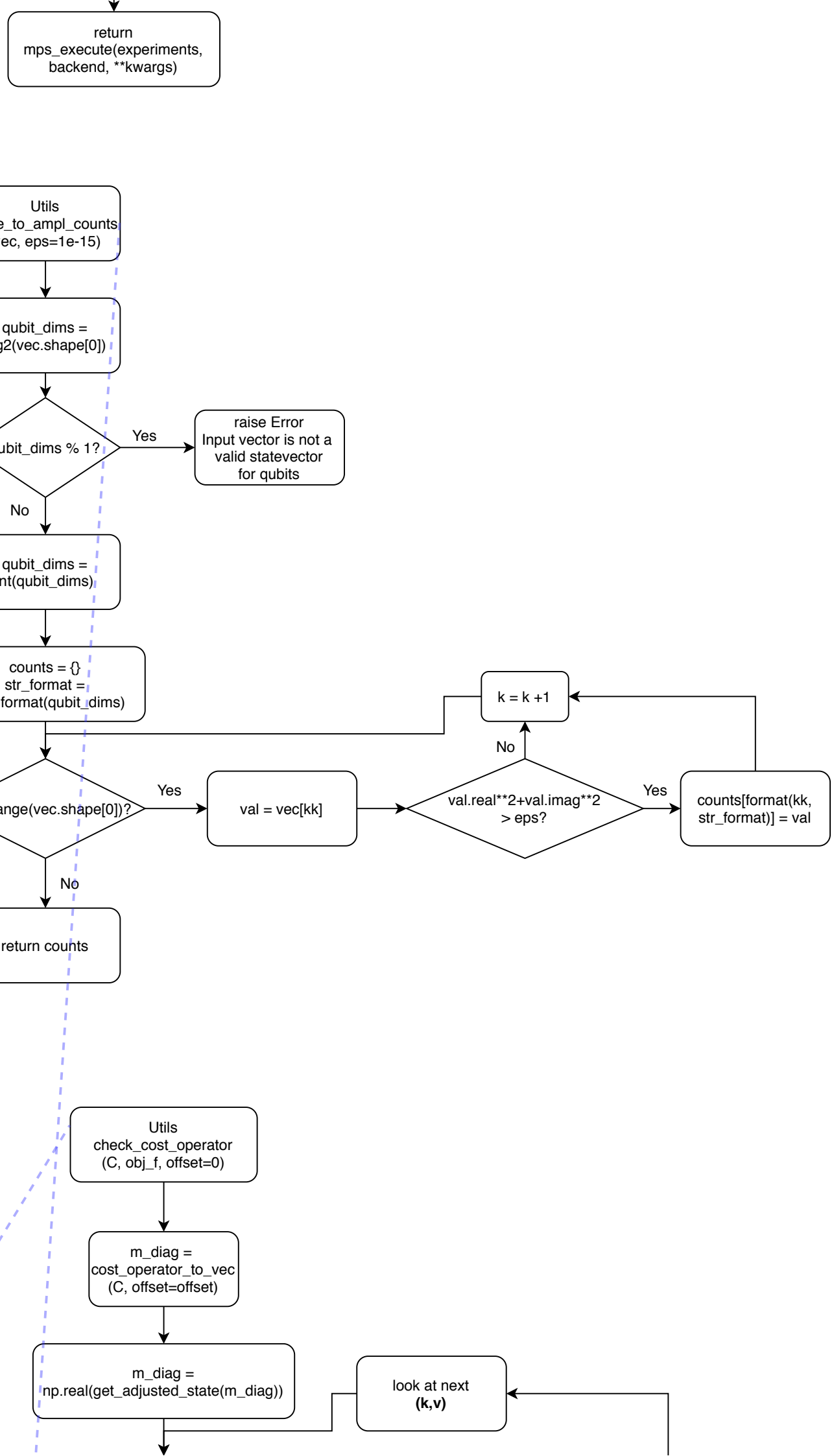

bjective_parameters,
strs')
self.resstrs.append(resstrs)
or x in
s)
logging.info
f"called at step {len(self.vals_statistic)},
objective: {objective_value} at point {theta}")
self.vals_statistic.
append(objective_value
bjective_parameters,
als')
self.vals.append(vals)
return objective_value
alue =
vals)
return f
isinstance(backend,
MPSSimulator)
No
RaiseError
'Invalid backend'
Yes

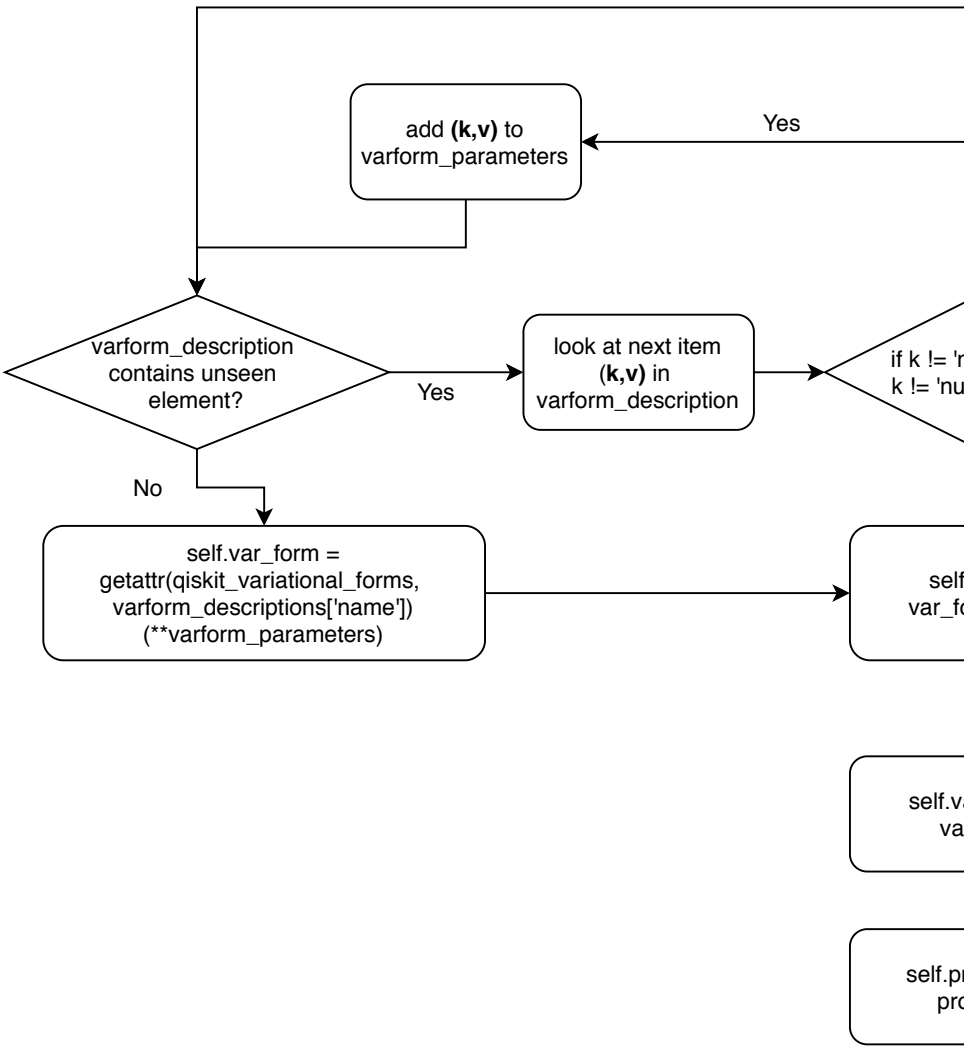

No
backend_description['package']
= 'qiskit'
Yes
backend_description['provider']
= 'Aer'
No
check_and_load_
accounts()
Yes
No
provider = qiskit.Aer
backend =
provider.get_backend(
backend_description['nam
backend_description['package']
= 'mpsbackend'
Yes
backend =
MPSSimulator()
circuit =
self.var_form.
construct_circuit(parameters)
backend_description['package'] == qiskit
and 'statevector' not in backend_description['name']
No


provider =
qiskit.IBMQ.get_provider(
backend_description['provider'])
Yes
not
circuit.cregs?
Yes
c = qiskit.ClassicalRegister(
self.num_qubits, name='c')
No
circuit.add_register(c)
circuit.measure(
circuit.qregs[0], circuit.cregs[0])
job = execute_wrapper(circuit,
backend, **execute_parameters)
result = job.result()
backend_description['package'] == qiskit
and 'statevector' in backend_description['name']
No
Yes
return
resstrs
Unseen items in
result.get_counts.items()?
look at next (k,v) in
result.get_counts().items(
No
resstrs = []
No
hasattr(result, 'getresstrs')?
Yes
return
result.get_re

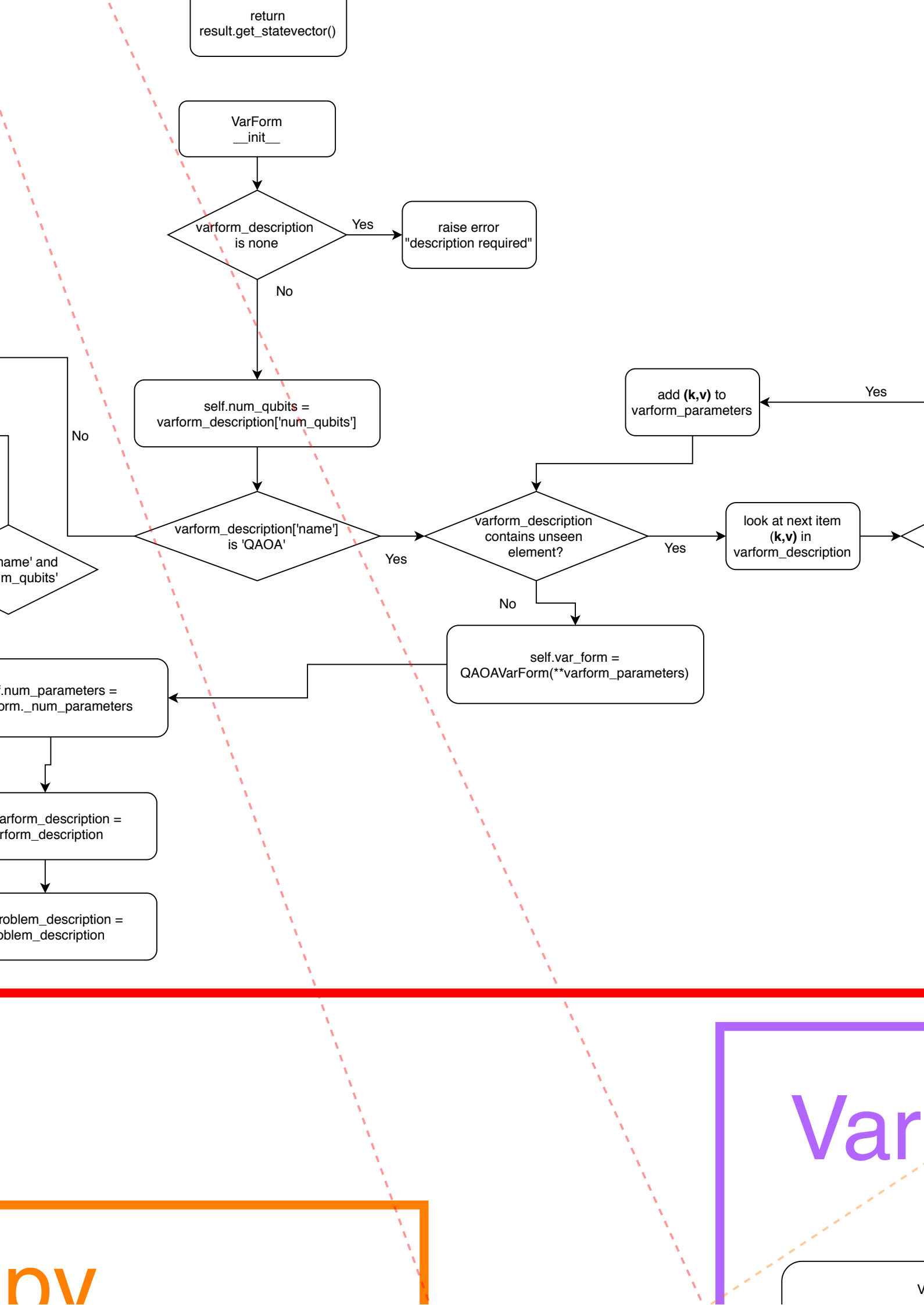

IBMQ.load_account()
IBMQ.active_account() is None?
utils validate_objective (obj, problem_size)
x = zeros(problem_size)
obj(x)
exception e?
Yes
Utils all_two_qubit_gates (qc)
res = []
unseen g in qc?
Yes
No
return res
for next item in range(v): resstrs.append(np.array([int(x) for x in k]))
Yes
Unseen items in range(v)?
esstrs()

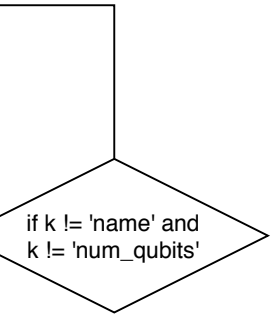

IBMQ.enable_account
(os.environ['QE_TOKEN'])
Yes
logging.debug("Using token:
{}".format(os.environ['QE_TOKEN']))
return
qiskit_execute(experiments,
backend, **kwargs)
utils
contains_and_raised
(d, k)
d is None
or
k not in d
Yes
return False
No
return
bool(d[k])
raise e
print
'objective function
doesnt accept lists of
size {problem size},
encountered {e}'
utils
allclose_periodical
(x,y,a,b,atol=1e-10)
assert(len(x) ==
len(y))
period = b - a
x_p =
(x - a) % period
y_p =
(y - a) % period
all elements of x_p and y_p
within period of each other?
Yes
return True
look at next g
No
len(g[1]) == 2
Yes
res.append((g[1]
[0].index, g[1]
[1].index))

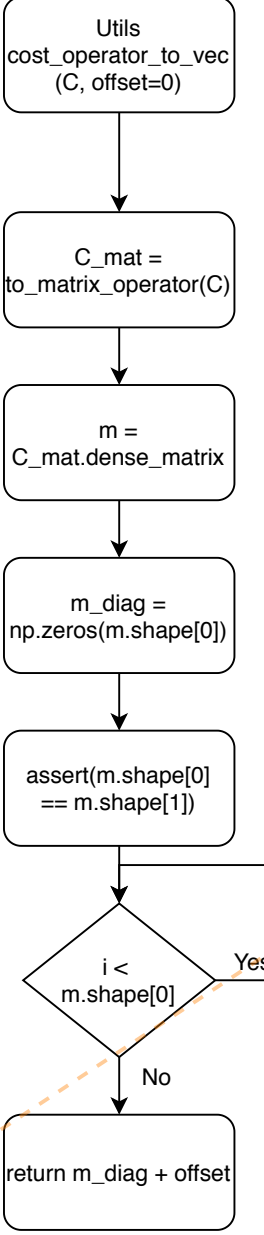

return
mps_execute(experiments,
backend, **kwargs)
Utils
e_to_ampl_counts
ec, eps=1e-15)
qubit_dims =
g2(vec.shape[0])
ubit_dims % 1?
Yes
raise Error
Input vector is not a
valid statevector
for qubits
No
qubit_dims =
nt(qubit_dims)
counts = {}
str_format =
format(qubit_dims)
k = k +1
No
ange(vec.shape[0])?
Yes
val = vec[kk]
val.real**2+val.imag**2
> eps?
Yes
counts[format(kk,
str_format)] = val
No
return counts
Utils
check_cost_operator
(C, obj_f, offset=0)
m_diag =
cost_operator_to_vec
(C, offset=offset)
m_diag =
np.real(get_adjusted_state(m_diag))
look at next
(k,v)


add **(k,v)** to varform_parameters
Yes
varform_description contains unseen element?
Yes
look at next item **(k,v)** in varform_description
No
self.var_form = getattr(qiskit_variational_forms, varform_descriptions['name']) (**varform_parameters)

Endianness

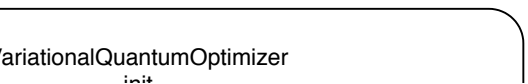

return
result.get_statevector()
VarForm
__init__
varform_description
is none
Yes
raise error
"description required"
No
self.num_qubits =
varform_description['num_qubits']
No
varform_description['name']
is 'QAOA'
Yes
varform_description
contains unseen
element?
Yes
look at next item
(k,v) in
varform_description
Yes
add (k,v) to
varform_parameters
No
self.var_form =
QAOAVarForm(**varform_parameters)
name' and
m_qubits'
.num_parameters =
orm._num_parameters
arform_description =
rform_description
roblem_description =
oblem_description
Var
py

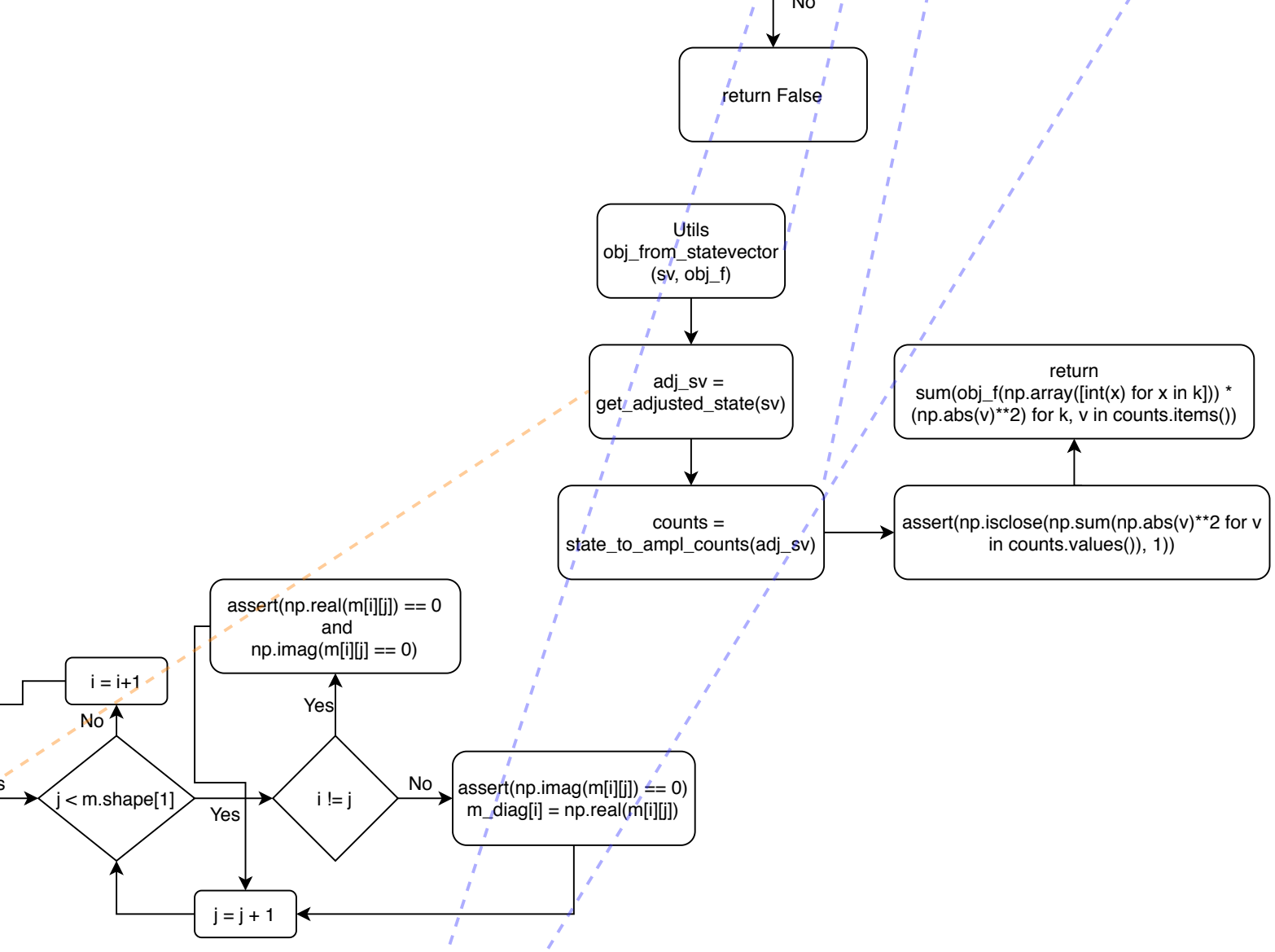




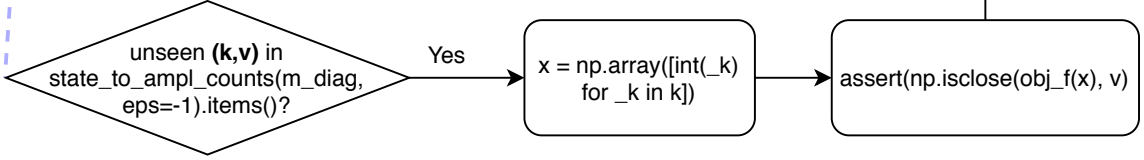
iationalQuantumOptimiz

Provides functions for optimizing the variational forms

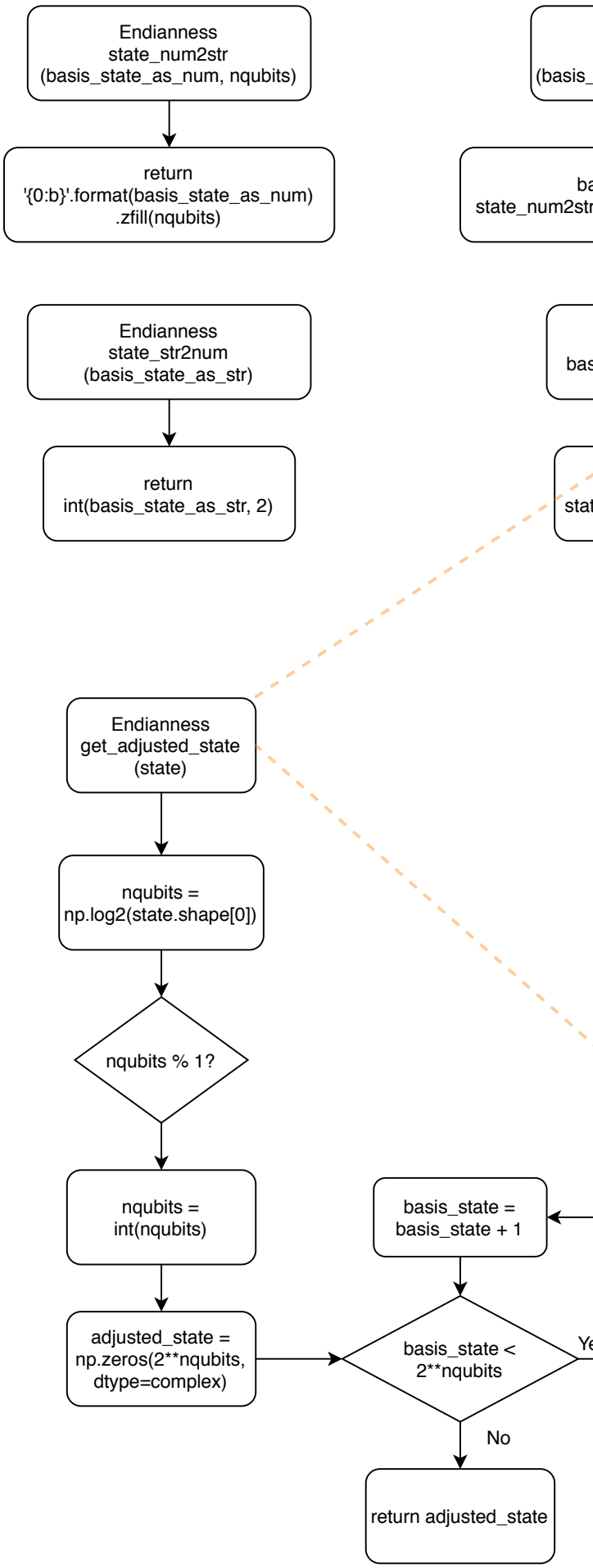


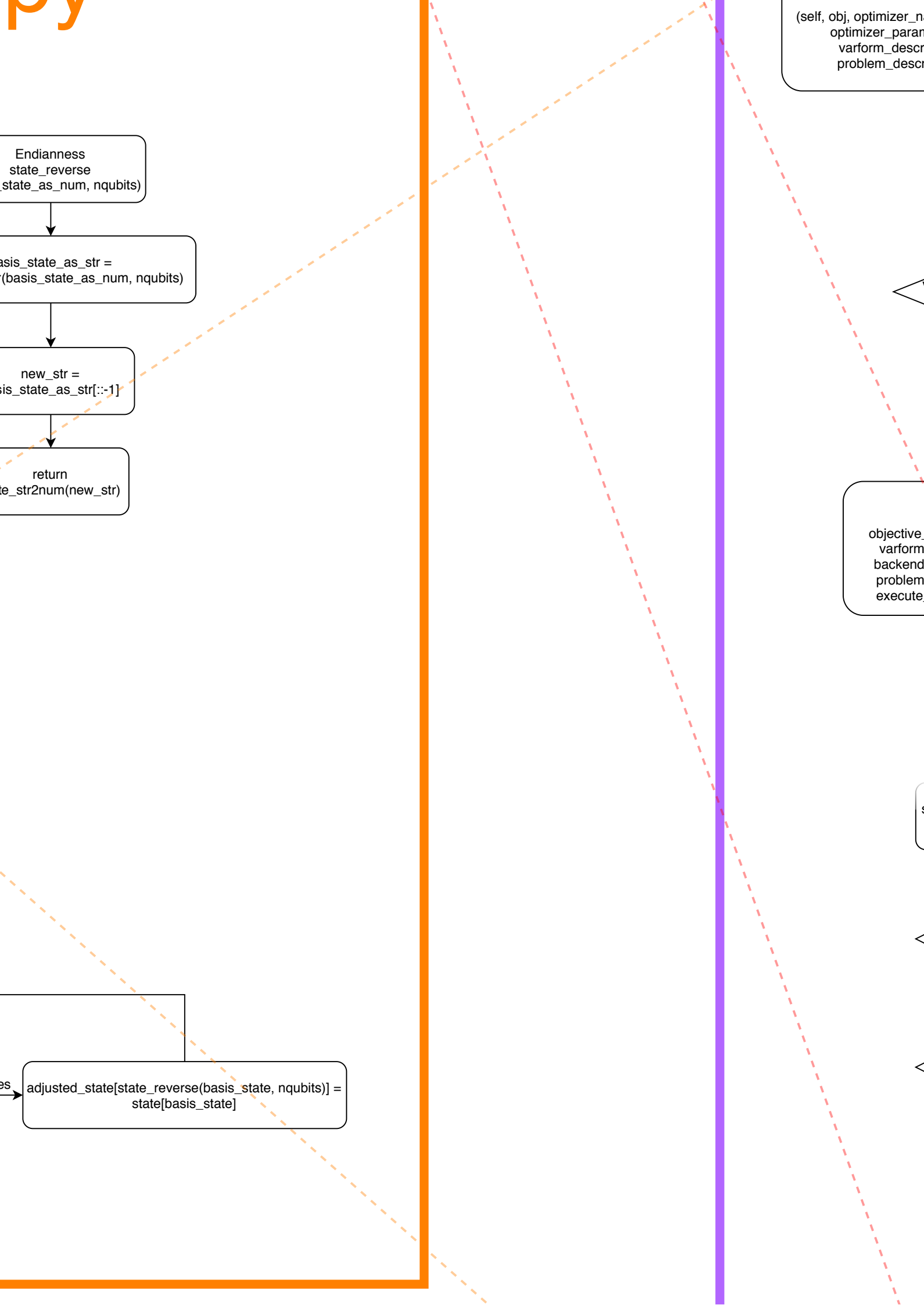

No
return False
Utils
obj_from_statevector
(sv, obj_f)
adj_sv =
get_adjusted_state(sv)
return
sum(obj_f(np.array([int(x) for x in k])) *
(np.abs(v)**2) for k, v in counts.items())
counts =
state_to_ampl_counts(adj_sv)
assert(np.isclose(np.sum(np.abs(v)**2 for v
in counts.values()), 1))
assert(np.real(m[i][j]) == 0
and
np.imag(m[i][j] == 0)
i = i+1
No
Yes
j < m.shape[1]
Yes
i != j
No
assert(np.imag(m[i][j]) == 0)
m_diag[i] = np.real(m[i][j])
j = j + 1

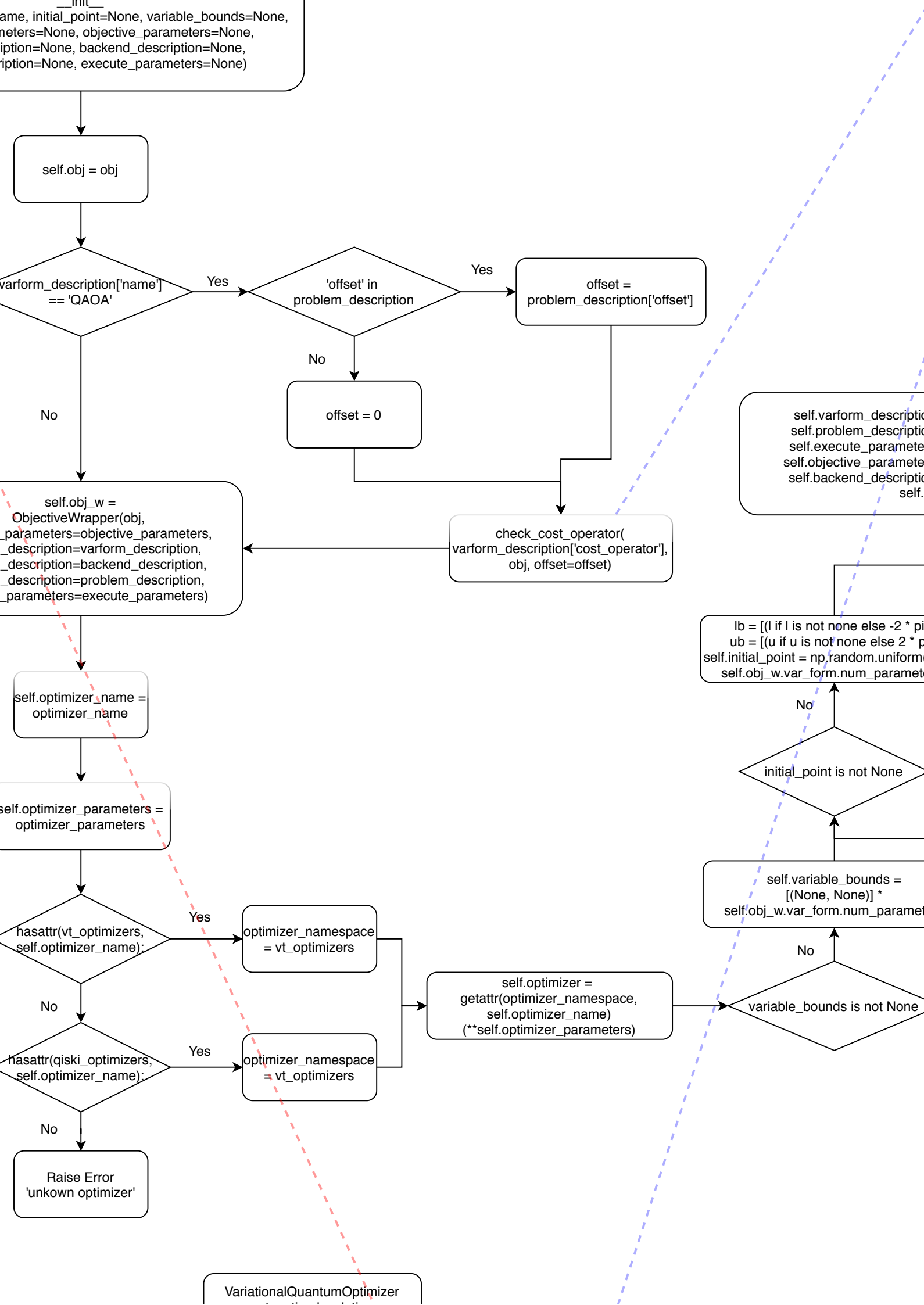
zer.py

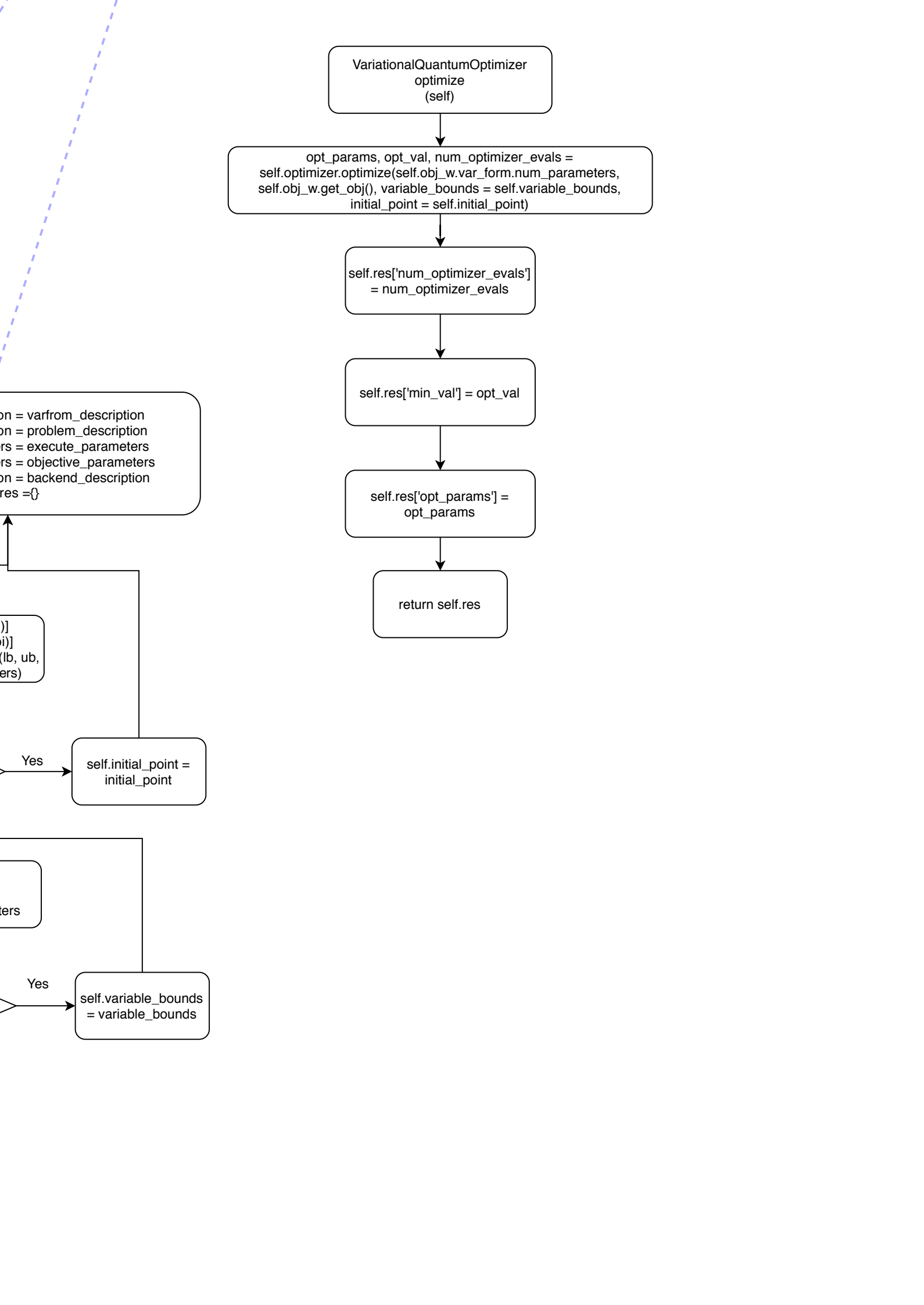

unseen **(k,v)** in state_to_ampl_counts(m_diag, eps=-1).items()?
Yes
x = np.array([int(_k) for _k in k])
assert(np.isclose(obj_f(x), v)

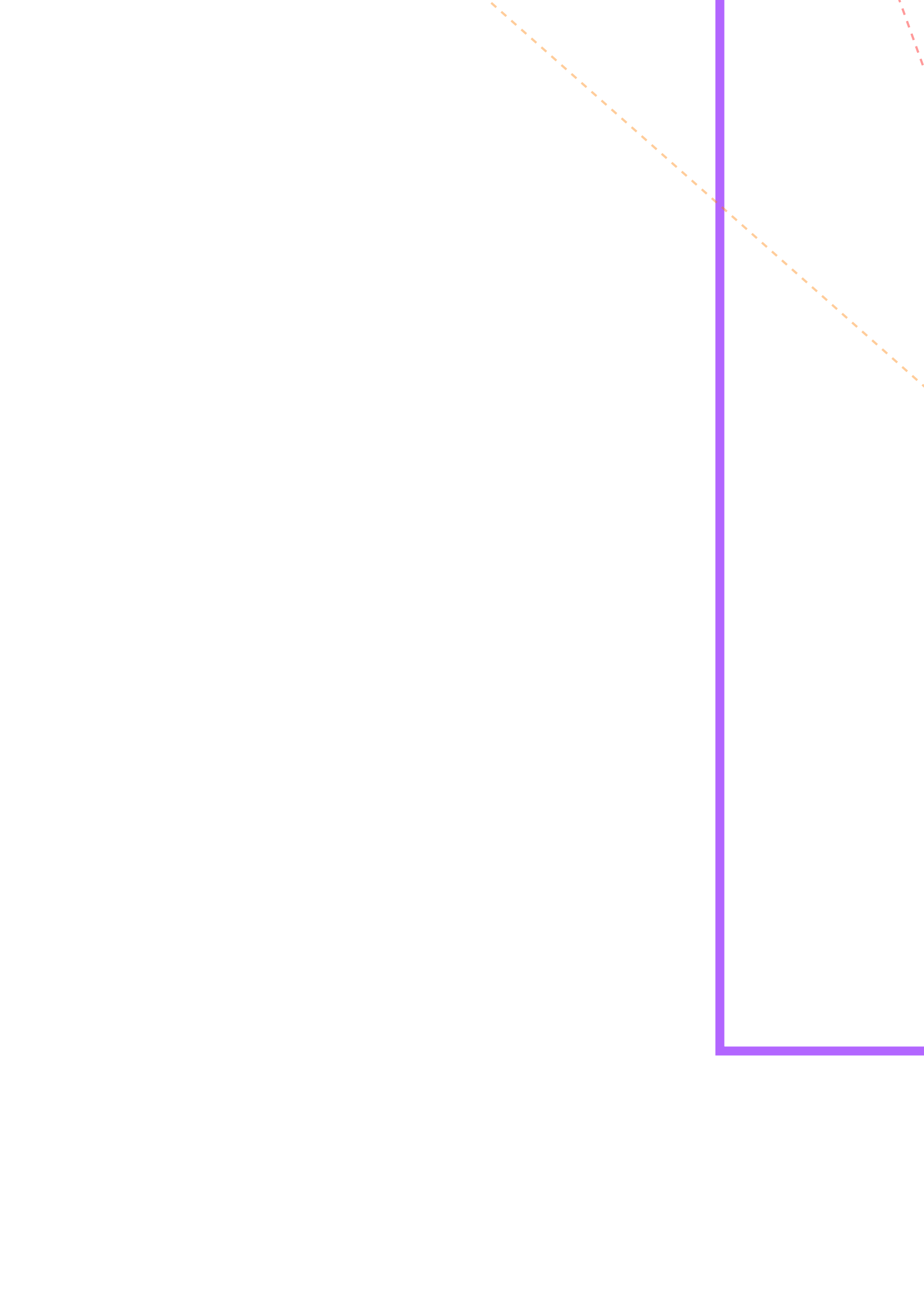
Endianness

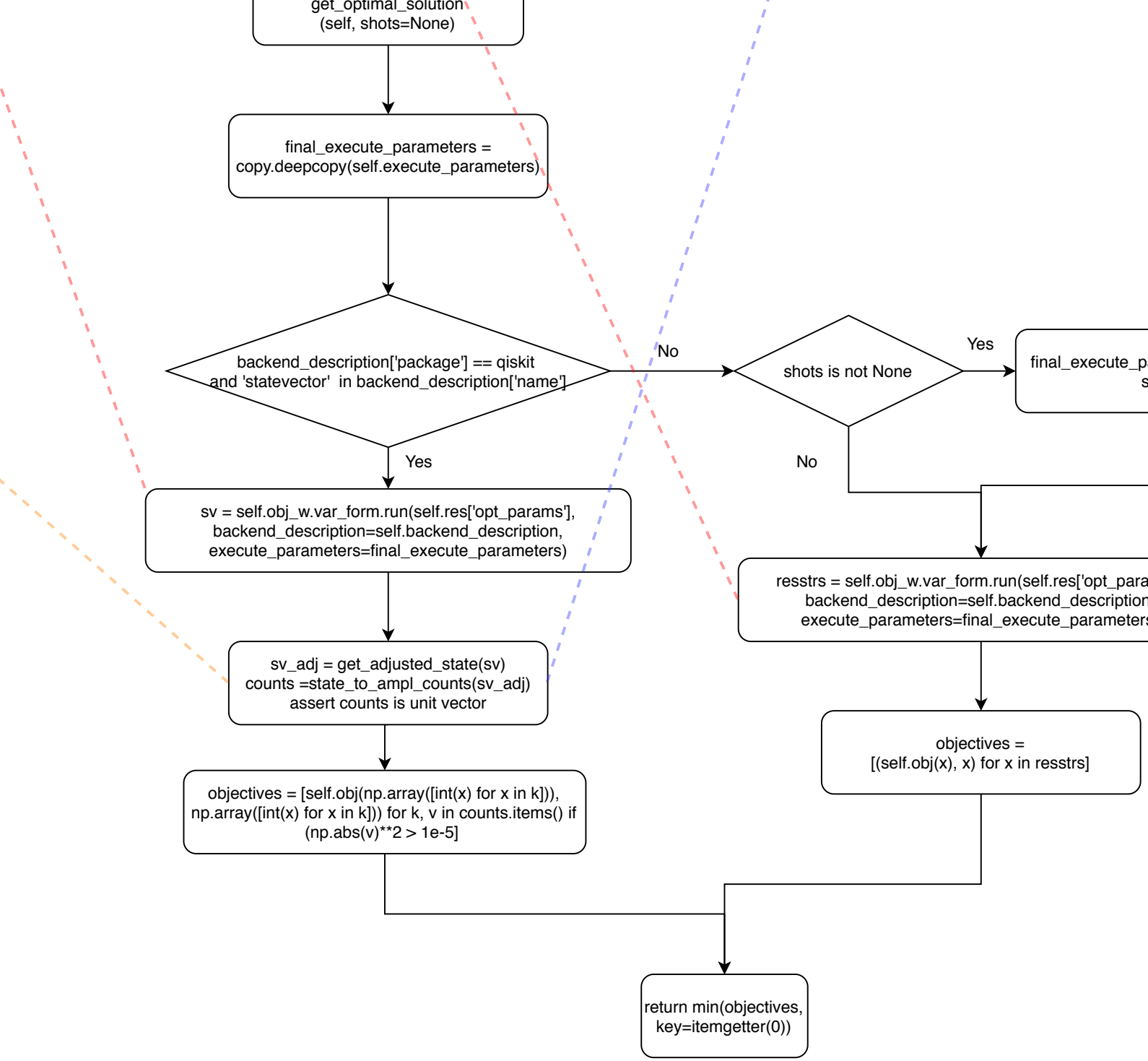

Endianness
state_num2str
(basis_state_as_num, nqubits)
return
'{0:b}'.format(basis_state_as_num)
.zfill(nqubits)
Endianness
state_str2num
(basis_state_as_str)
return
int(basis_state_as_str, 2)
Endianness
get_adjusted_state
(state)
nqubits =
np.log2(state.shape[0])
nqubits % 1?
nqubits =
int(nqubits)
adjusted_state =
np.zeros(2**nqubits,
dtype=complex)
basis_state =
basis_state + 1
basis_state <
2**nqubits
No
return adjusted_state

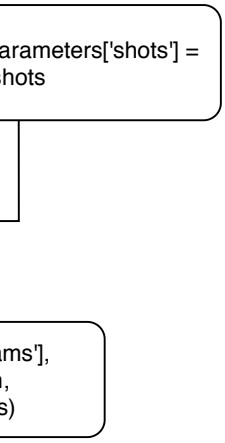

(self, obj, optimizer_n
optimizer_param
varform_descr
problem_descr
Endianness
state_reverse
_state_as_num, nqubits)
asis_state_as_str =
r(basis_state_as_num, nqubits)
new_str =
sis_state_as_str[::-1]
return
te_str2num(new_str)
objective_
varform
backend
problem
execute_
es
adjusted_state[state_reverse(basis_state, nqubits)] =
state[basis_state]


__init__
ame, initial_point=None, variable_bounds=None,
meters=None, objective_parameters=None,
iption=None, backend_description=None,
ription=None, execute_parameters=None)
self.obj = obj
varform_description['name']
== 'QAOA'
Yes
'offset' in
problem_description
Yes
offset =
problem_description['offset']
No
offset = 0
No
self.obj_w =
ObjectiveWrapper(obj,
_parameters=objective_parameters,
_description=varform_description,
_description=backend_description,
_description=problem_description,
_parameters=execute_parameters)
check_cost_operator(
varform_description['cost_operator'],
obj, offset=offset)
self.varform_descriptio
self.problem_descriptio
self.execute_paramete
self.objective_paramete
self.backend_descriptio
self.
self.optimizer_name =
optimizer_name
lb = [(l if l is not none else -2 * pi
ub = [(u if u is not none else 2 * p
self.initial_point = np.random.uniform
self.obj_w.var_form.num_paramet
No
initial_point is not None
self.optimizer_parameters =
optimizer_parameters
self.variable_bounds =
[(None, None)] *
self.obj_w.var_form.num_paramet
No
hasattr(vt_optimizers,
self.optimizer_name):
Yes
optimizer_namespace
= vt_optimizers
self.optimizer =
getattr(optimizer_namespace,
self.optimizer_name)
(**self.optimizer_parameters)
variable_bounds is not None
No
hasattr(qiski_optimizers,
self.optimizer_name):
Yes
optimizer_namespace
= vt_optimizers
No
Raise Error
'unkown optimizer'
VariationalQuantumOptimizer


VariationalQuantumOptimizer
optimize
(self)
opt_params, opt_val, num_optimizer_evals =
self.optimizer.optimize(self.obj_w.var_form.num_parameters,
self.obj_w.get_obj(), variable_bounds = self.variable_bounds,
initial_point = self.initial_point)
self.res['num_optimizer_evals']
= num_optimizer_evals
self.res['min_val'] = opt_val
self.res['opt_params'] =
opt_params
return self.res
on = varfrom_description
on = problem_description
rs = execute_parameters
rs = objective_parameters
on = backend_description
res ={}
)]
i)]
(lb, ub,
ers)
Yes
self.initial_point =
initial_point
ters
Yes
self.variable_bounds
= variable_bounds




get_optimal_solution
(self, shots=None)
final_execute_parameters =
copy.deepcopy(self.execute_parameters)
backend_description['package'] == qiskit
and 'statevector' in backend_description['name']
No
shots is not None
Yes
final_execute_p
s
Yes
No
sv = self.obj_w.var_form.run(self.res['opt_params'],
backend_description=self.backend_description,
execute_parameters=final_execute_parameters)
resstrs = self.obj_w.var_form.run(self.res['opt_para
backend_description=self.backend_description
execute_parameters=final_execute_parameters
sv_adj = get_adjusted_state(sv)
counts =state_to_ampl_counts(sv_adj)
assert counts is unit vector
objectives =
[(self.obj(x), x) for x in resstrs]
objectives = [self.obj(np.array([int(x) for x in k])),
np.array([int(x) for x in k])) for k, v in counts.items() if
(np.abs(v)**2 > 1e-5]
return min(objectives,
key=itemgetter(0))


arameters['shots'] =
hots
ams'],
,
s)