CPSC 540 Assignment 3 (due Friday March 12 at midnight)

USING ONE LATE DAY

1. Name(s): Ricky Ma, Patricia Ye
2. Student ID(s): 82943424, 18139162

1 Markov Chains

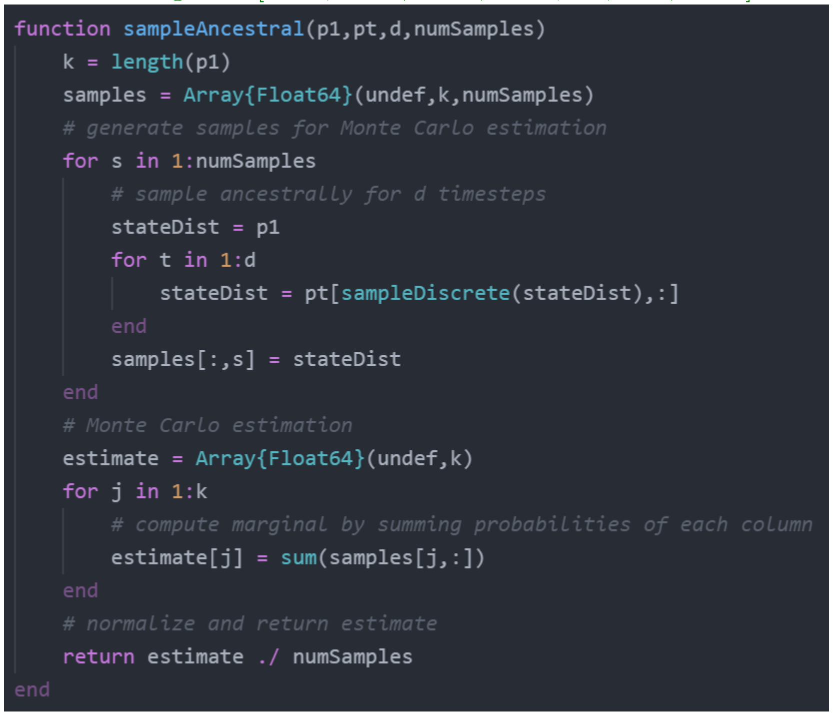
1.1 Inference with Discrete States

The function *example_markovChain.jl* loads the initial state probabilities and transition probabilities for a Markov chain model,

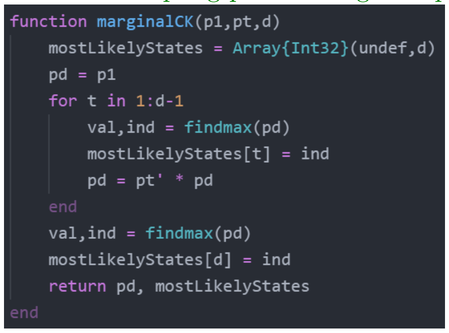
$$p(x_1, x_2, \dots, x_d) = p(x_1) \prod_{j=2}^{d} p(x_j \mid x_{j-1}),$$

corresponding to the "grad student Markov chain" from class.

1. Write a function, *sampleAncestral*, that uses ancestral sampling to sample a sequence x from this Markov chain of length d. Hand in this code and report the univariate marginal probabilities for time 50 using a Monte Carlo estimate based on 10000 samples.
   Hint: you can use *sampleDiscrete* in *misc.jl* to sample from a discrete probability mass function using the inverse transform method.
   Univariate marginals: [0.014, 0.388, 0.017, 0.005, 0.1, 0.08, 0.396]

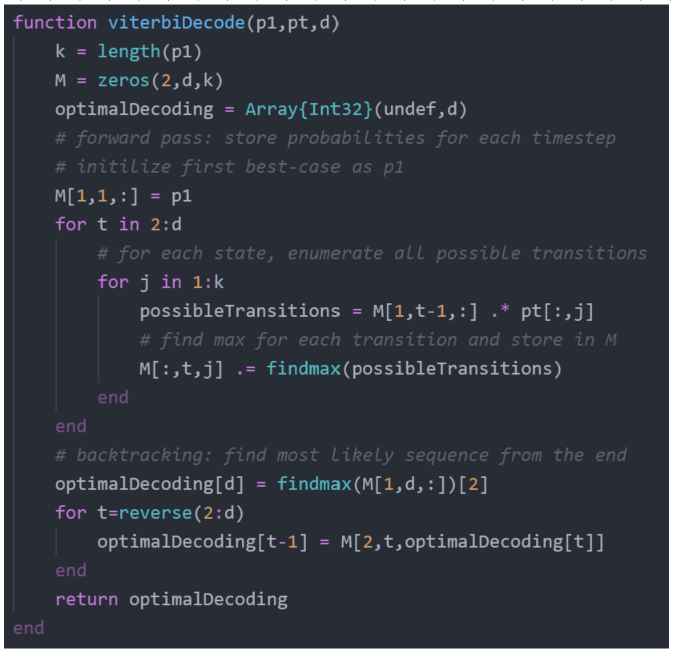


```
function sampleAncestral(p1,pt,d,numSamples)
    k = length(p1)
    samples = Array{Float64}(undef,k,numSamples)
    # generate samples for Monte Carlo estimation
    for s in 1:numSamples
        # sample ancestrally for d timesteps
        stateDist = p1
        for t in 1:d
            stateDist = pt[sampleDiscrete(stateDist),:]
        end
        samples[:,s] = stateDist
    end
    # Monte Carlo estimation
    estimate = Array{Float64}(undef,k)
    for j in 1:k
        # compute marginal by summing probabilities of each column
        estimate[j] = sum(samples[j,:])
    end
    # normalize and return estimate
    return estimate ./ numSamples
end
```

2. Write a function, *marginalCK*, that uses the CK equations to compute the exact univariate marginals up to a given time d. Hand in this code, report all exact univariate marginals at time 50, and report how this differs from the marginals in the previous question.
   Exact univariate marginals: [0.014, 0.386, 0.017, 0.005, 0.104, 0.084, 0.389]
   Error between approximation and exact: [0.0, 0.002, 0.0, 0.0, 0.004, 0.004, 0.007]
   As you can see, the difference between the approximation using ancestral sampling and the exact probabilities using the CK equations is very small. We can conclude that, with 10000 samples, using ancestral sampling provides a good approximation of the actual marginals.


```
function marginalCK(p1,pt,d)
    mostLikelyStates = Array{Int32}(undef,d)
    pd = p1
    for t in 1:d-1
        val,ind = findmax(pd)
        mostLikelyStates[t] = ind
        pd = pt' * pd
    end
    val,ind = findmax(pd)
    mostLikelyStates[d] = ind
    return pd, mostLikelyStates
end
```

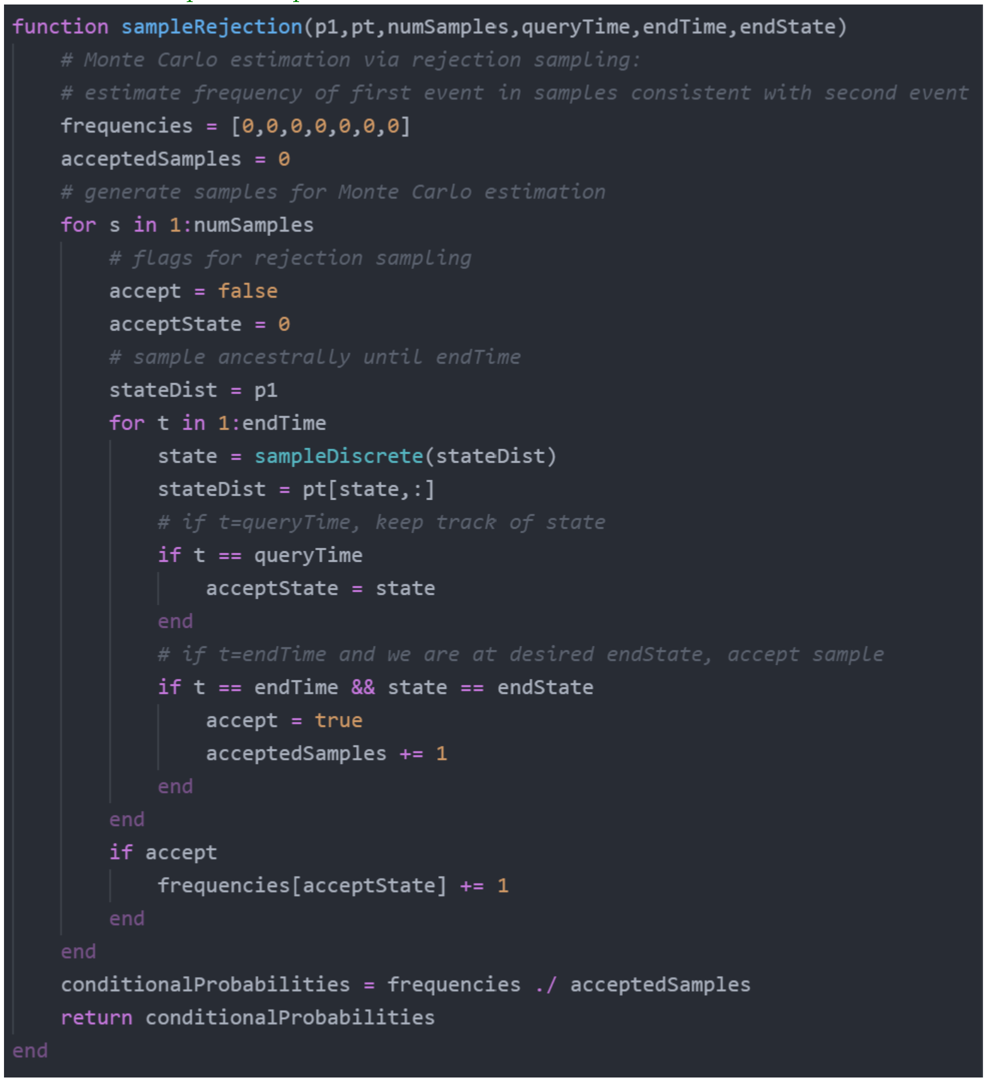
3. What is the state c with highest marginal probability, $p(x_j = c)$, for each time j?
   [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7]
   $c = 2$ *if* $j < 50$ and $c = 7$ *if* $j \geq 50$

4. Write a function, *viterbiDecode*, that uses the Viterbi decoding algorithm for Markov chains to find the optimal decoding up to a time d. Hand in this code and report the optimal decoding of the Markov chain up to time 50 and up to 100.
   viterbiDecode(p1, pt, 50) = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
   viterbiDecode(p1, pt, 100) = [2, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]


```
function viterbiDecode(p1,pt,d)
    k = length(p1)
    M = zeros(2,d,k)
    optimalDecoding = Array{Int32}(undef,d)
    # forward pass: store probabilities for each timestep
    # initilize first best-case as p1
    M[1,1,:] = p1
    for t in 2:d
        # for each state, enumerate all possible transitions
        for j in 1:k
            possibleTransitions = M[1,t-1,:] .* pt[:,j]
            # find max for each transition and store in M
            M[:,t,j] .= findmax(possibleTransitions)
        end
    end
    # backtracking: find most likely sequence from the end
    optimalDecoding[d] = findmax(M[1,d,:])[2]
    for t=reverse(2:d)
        optimalDecoding[t-1] = M[2,t,optimalDecoding[t]]
    end
    return optimalDecoding
end
```

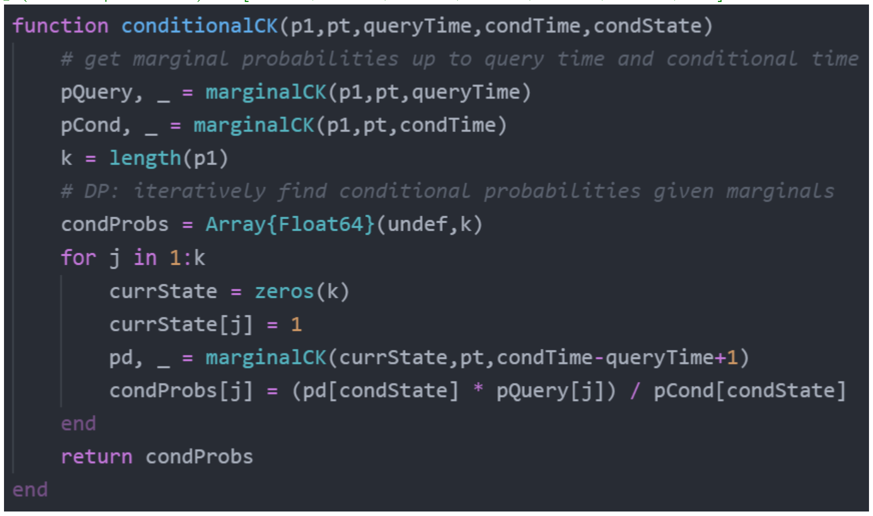
5. Report all the univariate conditional probabilities at time 50 if the student starts in grad school, $p(x_{50} = c \mid x_1 = 3)$ for all c. Hint: you should be able to do this by changing the input to the CK equations.
   $p(x_{50} = c|x_1 = 3) = [0.008, 0.231, 0.01, 0.008, 0.183, 0.17, 0.389]$

6. Report for all c the univariate conditional probabilities $p(x_5 = c \mid x_{10} = 6)$ ("where you were likely to be 5 years after graduation if you ended up in academia after 10 years") obtained using a Monte Carlo estimate based on 10000 samples and rejection sampling. Also report the number of samples accepted among the 10000 samples.
   $p(x_5 = c|x_{10} = 6) = [0.0, 0.029, 0.251, 0.058, 0.16, 0.503, 0.0]$
   Number of accepted samples: 382



```
function sampleRejection(p1,pt,numSamples,queryTime,endTime,endState)
    # Monte Carlo estimation via rejection sampling:
    # estimate frequency of first event in samples consistent with second event
    frequencies = [0,0,0,0,0,0,0]
    acceptedSamples = 0
    # generate samples for Monte Carlo estimation
    for s in 1:numSamples
        # flags for rejection sampling
        accept = false
        acceptState = 0
        # sample ancestrally until endTime
        stateDist = p1
        for t in 1:endTime
            state = sampleDiscrete(stateDist)
            stateDist = pt[state,:]
            # if t=queryTime, keep track of state
            if t == queryTime
                acceptState = state
            end
            # if t=endTime and we are at desired endState, accept sample
            if t == endTime && state == endState
                accept = true
                acceptedSamples += 1
            end
        end
        if accept
            frequencies[acceptState] += 1
        end
    end
    conditionalProbabilities = frequencies ./ acceptedSamples
    return conditionalProbabilities
end
```

7. Give code implementing a dynamic programming approach to exactly compute $p(x_5 = c \mid x_{10} = 6)$, and report the exact values for all c.
$p(x_5 = c|x_{10} = 6) = [0.002, 0.034, 0.248, 0.058, 0.161, 0.497, 0.0]$



```
function conditionalCK(p1,pt,queryTime,condTime,condState)
    # get marginal probabilities up to query time and conditional time
    pQuery, _ = marginalCK(p1,pt,queryTime)
    pCond, _ = marginalCK(p1,pt,condTime)
    k = length(p1)
    # DP: iteratively find conditional probabilities given marginals
    condProbs = Array{Float64}(undef,k)
    for j in 1:k
        currState = zeros(k)
        currState[j] = 1
        pd, _ = marginalCK(currState,pt,condTime-queryTime+1)
        condProbs[j] = (pd[condState] * pQuery[j]) / pCond[condState]
    end
    return condProbs
end
```

8. Why is $p(x_j = 7 \mid x_{10} = 6)$ equal to zero for all j less than 10?
State 7 is an absorbing state, or terminal state, that represents the probability of a student being deceased. This means that once this state is reached, it is not possible to move to a different state. Hence, if we know that $x_{10} = 6$, we know that up until $j = 10$, the absorbing state has not been reached. In other words, $p(x_j = 7 \mid x_{10} = 6)$ for all $j < 10$.

Hint: for some of the quesitons you may find it helpful to use a k by d matrix M to represent a dynamic programming table

1.2 Inference with Gaussian States

Consider a continuous-state Markov chain where the initial distribution is given by

$$x_0 \sim \mathcal{N}(m_0, v_0^2),$$

and the transition distributions for $j > 1$ are given by

$$x_j \mid x_{j-1} \sim \mathcal{N}(w_j x_{j-1} + m_j, v_j^2).$$

This model could be used to model an object moving through $\mathbb{R}$.[1] Because of the Gaussian assumptions, this defines a joint Gaussian distribution over the variables while the marginal distributions are also Gaussian. For a generic $j > 1$, derive the form of the marginal distribution of x_j, expressing the marginal parameters μ_j and σ_j recursively in terms of the parameters μ_{j-1} and σ_{j-1} of the previous marginal, $p(x_{j-1}) \sim \mathcal{N}(\mu_{j-1}, \sigma_{j-1}^2)$.

Hint: You can use Theorem 4.4.1 of Murphy's book. From Theorem 4.4.1 of Murphy's book, in this scenario, substitute $x_j = y$ and $x_{j-1} = x$. So

$$\begin{aligned} p(x_{j-1}) &= \mathcal{N}(\mu_{j-1}, \sigma_{j-1}^2) \\ p(x_j \mid x_{j-1}) &= \mathcal{N}(w_j x_{j-1} + m_j, v_j^2) \\ &\text{Using (4.126) in Murphy} \\ p(x_j) &= \mathcal{N}(w_j x_{j-1} + m_j, v_j + w_j^2 \sigma_{j-1}^2) \end{aligned}$$

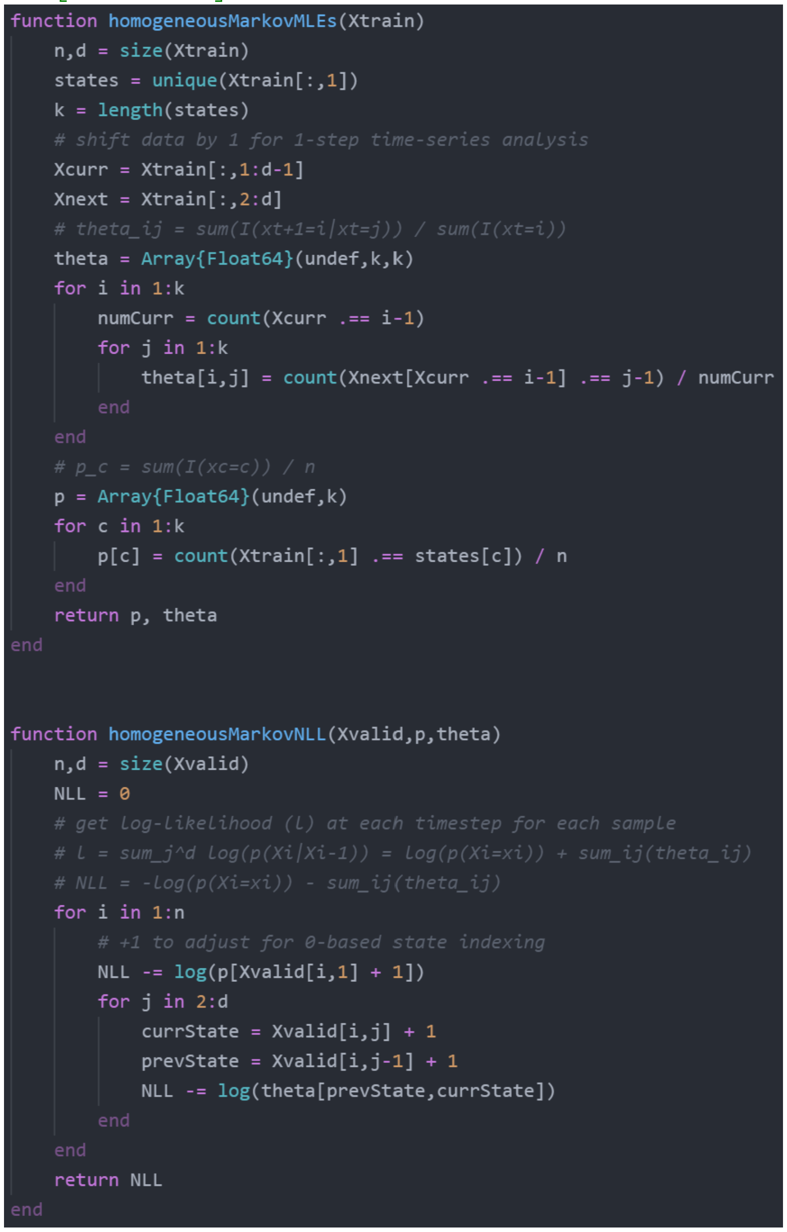
So $\mu_j = w_j x_{j-1} + m_j$, $\sigma_j^2 = v_j + w_j^2 \sigma_{j-1}^2$

[1] In practical applications like object tracking, we typically have that the states x_j are 2- or 3-dimensional if we are modeling an object moving through space, or even higher-dimensional if we are modeling things like stock prices.

1.3 Learning with Discrete States

If you run *example_rain* it will: load the Vancouver rain data set, split it into a training and validation set, fit an indepdendent (and homogeneous) Bernoulli model to the training set, and then compute the negative log-likeilhood (NLL) of this model on the validation set (a lower validation NLL means a better fit). As discussed in class, we expect that a Markov chain could be a better model of this dataset.

1. Give code for finding the MLE for the initial probabilities and transition probabilities in a homogeneous Markov chain, and report the MLE values for the training set.
   $\pi = [0.674, 0.326]$
   $$\theta = \begin{bmatrix} 0.758 & 0.242 \\ 0.375 & 0.625 \end{bmatrix}$$



```julia
function homogeneousMarkovMLEs(Xtrain)
    n,d = size(Xtrain)
    states = unique(Xtrain[:,1])
    k = length(states)
    # shift data by 1 for 1-step time-series analysis
    Xcurr = Xtrain[:,1:d-1]
    Xnext = Xtrain[:,2:d]
    # theta_ij = sum(I(xt+1=i|xt=j)) / sum(I(xt=i))
    theta = Array{Float64}(undef,k,k)
    for i in 1:k
        numCurr = count(Xcurr .== i-1)
        for j in 1:k
            theta[i,j] = count(Xnext[Xcurr .== i-1] .== j-1) / numCurr
        end
    end
    # p_c = sum(I(xc=c)) / n
    p = Array{Float64}(undef,k)
    for c in 1:k
        p[c] = count(Xtrain[:,1] .== states[c]) / n
    end
    return p, theta
end


function homogeneousMarkovNLL(Xvalid,p,theta)
    n,d = size(Xvalid)
    NLL = 0
    # get log-likelihood (L) at each timestep for each sample
    # L = sum_j^d log(p(Xi|Xi-1)) = log(p(Xi=xi)) + sum_ij(theta_ij)
    # NLL = -log(p(Xi=xi)) - sum_ij(theta_ij)
    for i in 1:n
        # +1 to adjust for 0-based state indexing
        NLL -= log(p[Xvalid[i,1] + 1])
        for j in 2:d
            currState = Xvalid[i,j] + 1
            prevState = Xvalid[i,j-1] + 1
            NLL -= log(theta[prevState,currState])
        end
    end
    return NLL
end
```

2. Report the NLL of the Markov chain model on the validation set. NLL = 8966.21375

2 Directed Acyclic Graphical Models

2.1 D-Separation

Consider a directed acyclic graphical (DAG) model with the following graph structure:

Assuming that the conditional independence properties are faithful to the graph, using d-separation briefly explain why the following are true or false:

1. $H \perp I$.
   False, H and I are dependent because they share a common parent, A.
2. $H \perp I \mid A$.
   True, observing the common parent A blocks the path between H and I, making them independent.
3. $H \perp I \mid J$.
   False, H and I are dependent. By observing a common child, knowing a parent makes the other more/less likely.
4. $H \perp I \mid A, J$.
   False, H and I are dependent, with the same reason as above.
5. $C \perp I$.
   True, because C and I do not have any common parents and all children are unobserved.
6. $C \perp I \mid J$.
   False, C and I are dependent. By observing a common child J, knowing a parent makes the other more/less likely.
7. $C \perp I \mid H$.
   False, knowing I gives information about D since they have a common unobserved parent. D and C share a common child H, which when observed creates a dependence between C and I.
8. $C \perp I \mid A, H$.
   True, the common parent between D and I, A, is observed. Therefore, knowing I does not give information about D, and in turn, C.
9. $C \perp I \mid E, H, J$.
   True, observing E and H blocks the path from C to J. Therefore, the path from I to C is blocked even though their common child is observed.

2.2 Exact Inference

Consider a directed acyclic graphical (DAG) model with the following graph structure: Assume that all variables are binary and that we use the following parameterization of the network:

$$\begin{aligned}
&p(A=1)=0.7\\
&p(B=1\mid A=0)=0.8\\
&p(B=1\mid A=1)=1.0\\
&p(C=1)=0.8\\
&p(D=1\mid B=0)=0.8\\
&p(D=1\mid B=1)=0.6\\
&p(E=1\mid B=0,C=0)=0.3\\
&p(E=1\mid B=0,C=1)=0.7\\
&p(E=1\mid B=1,C=0)=0.4\\
&p(E=1\mid B=1,C=1)=0.5\\
&p(F=1\mid A=0)=0.5\\
&p(F=1\mid A=1)=0.9\\
&p(G=1\mid E=0,F=0)=0.5\\
&p(G=1\mid E=0,F=1)=0\\
&p(G=1\mid E=1,F=0)=0.1\\
&p(G=1\mid E=1,F=1)=0.1
\end{aligned}$$

Compute the following quantities:

1. $p(A=0)$.
   $p(A=0)=1-p(A=1)=1-0.7=0.3$
2. $p(B=1\mid A=0)$.
   $p(B=1\mid A=0)=0.8$
3. $p(B=1)$.
   $p(B=1)=p(B=1|A=0)p(A=0)+p(B=1|A=1)p(A=1)=0.8*0.3+1.0*0.7=0.94$
4. $p(D=1)$.
   $p(B=0)=1-p(B=1)=1-0.94=0.06$
   $p(D=1)=p(D=1|B=0)p(B=0)+p(D=1|B=1)p(B=1)=0.8*0.06+0.6*0.94=0.612$
5. $p(B=1\mid D=1)$.
   $p(B=1|D=1)=\frac{p(D=1|B=1)p(B=1)}{p(D=1)}=\frac{0.6*0.94}{0.612}=0.922$
6. $p(B=1\mid C=1)$.
   B and C are independent since common children are unobserved.
   Therefore, $p(B=1\mid C=1)=p(B=1)=0.94$
7. $p(B=1\mid A=0,C=1,F=1)$.
   Knowing A blocks the path from B to F. The path from C to B is also blocked.
   Therefore, $p(B=1\mid A=0,C=1,F=1)=p(B=1\mid A=0)=0.8$

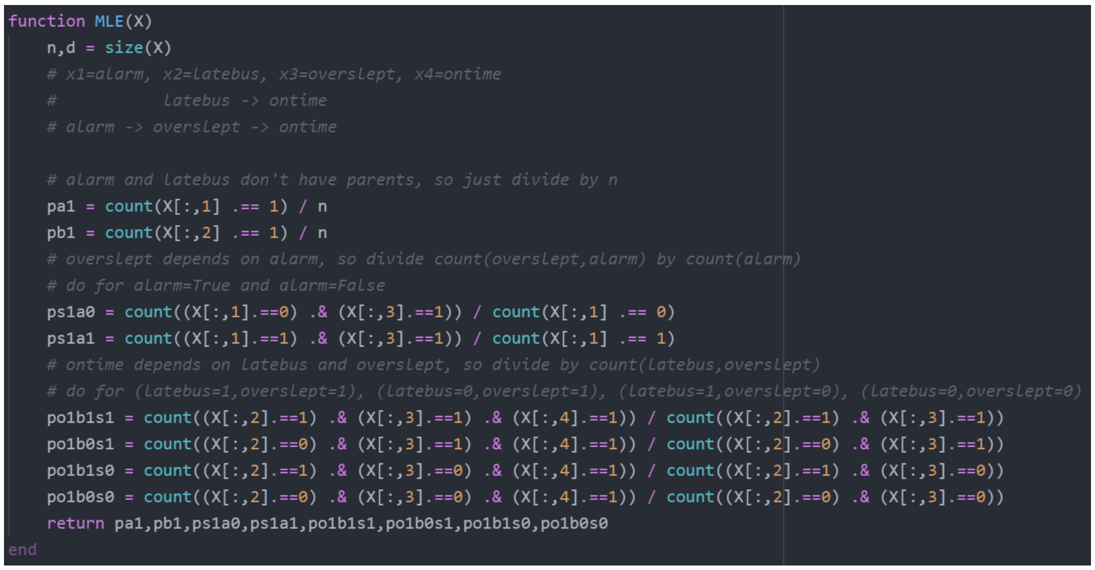
Hints: some of the above quantities can be read from the table, some require using that probabilities sum to 1, some require the marginalization rule, some require Bayes rule, some require using [conditional] independence, and some will be simplified using calculations from previous sub-questions.

2.3 Learning in DAGs

The file *onTime.jld* contains a matrix X containing samples from the following DAG model:[2] The first column of X is the "alarm" variable, the second is "bus late", the third is "over-slept", and the last is "on time".

1. Assuming the DAG structure above, give code for computing the MLE of the parameters in the model, and report the MLE values up to 2 decimal places.

$$
\begin{aligned}
p(\text{busLate} = 1) &= 0.18,\\
p(\text{alarm} = 1) &= 0.91,\\
p(\text{overslept} = 1 \mid \text{alarm} = 1) &= 0.10,\\
p(\text{overslept} = 1 \mid \text{alarm} = 0) &= 0.89,\\
p(\text{onTime} = 1 \mid \text{busLate} = 1, \text{overslept} = 1) &= 0.03,\\
p(\text{onTime} = 1 \mid \text{busLate} = 1, \text{overslept} = 0) &= 0.15,\\
p(\text{onTime} = 1 \mid \text{busLate} = 0, \text{overslept} = 1) &= 0.26,\\
p(\text{onTime} = 1 \mid \text{busLate} = 0, \text{overslept} = 0) &= 0.91
\end{aligned}
$$

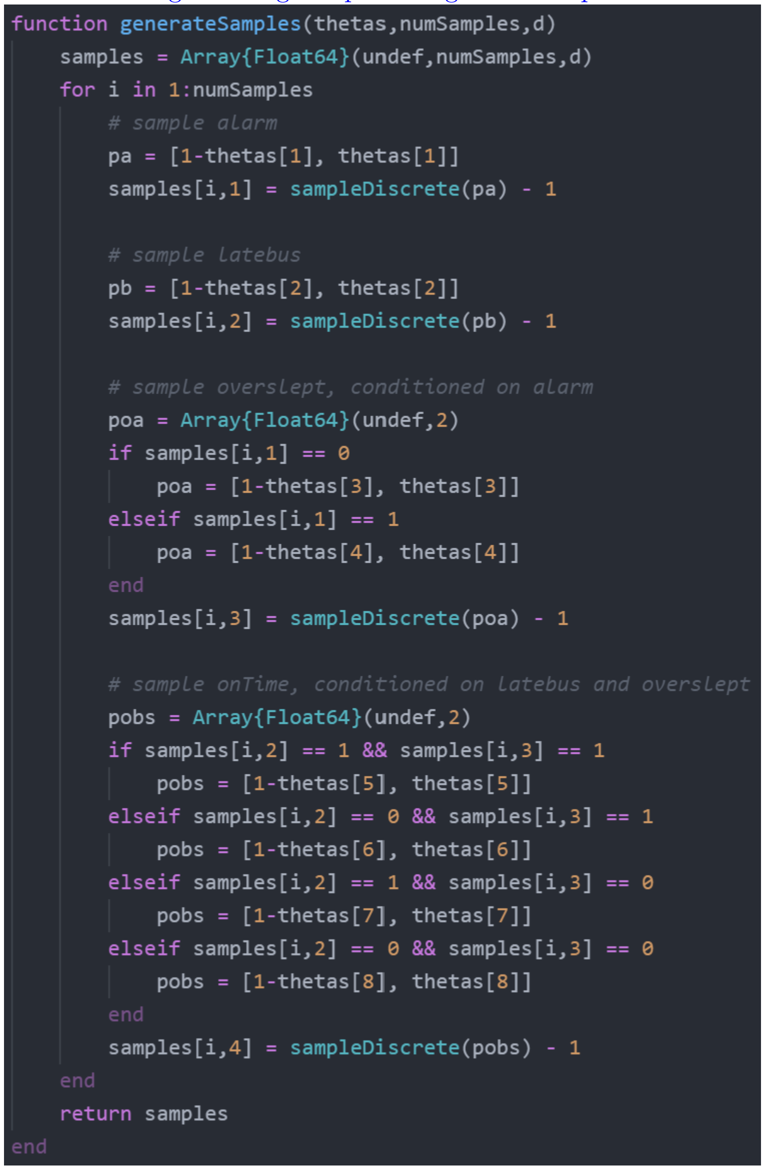


```
function MLE(X)
    n,d = size(X)
    # x1=alarm, x2=latebus, x3=overslept, x4=ontime
    #           latebus -> ontime
    # alarm -> overslept -> ontime

    # alarm and latebus don't have parents, so just divide by n
    pa1 = count(X[:,1] .== 1) / n
    pb1 = count(X[:,2] .== 1) / n
    # overslept depends on alarm, so divide count(overslept,alarm) by count(alarm)
    # do for alarm=True and alarm=False
    ps1a0 = count((X[:,1].==0) .& (X[:,3].==1)) / count(X[:,1] .== 0)
    ps1a1 = count((X[:,1].==1) .& (X[:,3].==1)) / count(X[:,1] .== 1)
    # ontime depends on latebus and overslept, so divide by count(latebus,overslept)
    # do for (latebus=1,overslept=1), (latebus=0,overslept=1), (latebus=1,overslept=0), (latebus=0,overslept=0)
    po1b1s1 = count((X[:,2].==1) .& (X[:,3].==1) .& (X[:,4].==1)) / count((X[:,2].==1) .& (X[:,3].==1))
    po1b0s1 = count((X[:,2].==0) .& (X[:,3].==1) .& (X[:,4].==1)) / count((X[:,2].==0) .& (X[:,3].==1))
    po1b1s0 = count((X[:,2].==1) .& (X[:,3].==0) .& (X[:,4].==1)) / count((X[:,2].==1) .& (X[:,3].==0))
    po1b0s0 = count((X[:,2].==0) .& (X[:,3].==0) .& (X[:,4].==1)) / count((X[:,2].==0) .& (X[:,3].==0))
    return pa1,pb1,ps1a0,ps1a1,po1b1s1,po1b0s1,po1b1s0,po1b0s0
end
```

[2]from here: `https://www.uib.no/en/rg/ml/119695/bayesian-networks`

2. Give code for generating samples using the MLE parameters.



```julia
function generateSamples(thetas,numSamples,d)
    samples = Array{Float64}(undef,numSamples,d)
    for i in 1:numSamples
        # sample alarm
        pa = [1-thetas[1], thetas[1]]
        samples[i,1] = sampleDiscrete(pa) - 1

        # sample latebus
        pb = [1-thetas[2], thetas[2]]
        samples[i,2] = sampleDiscrete(pb) - 1

        # sample overslept, conditioned on alarm
        poa = Array{Float64}(undef,2)
        if samples[i,1] == 0
            poa = [1-thetas[3], thetas[3]]
        elseif samples[i,1] == 1
            poa = [1-thetas[4], thetas[4]]
        end
        samples[i,3] = sampleDiscrete(poa) - 1

        # sample onTime, conditioned on latebus and overslept
        pobs = Array{Float64}(undef,2)
        if samples[i,2] == 1 && samples[i,3] == 1
            pobs = [1-thetas[5], thetas[5]]
        elseif samples[i,2] == 0 && samples[i,3] == 1
            pobs = [1-thetas[6], thetas[6]]
        elseif samples[i,2] == 1 && samples[i,3] == 0
            pobs = [1-thetas[7], thetas[7]]
        elseif samples[i,2] == 0 && samples[i,3] == 0
            pobs = [1-thetas[8], thetas[8]]
        end
        samples[i,4] = sampleDiscrete(pobs) - 1
    end
    return samples
end
```

3. The combination $(0, 1, 0, 1)$ does not occur in the training data, so if we fit the MLE general discrete distribution to this data we would get that it has a probability of 0. This is in contrast to the true model above, where we have $p(0, 1, 0, 1) = 0.0004$. Does your model give a better or worse estimate of the true proaibility of this event than the general discrete distribution? Why do you think it gives a better/worse estimate?

$$\begin{aligned} p(0,1,0,1) &= p(alarm = 0)p(bus = 1)p(slept = 0|alarm = 0)p(onTime = 1|bus = 1, slept = 0) \\ &= 0.09 * 0.18 * 0.11 * 0.15 \\ &= 0.0002673 \end{aligned}$$

This model gives a better estimate because none of the conditional probabilities are zero, so the joint probability of $(0, 1, 0, 1)$ is small, but not zero.

3 Very-Short Answer Questions

Give a short and concise 1-sentence answer to the below questions.

1. What is the difference between computing marginals and computing the stationary distribution of a Markov chain. Marginalization is the process of computing the probability of being in a state at a single time-step. Computing the stationary distribution means computing the probability of being in a state as time goes to infinity.

2. What is the inverse transform method used for? The inverse transform method: sample $u \sim U(0, 1)$ and return $F^{-1}(u)$. We can use it to generate samples from simple 1D probability distributions using their (inverse) CDFs.

3. Describe how we could use ancestral sampling to sample from the joint density over x and y defined by a Gaussian discriminant analysis model. In GDA, $p(x, y) = p(y)p(x|y)$, meaning features x_j are dependent on y. Hence, we can first sample y, then sample each x_j given the sampled y.

4. Suppose you had a black box that could generate IID samples from a distribution. Describe how you could use a Monte Carlo method to approximate $p(x \leq c)$ for this distribution. We can repeatedly sample from the black box, and just divide the number of samples where $x \leq c$ by the total number of times we sampled to get an approximation of $p(x \leq c)$.

5. What is the cost of generating a sample from a Markov chain of length d with k possible states for each time? What is the cost of decoding? It costs k to generate a sample for a single timestep, so for a Markov chain of length d, it takes $O(dk)$ to generate a sample. To decode, it takes $O(k^2)$ to compute $M_j(x_j)$, which must be done d times by backtracking. Hence, decoding takes a total of $O(dk^2)$ time.

6. What is the difference between inference and decoding in Markov chains? Decoding is a type of inference task. Decoding in Markov chains means we compute assignments to all x_j with the highest joint probability.

7. Suppose we are using a hidden Markov model to track the location of a submarine using sonar measurements. What would x_j and z_j represent in this example? z_j would represent the location of the submarine (hidden states), while x_j would represent the sonar measurements (observed evidence).

8. What is "explaining away"? "Explaining away" is the idea that observing a child makes the parents of that child conditionally dependent. Intuitively, each parent has a certain amount of "weight" on the child, and knowing one parent affects the conditional distribution of the other parent(s).

9. If two variables are not d-separated, are they necessarily dependent? If two variables are d-separated, are they necessarily independent? If two variables are d-separated, then they are independent. If they are not d-separated, they are NOT necessarily dependent.

10. What is an advantage and a disadvantage of using logistic regression to parameterize the CPDs in DAGs compared to a tabular representation? Using logistic regression to parameterize the CPDs results in less parameters than a tabular representation, but the DAG cannot model causal relationships.

11. When decoding a DAG, why does the order that we compute the messages matter? The order matters because the graphs are directed and messages may depend on multiple variables. Hence, starting at a leaf node would give a different decoding than starting at an internal node.

12. What is the relationship between multivariate Gaussians and UGMs? A multivariate Gaussian is a pairwise UGM.

13. What is the relevance of the Markov blanket in ICM? The Markov blanket reduces runtime from $O(dk)$ to $O(mk)$ if there are only m nodes in the Markov blanket.

14. Why might we do "thinning" of the samples when we use Gibbs sampling? When we have very long Markov chains (many states), thinning reduces the amount of computational memory/storage needed.