1 Maximum-entropy reconstruction

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal, misc
```

Given a measurement D of a system I_0, we wish to reconstruct I_0 from D. We take the Bayesian approach and find the I that maximizes $P(I|D) \propto P(D|I)P(I)$, where the likelihood is related to the error in D due to I and the prior is $P(I) = \exp(\lambda S(I))$, where S is some notion of the entropy of an image. If $D = T(I_0)$, then $P(D|I) = \exp(-E(T(I), D))$, for some metric E on D's. From the form of the objective $\ln P(D|I) + \ln(I) = -E(T(I), D) + \lambda S(I)$, we see that the entropy acts as a regularizer.

The astronomers have both I and D as intensity lists $\mathbb{Z}_N \to \mathbb{R}_{\geq 0}$, where

$$S(I) = \sum_i I_i \ln I_i$$

and

$$E(D', D) = \sum_i \exp\left(-\frac{(D'_i - D_i)^2}{2\sigma_i}\right)$$

(with empirical σ_i). The transformation T is convolution with the point spread function of the telescope.

To perform the optimization, we need not only the functions S and E, but also a procedure to modify candidate images.

```python
def maxent_objective(D, I, λ): # For minimization
    return D.E(I.transform()) - λ*I.S
```

```python
# Greedy for now, but can be something like simulated annealing
def maxent(D, I, λ = 1, N = 1_000_000, ε = 1e-8):
    f0 = np.inf
    f = maxent_objective(D, I, λ)
    i = 0
    while ε < f0 - f and i < N:
        I.propose()
        fv = maxent_objective(D, I, λ)
        if fv < f: # Greedy
            f0, f = f, fv
            I.accept()
        i += 1
```

```
        if i % (N // 20) == 0:
            print("Maxent: {} / {}".format(i, N))

    print("Maxent: i: {} / {}, Δf: {}".format(i, N, f0 - f))
    return I, i, f
```

1.1 Example: 1D Point from Gaussian

```
class Gaussian:
    def __init__(self, μ=0, σ=1):
        self.μ = μ
        self.σ = σ
    def E(self, Gv):
        return (self.μ - Gv.μ)**2 + (self.σ - Gv.σ)**2

class Point:
    def __init__(self, x=0):
        self.x = x
        self.dx = 0
        self.S = self.entropy()
    def propose(self):
        self.dx = np.random.rand() - 0.5
        self.S = self.entropy()
    def accept(self):
        self.x += self.dx
        self.dx = 0 # Idempotence
    def entropy(self):
        return -(self.x - 10)**2 # Opposite from true max
    def transform(self):
        return Gaussian(μ = self.x + self.dx)
```

```
D = Gaussian(0, 1)
I = Point(9)
I0, _, _ = maxent(D, I, λ = 1, N = 1_000_000, ε = 1e-4)
I0.x
```

```
Maxent: i: 1000000 / 1000000, Δf: 0.011299906795997572
```

```
4.268883578482481
```

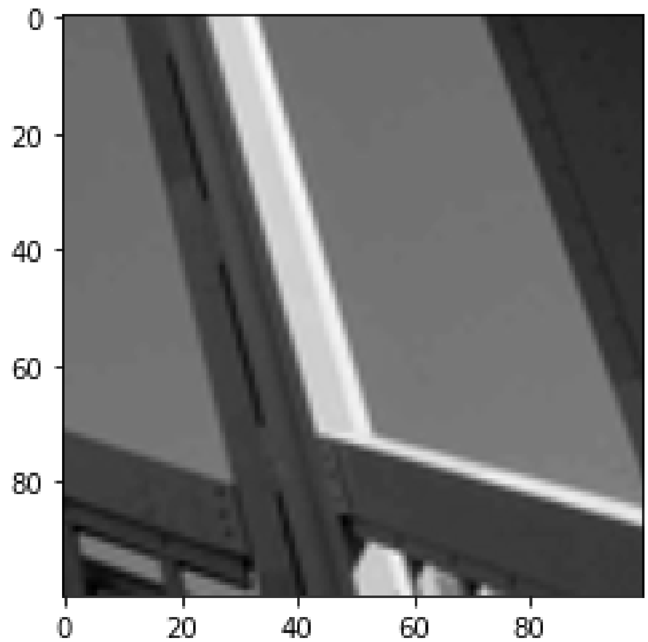
Results are sort of near 5. The optimization is terrible, but you get the idea: the entropy shifts the best point away from zero.

1.2 Example: Image from PSF convolution (measurement)

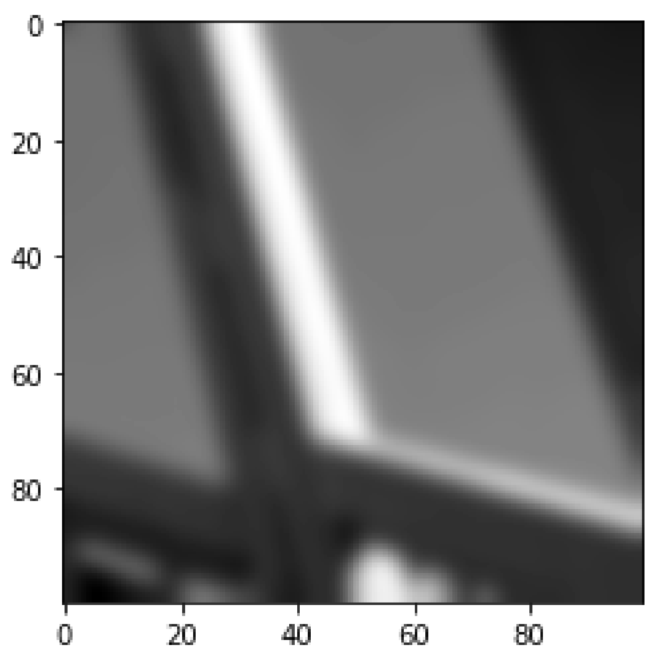
```
class DImage:
    def __init__(self, I):
        self.I = I
    def E(self, Dv):
        return np.sum((self.I - Dv.I)**2)

class IImage:
    def __init__(self, I):
        self.I = I
        self.w, self.h = np.shape(I)
        self.i, self.j = 0, 0
        self.I0 = 0
        self.Iv = 0
        n = int(np.sqrt(self.w * self.h))
        self.G = signal.windows.gaussian(n // 10, n // 50)
        self.S = self.entropy()
    def propose(self):
        self.i, self.j = np.random.randint(self.w), np.random.randint(self.h)
        I0 = self.I[self.i, self.j]
        self.I0 = I0
        self.Iv = np.random.randint(256)
        self.S = self.entropy()
    def accept(self):
        self.I[self.i, self.j] = self.Iv
    def entropy(self):
        return -np.sum(np.log(self.I + 1)) + self.I0*np.log(self.I0 + 1) - self.Iv*np.log(self.Iv + 1)
    def transform(self):
        Iv = self.I.copy()
        Iv[self.i, self.j] = self.Iv
        return DImage(signal.sepfir2d(Iv, self.G, self.G))
```

```
I = IImage(misc.ascent()[250:350,250:350])
plt.imshow(I.I, cmap='gray');
```



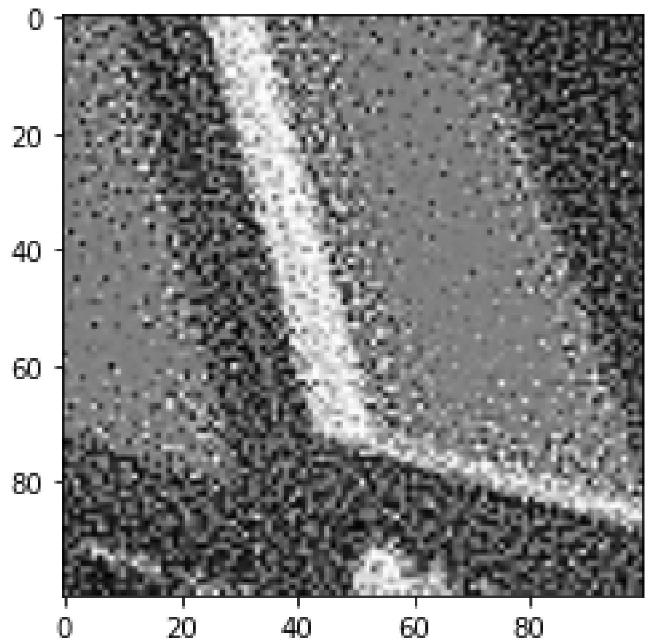
```
plt.imshow(I.transform().I, cmap='gray');
```



```python
I = IImage(misc.ascent()[250:350,250:350])
Iguess = IImage(128 * np.ones((I.w, I.h), dtype=int))
I0, _, _ = maxent(I.transform(), Iguess, λ = 1e-9, N = 1_000_000, ε = 1e-20) # Just do the max iterations
```

```
Maxent: 50000 / 1000000
Maxent: 100000 / 1000000
Maxent: 150000 / 1000000
Maxent: 200000 / 1000000
Maxent: 250000 / 1000000
Maxent: 300000 / 1000000
Maxent: 350000 / 1000000
Maxent: 400000 / 1000000
Maxent: 450000 / 1000000
Maxent: 500000 / 1000000
Maxent: 550000 / 1000000
Maxent: 600000 / 1000000
Maxent: 650000 / 1000000
Maxent: 700000 / 1000000
Maxent: 750000 / 1000000
Maxent: 800000 / 1000000
Maxent: 850000 / 1000000
Maxent: 900000 / 1000000
Maxent: 950000 / 1000000
Maxent: 1000000 / 1000000
Maxent: i: 1000000 / 1000000, Δf: 4.847227362683043
```

```python
plt.imshow(I0.I, cmap='gray');
```



0
20
40
60
80
0
20
40
60
80