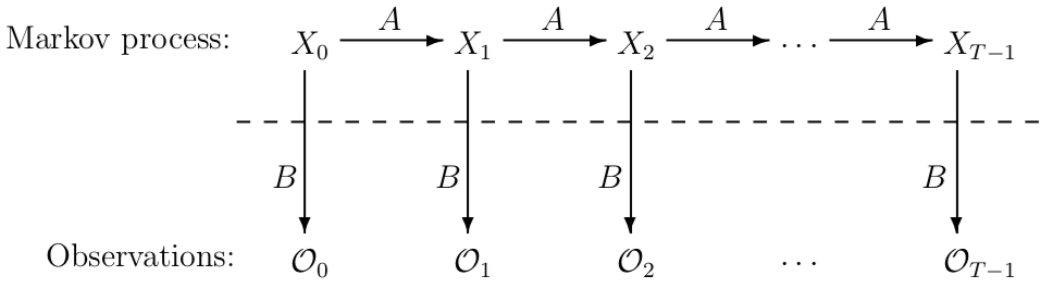
blang HMM Report - annualTemps

1 Setup

1.1 Model Parameters

- States:
  - 'H' : Hot
  - 'C' : Cold
- Transition Matrix $\mathbf{A} =$ $\begin{matrix} & H & C \\ H & 0.7 & 0.3 \\ C & 0.4 & 0.6 \end{matrix}$
- Observation Matrix $\mathbf{B} =$ $\begin{matrix} & S & M & L \\ H & 0.1 & 0.4 & 0.5 \\ C & 0.7 & 0.2 & 0.1 \end{matrix}$
- Initial Distribution Vector $\pi =$ $\begin{matrix} H & C \\ [0.6 & 0.4] \end{matrix}$

1.2 Graphical Representation



2 Problem 1: Given the model $\lambda = (A, B, \pi)$ and a sequence of observations $\mathbb{O}$, find $P(\mathbb{O}|\lambda)$.

2.1 blang Code & output

```
model HMM {
    //Number of States for the Markov Process
    param   int             nLatentStates ?: 2

    //Sequence of observations, fixing it solves HMM problem 2
    random List<IntVar>      observations ?: fixedIntList(0,1,0,2)
    //Sets length of the process to size of observations
    random List<IntVar>      states ?: latentIntList(observations.size)

    //Initial Dist. Vector (notation: pi)
    param DenseSimplex           initialDistribution ?: fixedSimplex(0.6, 0.4)
    //Transition Matrix (notation A)
    param DenseTransitionMatrix  transitionProbabilities ?: fixedTransitionMatrix(#[#[0.7, 0.3], #[0.4, 0.6]])
    //Observation Matrix (notation: B)
    param DenseTransitionMatrix  emissionProbabilities ?: fixedTransitionMatrix(#[#[0.1, 0.4, 0.5], #[0.7, 0.2, 0.1]])

    laws {
        // Uses the Built-In Markov Chain module to generate the latent states of
        //  the markov process.
        states | initialDistribution, transitionProbabilities
            ~ MarkovChain(initialDistribution, transitionProbabilities)

        for (int obsIdx : 0 ..< observations.size) {
            observations.get(obsIdx) |
            emissionProbabilities,
            IntVar curIndic = states.get(obsIdx)
            ~ Categorical(emissionProbabilities.row(curIndic))
        }
    }
}
```

```
Problems  Javadoc  Declaration  Console
<terminated> New_configuration (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (Oct 18, 2018, 5:18:17 AM)
WARNING: Bare Repository has neither a working tree, nor an index.
Preprocessing started
4 samplers constructed with following prototypes:
IntScalar sampled via: []
Categorical sampled via: [CategoricalSampler]
Categorical sampled via: []
Sampling started
Change of measure complete [iter=134, Z=-4.633800222992608]
Normalization constant estimate: -4.633800222992608
Final rejuvenation started
Preprocessing time: 70.83 ms
Sampling time: 1.320 s
executionMilliseconds : 1396
outputFolder : /Users/sahand/Desktop/blang/workspace/blang-BMFH/results/all/2018-10-18-05-18-17-vF1ih3ub.exec
```

$$\mathbf{P(\mathbb{O}|\lambda) = e^{-4.633800222992608} \approx 0.00972}$$

3 Given $\lambda = (A, B, \pi)$ and an observation sequence $\mathbb{O}$, find an optimal state sequence for the underlying Markov process.

3.1 blang Code

```
model HMM {
    //Number of States for the Markov Process
    param   int             nLatentStates ?: 2

    //Sequence of observations, fixing it solves HMM problem 2
    random List<IntVar>      observations ?: fixedIntList(0,1,0,2)
    //Sets length of the process to size of observations
    random List<IntVar>      states ?: latentIntList(observations.size)

    //Initial Dist. Vector (notation: pi)
    param DenseSimplex           initialDistribution ?: fixedSimplex(0.6, 0.4)
    //Transition Matrix (notation A)
    param DenseTransitionMatrix  transitionProbabilities ?: fixedTransitionMatrix(#[#[0.7, 0.3], #[0.4, 0.6]])
    //Observation Matrix (notation: B)
    param DenseTransitionMatrix  emissionProbabilities ?: fixedTransitionMatrix(#[#[0.1, 0.4, 0.5], #[0.7, 0.2, 0.1]])

    laws {
        // Uses the Built-In Markov Chain module to generate the latent states of
        //  the markov process.
        states | initialDistribution, transitionProbabilities
            ~ MarkovChain(initialDistribution, transitionProbabilities)

        for (int obsIdx : 0 ..< observations.size) {
            observations.get(obsIdx) |
            emissionProbabilities,
            IntVar curIndic = states.get(obsIdx)
            ~ Categorical(emissionProbabilities.row(curIndic))
        }
    }
}
```

3.2 Posterior Plot