`__init__` and RandomStabilizerState

```
# Create an empty stabilizer state as used by
# the RandomStabilizerState function. It has
# K =\mathbb{F}^n_2 and has q(x) = 0 for all x.
def __init__(self, n, k):
    # define K from RandomStabilizerState algorithm (page 16)
    self.n = n
    self.k = k
    self.h = np.zeros(n)         # in F_2^n
    self.G = np.identity(n)      # in F_2^{n×n}
    self.Gbar = np.identity(n)   # = (G^{-1})^T

    # define q to be zero for all x
    self.Q = 0                   # in Z_8
    self.D = np.zeros(k)         # in {0,2,4,6}^k
    self.J = np.zeros((k, k))    # in {0,4}^{k×k}, symmetric
```

```
# cache probability distributions for stabilizer state dimension k
# in a dictionary, with a key for each n
dDists = {}

@classmethod
def randomStabilizerState(cls, n, provide_d=False):
    # ensure probability distribution is available for this n
    if n not in cls.dDists:
        # compute distribution given by equation 79 on page 15
        def eta(d):
            if d == 0: return 0

            product = 1
            for a in range(1, d+1):
                product *= (1 - 2**(d - n - a))
                product /= (1 - 2**(-a))
            return 2**(-d*(d+1)/2) * product

        # collect numerators
        dist = np.array([])
        for d in range(n):
            dist = np.append(dist, [eta(d)], 0)

        # normalize
        norm = sum(dist)
        dist /= norm

        # cache result
        cls.dDists[n] = dist

    # sample d from distribution
    sample = 1-np.random.random()  # sample from (0.0, 1.0]
    d = 0
    cumulative = 0
    while cumulative < sample:
        cumulative += cls.dDists[n][d]
        d += 1
    k = n - d
```

```
39
40      # pick random X in F_2^{d,n} with rank d
41      while True:
42          X = np.random.random_integers(0, 1, (d, n))
43
44          if np.linalg.matrix_rank(X) == d: break
45
46      # create the state object. __init__ gives the correct properties
47      state = StabilizerState(n, k)
48
49      for a in range(d):
50          # lazy shrink with a'th row of X
51          state.shrink(X[a], 0, lazy=True)
52
53          # reset state's k after shrinking
54          state.k = k
55
56      # now K = ker(X) and is in standard form
57
58      state.h = np.random.random_integers(0, 1, n)
59      state.Q = np.random.random_integers(0, 7)
60      state.D = 2*np.random.random_integers(0, 3, state.k)
61
62      state.J = np.zeros((state.k, state.k))
63      for a in range(state.k):
64          state.J[a, a] = 2*state.D[a] % 8
65          for b in range(a):
66              state.J[a, b] = 4*np.random.random_integers(0, 1)
67              state.J[b, a] = state.J[b, a]
68
69      if not provide_d: return state
70      else: return state, d
```