Complexity Lab 3: Epsilon Machines

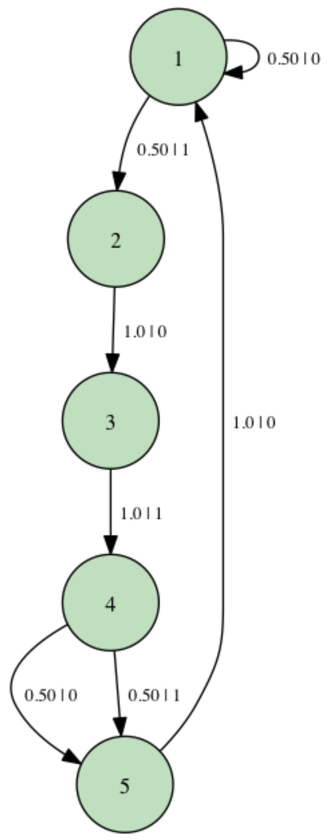
[redacted] [redacted] [redacted]

last edited on June 24, 2011 03:48 PM by autoplectic

```
%hide
```

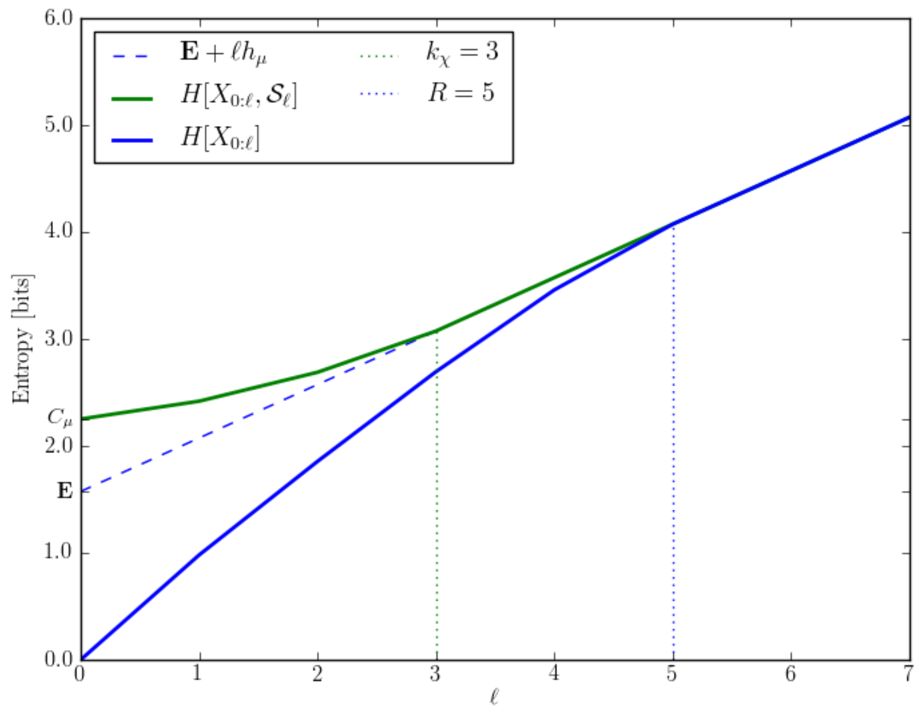
In this lab we will be looking at both block entropy and block-state entropy curves. In the latter the states will be the causal states of the epsilon machine. This will help us explore crypticity in processes. Here we will have the same processes available to us as in the previous labs, plus a more complicated one known as the Noisy Random Phase Slip, *nrps*:

```
nrps.draw()
```



Let's look at its block and block-state entropy curves:

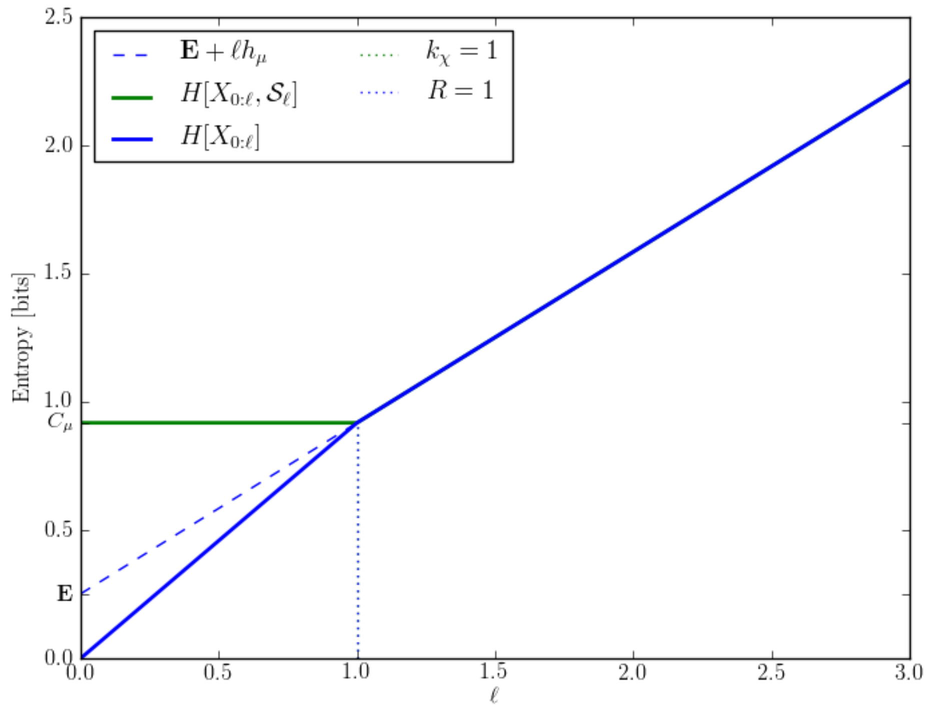
```
block_entropy(nrps, length=7)
```



Here we find a very generic looking pair of curves. It is order 5 Markov and order 3 cryptic. We can see that the statistical complexity is around 2.2 bits and the excess entropy is approximately 1.6 bits, making the crypticity somewhere around 0.6 bits.

Now you should plot the *goldenmean* process, length=3 is a good parameter to use:

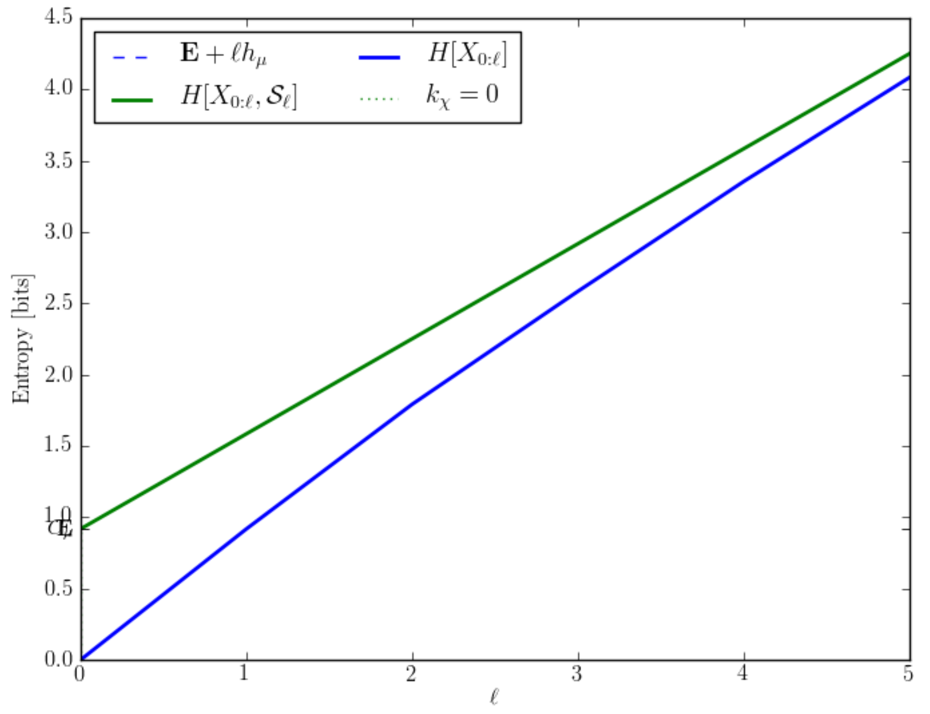
```
block_entropy(goldenmean, length=3)
```



We see here that the Markov and cryptic orders are the same: one.

Let us now look at the even process. Here length=5 is a reasonable choice:

```
block_entropy(even, length=5)
```

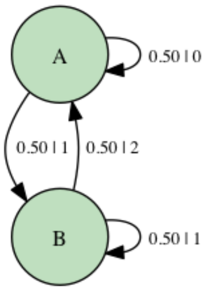


In this plot you can not see the dashed blue line, **E** + ℓ h_μ, because it is underneath the green block-state entropy curve. We also see that because of this C_μ and **E** are identical, and that the process has a cryptic order of 0. See again that the Markov order is infinite for this process.

Let us now turn our attention to a generator known as the *mystery_machine.*

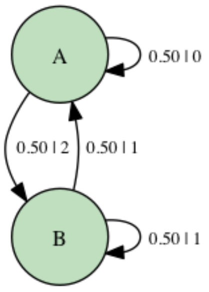
First let's see what it looks like:

```
mystery_machine.draw()
```



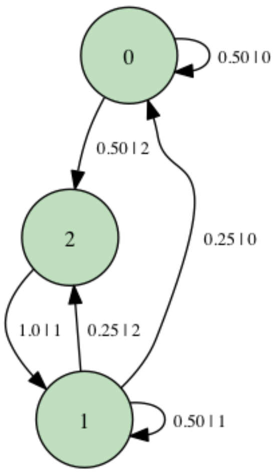
Wh can reverse the direction of time by flipping the arrows and renormalizing using the *reverse* method. After reversing, let's take a look at what we get:

```
reverse_mystery_machine = mystery_machine.reverse()
reverse_mystery_machine.draw()
```



Did you notice that it was non-unifilar (look at state B)? Let's convert this to the epsilon machine by using the *build_eM* function, and since we are not interested in the transient states we ask the function not to produce them. After building the epsilon machine, let's look at it:

```
rem = build_eM(reverse_mystery_machine, transients=False)
rem.draw()
```



Surprisingly, the reverse epsilon machine of the mystery machine has three states whereas the mystery machine had two states. This tells us that some processes require more computational resources to predict in one direction than in the other. We can see this further by looking at the statistical complexity of both epsilon machines:

```
print "Mystery Machine: statistical complexity is", mystery_machine.statistical_complexity(), "bits"
print "Reverse Mystery Machine: statistical complexity is", rem.statistical_complexity(), "bits"
```

evaluate

```
    Mystery Machine: statistical complexity is 1.0 bits
    Reverse Mystery Machine: statistical complexity is 1.5 bits
```