Homework 1

Zachary DeStefano, 15247592
CS 274B: Spring 2016

Problem 1:

I will assume that each random variable can take on d values.

Part A

To satisfy $p(W|X,Y,Z)$, we need $d-1$ parameters for values of W and each of those values are conditioned on d^3 possible configurations of X,Y,Z, thus we need $d^3(d-1)$ parameters.

To satisfy $p(Z|X,Y)$, we need $d^2(d-1)$ parameters because we have $d-1$ parameter values each conditioned on d^2 configurations.

To satisfy $p(Y|X)$, we need $d(d-1)$ parameters because we have $d-1$ parameter values each conditioned on d configurations.

To satisfy $p(X)$, we need $d-1$ parameters because we have $d-1$ parameter values.

Our total is thus

$$\frac{d^4-1}{d-1}(d-1)$$

Simplifying, our final total is

$$d^4-1$$

Thus this Bayesian network does not simplify the joint distribution

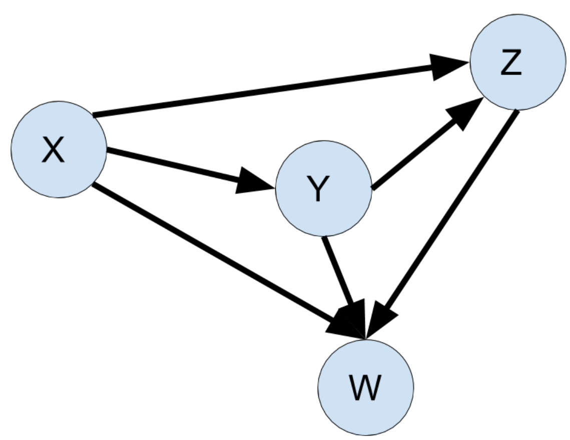

Figure 1: Minimal Directed Graphical Model for Part A

Part B

For each random variable, we need $d-1$ parameters.
They are all independent
Thus our total is just $4(d-1)$

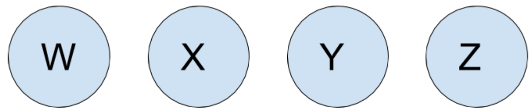

Figure 2: Minimal Directed Graphical Model for Part B

Part C

The variables $p(Z|Y)$, $p(W|Y)$, $p(X|Y)$ each need $d(d-1)$ parameters since we have $d-1$ parameter values conditioned on d configurations.

The factor $p(Y)$ needs $d-1$ parameters

Thus our total is $(3d+1)(d-1)$

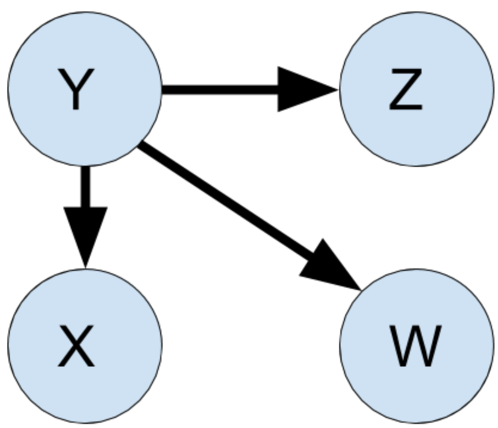

Figure 3: Minimal Directed Graphical Model for Part C

Part D

To satisfy $p(X)$ and $p(Y)$ we need $d-1$ parameters for each of them

To satisfy $p(W|X)$ we need $d(d-1)$ parameters since there are $d-1$ parameter values conditioned on d configurations.
To satisfy $p(Z|X,Y)$ we need $d^2(d-1)$ parameters since there are $d-1$ parameter values conditioned on d^2 configurations.

Our total is thus

$$(d^2+d+2)(d-1) = (d^2+d+1)(d-1)+(d-1) = (d^3-1)+(d-1) = d^3+d-2$$

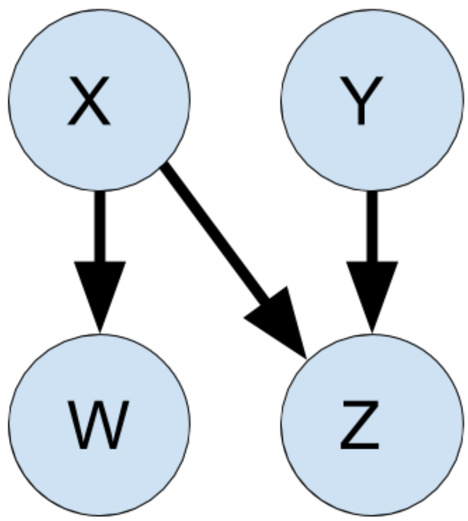

Figure 4: Minimal Directed Graphical Model for Part D

Part E

To satisfy $p(Z)$ we need $d-1$ parameters

To satisfy $p(Y|Z)$ we need $d(d-1)$ parameters as there are $d-1$ values conditioned on d configurations.
To satisfy $p(X|Y)$ we need $d(d-1)$ parameters for same reason as $p(Y|Z)$

To satisfy $p(W|X)$ we need $d(d-1)$ parameters for same reason as $p(Y|Z)$

Our total is thus $(3d+1)(d-1)$ parameters

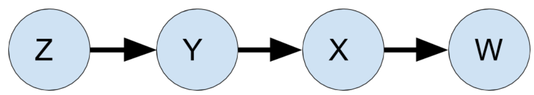

Figure 5: Minimal Directed Graphical Model for Part E

Part F

To satisfy $p(X)$ we need $d-1$ parameters
To satisfy the other three factors, we need $d(d-1)$ parameters for each of them for the same reason as $p(Y|Z)$ in part E
Our total is thus $(3d+1)(d-1)$ parameters

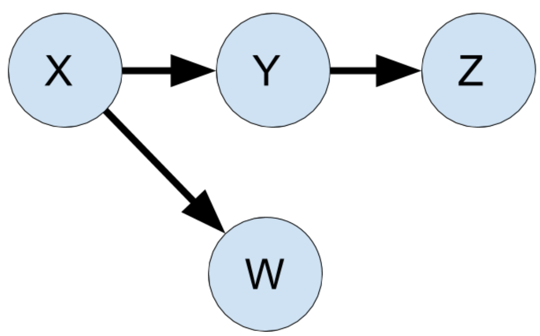

Figure 6: Minimal Directed Graphical Model for Part F

Problem 2:

Part A

No
There is a "vee" structure between them and the variables are unobserved
Thus all paths are inactive and they are conditionally independent

Part B

Yes
This allows you to infer new values for *power_in_building*
This will in turn mean new probabilities for *Sam_reading_book*

Part C

Yes
Observing a value for *screen_lit_up* lets you infer information about the probability of *power_in_wire*. Knowledge of *projector_plugged_in* combined with the information about *power_in_wire* will affect *power_in_building*. This will in turn affect *Sam_reading_book* because it is connected through a chain to *power_in_building*.

Part D

If *lamp_works* was observed, then we would update the probabilities for *projector_lamp_on*
We would then have to update the probabilities for *screen_lit_up*
This would cause us to update probabilities for *ray_says_screen_is_dark*

Part E

If we observe just *power_in_projector* then the same variables from Part D will have their probabilities changed.
We would also update the probability for *projector_switch_on*
We would also have to update *power_in_building* and *power_in_wire*
It would propagate and affect *Sam_reading_book* and *room_light_on*

Problem 3:

Part A

We need to solve the following

$$p(0,0;\theta) + p(0,1;\theta) + p(1,0;\theta) + p(1,1;\theta) = 1$$

This ends up being the following:

$$exp(-A(\theta)) + exp(-A(\theta)) + exp(\theta_x - A(\theta)) + exp(\theta_x + \theta_{xy} - A(\theta)) = 1$$

After doing some factoring

$$\frac{exp(\theta_x) + exp(\theta_{xy} + \theta_x) + 2}{exp(A(\theta))} = 1$$

After cross multiplying and solving for $A(\theta)$

$$A(\theta) = log(exp(\theta_x) + exp(\theta_{xy} + \theta_x) + 2)$$

Part B

After letting $\theta_{xy} = 1$ we have the following

$$A(\theta) = log(exp(\theta_x) + exp(1 + \theta_x) + 2)$$

After some factoring

$$A(\theta) = log(exp(\theta_x)(1 + exp(1)) + 2)$$

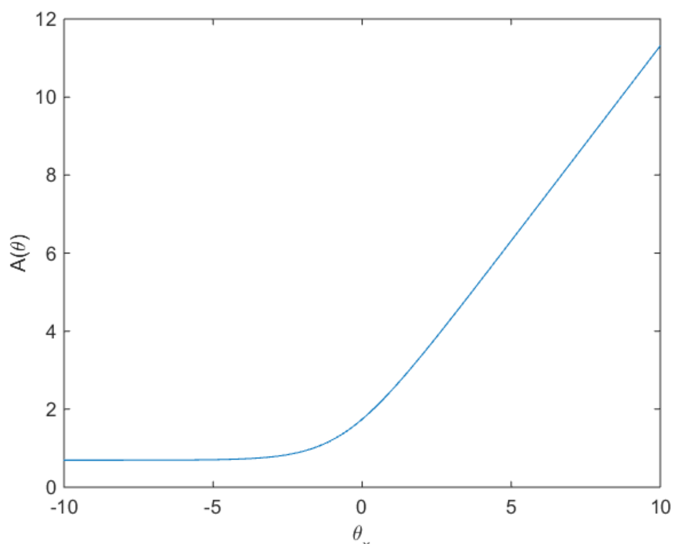

Figure 7: Plot for $A(\theta)$. It appears convex as expected

Part C

This is the partial with respect to θ_x

$$\frac{\partial A}{\partial \theta_x} = \frac{exp(\theta_x) + exp(\theta_x + \theta_{xy})}{exp(\theta_x) + exp(\theta_x + \theta_{xy}) + 2}$$

This is the partial with respect to θ_{xy}

$$\frac{\partial A}{\partial \theta_{xy}} = \frac{exp(\theta_x + \theta_{xy})}{exp(\theta_x) + exp(\theta_x + \theta_{xy}) + 2}$$

Thus we have

$$\bigtriangledown A(\theta) = [\frac{exp(1) + exp(3)}{exp(1) + exp(3) + 2}, \frac{exp(3)}{exp(1) + exp(3) + 2}]$$

Approximately

$$\bigtriangledown A(\theta) = [0.91937, 0.80978]$$

This is also the expected value of x and xy respectively if $\theta = [1, 2]$

Problem 4

Part A

Here is the code to compute Part A (it runs after the code from markov_chain.py)

```
p0vals = np.zeros(len(xvals))
for i in range(mSeq):
    curSeq = x[i]
    x0val = curSeq[0]
    p0vals[x0val] += 1
p0vals = np.divide(p0vals,mSeq)
print np.transpose(np.matrix(p0vals))
```

This is the result of the print statement

```
[[ 0.87656353]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.03587887]
 [ 0.        ]
 [ 0.0875576 ]
 [ 0.        ]]
```

Part B

Here is the code for Part B (follows the part A code)

```
Tmatrix = np.zeros((len(xvals),len(xvals)))
for i in range(mSeq):
    curSeq = x[i]
    for j in range(1,len(curSeq)):
        xPrev = curSeq[j-1]
        xCurrent = curSeq[j]
        Tmatrix[xPrev,xCurrent] += 1
Tsum = np.matrix(np.sum(Tmatrix,axis=1))
Tsum = np.transpose(Tsum)
TsumTiled = np.matlib.repmat(Tsum,1,8)
Tmatrix = np.divide(Tmatrix,TsumTiled)
print 'Transition Matrix (first 5 states) is as follows:'
print Tmatrix[0:5,0:5]

epsilon = 1e-8
Tmatrix2 = np.matrix(Tmatrix)
curX = np.matrix(p0vals)
for i in range(500):
    prevX = np.copy(curX)
    curX = curX*Tmatrix2
    diffX = np.abs(np.subtract(prevX,curX))
    if np.sum(diffX)<epsilon:
        break
print
print 'Stationary Distribution:'
print np.transpose(np.matrix(curX))
```

This is the output of those print statements

```
Transition Matrix (first 5 states) is as follows:
[[  6.02215434e-01   3.40558643e-02   1.20877341e-01   3.68453980e-02
    1.02646009e-01]
 [  4.60651517e-01   2.19411172e-02   4.17552894e-02   2.52994515e-02
    4.52255681e-02]
 [  8.53601579e-02   3.56129514e-03   8.14666698e-01   4.89824037e-03
    5.41783917e-03]
 [  1.61422755e-01   7.77332840e-03   2.13345896e-02   7.02897331e-01
    3.12952182e-02]
 [  5.62093936e-02   5.89229505e-04   7.55919431e-04   3.38806965e-03
    9.18310307e-01]]

Stationary Distribution:
[[  1.78394852e-01]
 [  1.16882378e-02]
 [  2.24024192e-01]
 [  3.89020939e-02]
 [  3.36189664e-01]
 [  9.78638026e-05]
 [  2.10385915e-01]
 [  3.17181525e-04]]
```

Part C

Here is the code for it (follows part B code)

```
Omatrix = np.zeros((len(xvals),len(ovals)))
for i in range(mSeq):
    curSeq = x[i]
    curObs = o[i]
    for j in range(len(curSeq)):
        xt = curSeq[j]
        ot = curObs[j]
        Omatrix[xt,ot] += 1
Osum = np.matrix(np.sum(Omatrix,axis=1))
Osum = np.transpose(Osum);
OsumTiled = np.matlib.repmat(Osum,1,20)
Omatrix = np.divide(Omatrix,OsumTiled)
print
print 'Emission Probability Matrix (first 5 states) is as follows:'
print Omatrix[0:5,0:5]
```

This is the output of the print statements

```
Emission Probability Matrix (first 5 states) is as follows:
[[ 0.05946997  0.01452542  0.06708915  0.04987037  0.03195305]
 [ 0.05294974  0.02418001  0.04701668  0.03918057  0.05093474]
 [ 0.06182642  0.01925435  0.03359294  0.0473769   0.05593569]
 [ 0.09200121  0.01224888  0.07588249  0.09230407  0.03970791]
 [ 0.11633794  0.01160627  0.04784776  0.08957451  0.0400521 ]]
```

Part D

For each sequence, we will append a value at the end signifying the end state. In the transition matrix, we will add a row and column that signify the "end" value.

Part E

Here is my Markov Maginals function

```
def markovMarginals(x,o,p0,Tr,Ob):
    dx,do = Ob.shape   # if a numpy matrix
    L = len(o)
    f = np.zeros((L,dx))
    r = np.zeros((L,dx))
    p = np.zeros((L,dx))

    p0 = np.reshape(p0, (dx, 1))
    compF = np.multiply(Ob[:, o[0]], p0)
    f[0, :] = np.reshape(compF, dx)  # compute initial forward message
    log_pO = np.log(f[0,:].sum())   # update probability of sequence so far
    f[0,:] /= f[0,:].sum()  # normalize (to match definition of f)

    for t in range(1,L):    # compute forward messages
        prevF = np.reshape(f[t - 1, :], (1, dx))
        curXprobs = np.transpose(prevF * Tr)
        curObcol = Ob[:, o[t]]
        f[t, :] = np.reshape(np.multiply(curXprobs, curObcol), dx)
        log_pO += np.log(f[t, :].sum())
        f[t, :] /= f[t, :].sum()  # normalize (to match definition of f)

    r[L-1,:] = 1.0  # initialize reverse messages
    p[L-1,:] = np.multiply(r[L-1,:],f[L-1,:])  # and marginals

    for t in range(L-2,-1,-1):
        prevR = np.reshape(r[t + 1, :], (dx, 1))
        curObcol = Ob[:, o[t + 1]]
        curCol = np.matrix(np.multiply(prevR, curObcol))
        r[t, :] = np.reshape(Tr * curCol, dx)
        r[t, :] /= r[t, :].sum()
        p[t, :] = np.multiply(r[t, :], f[t, :])
        p[t, :] /= p[t, :].sum()

    return log_pO, p
```

I printed out the files for the first 5 sequences as well as the requested probabilities. Here is the result

```
Files corresponding to first 5 sequences:
12as.txt
153l.txt
16pk.txt
16vp.txt
1914.txt

p6 for sequence 0:
[  1.02101704e-01   5.60632171e-03   1.54825441e-01   2.52220214e-02
   6.45436063e-01   3.70925988e-06   6.67175945e-02   8.71440954e-05]

p9 for sequence 2:
[  1.92279325e-01   2.16531983e-02   2.04049763e-01   5.81202218e-02
   2.37895589e-01   2.72872172e-04   2.85012935e-01   7.16096328e-04]

logp for sequence 4:
-493.736667034
```

Part E, Toy Example

As a toy example, I made an emission matrix that was close to the identity matrix. For the transition matrix, there is a high probability of transitioning from state 0 to state 1, state 1 to state 2, and state 2 to state 0, with the rest of the probabilities relatively low. I made the observation sequence follow this transition pattern.

I wrote the following code:

```
toyT = np.matrix([[0.1,0.8,0.1],[0.1,0.1,0.8],[0.8,0.1,0.1]])
toyOmat = np.matrix([[0.8,0.1,0.1],[0.1,0.8,0.1],[0.1,0.1,0.8]])
toyP0 = np.matrix([0.33,0.33,0.34])
toyObs = np.array([1, 2, 0, 1, 2, 0, 1])

[toyLog,toyPmatrix] = markovMarginals(x,toyObs,toyP0,toyT,toyOmat)

print
print 'Toy P Matrix:'
print toyPmatrix
```

As expected when I printed, the probabilities of the states that correspond to the observations was quite high:

```
Toy P Matrix:
[[ 0.01744661  0.9645781   0.01797529]
 [ 0.00468024  0.00462266  0.9906971 ]
 [ 0.99422749  0.00288939  0.00288312]
 [ 0.00267809  0.99464313  0.00267878]
 [ 0.0028817   0.00288162  0.99423668]
 [ 0.99076621  0.0046169   0.00461689]
 [ 0.01745585  0.9650883   0.01745585]]
```