0.0.1 System: The 2D Ising model

```
import numpy as np
from numba.experimental import jitclass
from numba import int64
integer = int64
```

```
__all__ = ['IsingModel']
@jitclass([
    ('spins',  integer[:,:]),
    ('L',  integer),
    ('sweep_steps', integer),
    ('E',  integer),
    ('Ev', integer),
    ('dE', integer),
    ('i',  integer),
    ('j',  integer)
])
class IsingModel:
    def __init__(self, spins):
        shape = np.shape(spins)
        if len(shape) != 2 or shape[0] != shape[1]:
            raise ValueError('IsingModel spin array is not a square.')
        self.spins = spins
        self.L = shape[0]
        self.sweep_steps = shape[0]**2
        self.E = self.energy()
        self.Ev = self.E
        self.dE = 0
        self.i = 0
        self.j = 0
    def state(self):
        return (self.spins.copy(),)
    def state_names(self):
        return ('spins',)
    def copy(self):
        return IsingModel(*self.state())
    def energy_bins(self):
        Ex = 2 * self.L**2
        ΔE = 4
        Es = np.arange(-Ex, Ex + ΔE + 1, ΔE)
        # Penultimate indices are not attainable energies
        return np.delete(Es, [1, -3])
    def neighbors(self, i, j):
        return np.array([
            self.spins[i-1, j],
            self.spins[(i+1) % self.L, j],
            self.spins[i, j-1],
            self.spins[i, (j+1) % self.L],
        ])
    def energy(self):
        E = 0
        for i in range(self.L):
            for j in range(self.L):
                E -= np.sum(self.spins[i, j] * self.neighbors(i, j))
        return E // 2
    def propose(self):
        i, j = np.random.randint(self.L), np.random.randint(self.L)
```

```
            self.i, self.j = i, j
            dE = 2 * np.sum(self.spins[i, j] * self.neighbors(i, j))
            self.dE = dE
            self.Ev = self.E + dE
        def accept(self):
            self.spins[self.i, self.j] *= -1
            self.E = self.Ev
```