Probabilistic Modelling and Reasoning
Assignment 2

Chris Swetenham (s1149322)

Dec 6, 2011

1 Principal Component Analysis

Question 1

We create a function to compute the components of the provided dataset and order them by the magnitude of the variance along each component. The components are the eigenvectors of the covariance matrix of the dataset.

```
function [Mu, E, Lambda, P] = getEigenvectors(Sequence)
% getEigenvectors computes the eigenvectors and eigenvalues of a dataset.
% INPUT Sequence: [NFrames x NFeatures]
%           Matrix with instances as rows and features as columns.
% OUTPUT Mu: [NFeatures x 1]
%           Column vector of means for each feature.
%        Lambda: [NFeatures x 1]
%           Column vector eigenvalues of covariance matrix, in descending
%           order.
%        E: [NFeatures x NFeatures]
%           Matrix whose columns are the eigenvectors of the covariance
%           matrix in corresponding order with the eigenvalues in lambda.
%        P: [NFeatures x 1]
%           Column vector of the cumulative percentage of variance
%           explained by each of the eigenvalues in lambda.

% Compute the mean vector for the sequence.
Mu = mean(Sequence)';
% Compute the eigendecomposition of the covariance matrix.
% V: [NFeatures x NFeatures] has the eigenvectors as columns.
% D: [NFeatures x NFeatures] is a diagonal matrix of the eigenvalues.
[V, D] = eig(cov(Sequence));
% Extract and sort the eigenvalues in descending order.
% I: [NFeatures x 1] stores the corresponding indices of the sorted entries
% in the original matrix.
[Lambda, I] = sort(diag(D), 1, 'descend');
% Use the index vector to get the sorted eigenvectors.
E = V(:, I);
% Get the cumulative sum of the variance % explained by each eigenvector.
P = 100 * cumsum(Lambda) / sum(Lambda);
```

Listing 1: getEigenvectors.m

Question 2

We use the function defined in the last question to process the provided dataset. We use `find` to find the index of the last component required to explain 95% or more of the variation in the dataset.

```
[Mu, E, Lambda, P] = getEigenvectors(sequence_X);
[Log, Cleanup] = makeLogFile('p1q2.log');
fprintf(Log, 'Eigenvalue %d: %4.1f\n', 1, Lambda(1));
fprintf(Log, 'Eigenvalue %d: %4.1f\n', 2, Lambda(2));
% Get the index of the first element in P which is >= 95.
NComponents = find(P >= 95, 1);
% Get its value.
Y = P(NComponents);
figure;
plot(P);
line([4 4 0 4], [50 Y Y Y]);
xlabel('number of components');
ylabel('cumulative % variance');
set(gca,'XTick',sort([0:10:NFeatures NComponents]), 'YTick',sort([0:10:100 Y]));
writeFigurePDF('p1q2.pdf');
```

Listing 2: script.m

The first two eigenvalues for the provided dataset are 1401.9 and 631.2. As shown in Figure 1, the first 4 components are enough to explain 95.8% of the variance in the dataset.

```
Eigenvalue 1: 1401.9
Eigenvalue 2: 631.2
```

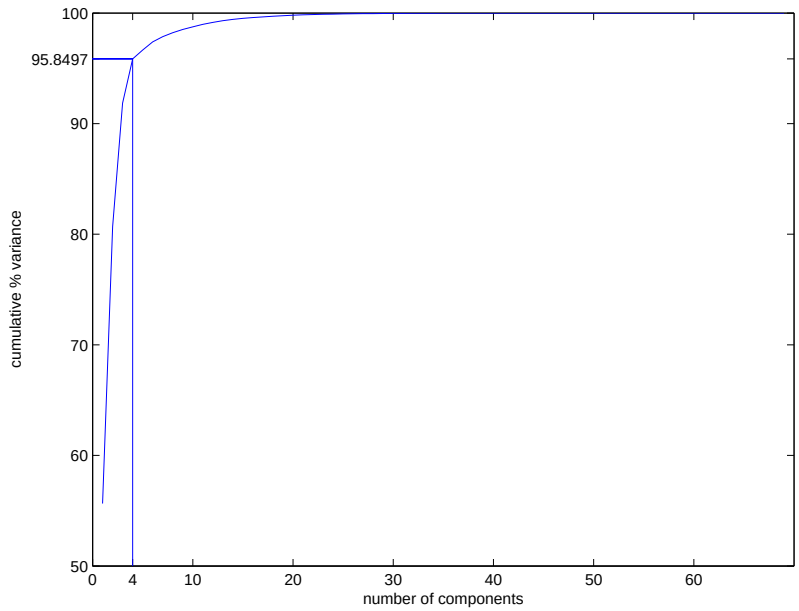


Figure 1: Cumulative % variance with increasing components

Question 3

We wish to investigate the kind of variability captured by each component. The eigenvalues for each component give us the variance for each component, so by taking the square root we can view the variability in the dataset along the corresponding component in the range of one standard deviation.

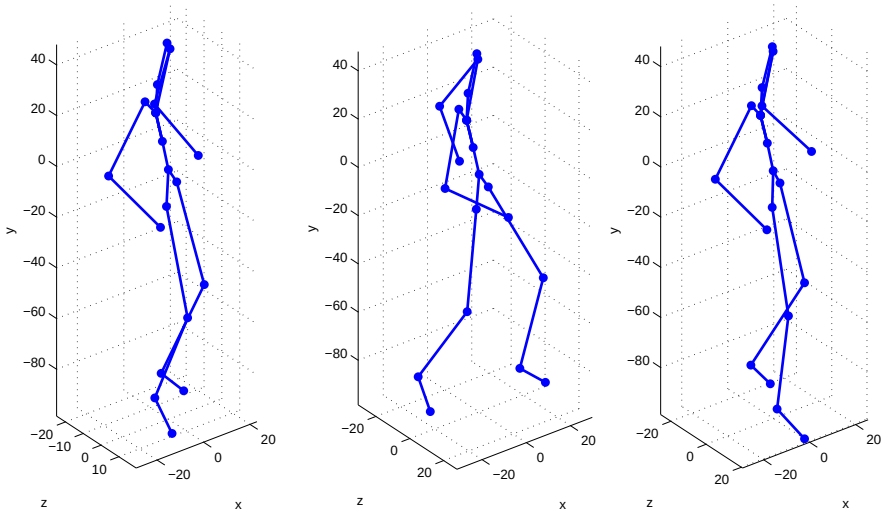
```
function [NewSeq] = makeSequence(NFrames, Mu, Lambda, Eig)
% makeSequence makes a sequence showing 1 stdev along Eig.
% INPUT NFrames: [1]
%           The number of frames to generate.
%       Mu: [NFeatures x 1]
%           The average pose.
%       Lambda: [1]
%           The variance along Eig.
%       Eig: [NFeatures x 1]
%           The eigenvector along which to animate.
% OUTPUT NewSeq: [NFrames x NFeatures]
%           The generated sequence.

% Replicate the mean into a matrix. MMu: [NFrames x NFeatures]
MMu = repmat(Mu', [NFrames 1]);
% Lambda is the variance along the eigenvector; take the standard dev.
StDev = sqrt(Lambda);
% Linear interpolation from -StDev to +StDev. Z: [NFrames x 1]
Z =  2 * StDev * (0 : NFrames-1)'/(NFrames - 1) - StDev;
% Pose reconstruction.
NewSeq = MMu + Z * Eig';
```

Listing 3: makeSequence.m

```
% Visualise the mean pose.
sequence_Y0 = makeSequence(NFrames, Mu, 0, E(:, 1));
figure;
skelPlayData(skeleton, sequence_Y0, frame_length);
writeFigurePDF('p1q3-mean.pdf');
% Visualise the first component.
sequence_Y1 = makeSequence(NFrames, Mu, Lambda(1), E(:, 1));
figure;
skelPlayData(skeleton, sequence_Y1, frame_length);
writeFigurePDF('p1q3-comp1.pdf');
% Visualise the second component.
sequence_Y2 = makeSequence(NFrames, Mu, Lambda(2), E(:, 2));
figure;
skelPlayData(skeleton, sequence_Y2, frame_length);
writeFigurePDF('p1q3-comp2.pdf');
```

Listing 4: script.m

The mean pose is shown in Figure 2a. The first component, shown at one standard deviation in Figure 2b, captures mostly the swing and yaw of the shoulder and hip joints. The second component, shown at one standard deviation in Figure 2c, captures mostly the bend of the knees and ankles.



(a) Mean Pose (b) Comp 1 at one st. dev. (c) Comp 2 at one st. dev.

Figure 2: Visualising the principal components

Question 4

Since the eigenvectors form a basis for the original state space, we can project frames onto our principal components. Taking the first two principal components, we project the provided sequence and visualise it as a trajectory in 2D state space. The result is shown in Figure 3.

```
function [ZSeq] = projectSequence(Mu, E, Seq, ZDims)
% projectSequence projects a data sequence onto the first ZDims components.
% INPUT Seq: [NFrames x NFeatures]
%           Input data sequence.
%       Mu: [NFeatures x 1]
%           Average of the features in the data sequence.
%       E: [NFeatures x NFeatures]
%           Column eigenvector matrix.
% OUTPUT ZSeq: [NFrames x ZDims]
%           The projected sequence.

[NFrames, ~] = size(Seq);
% Transformation to the reduced space. W: [NFeatures x ZDims]
W = E(:, 1:ZDims);
% Replicate the mean into a matrix. MMu: [NFrames x NFeatures]
MMu = repmat(Mu', [NFrames 1]);
% Perform the transformation.
ZSeq = (Seq - MMu) * W;
```

Listing 5: projectSequence.m

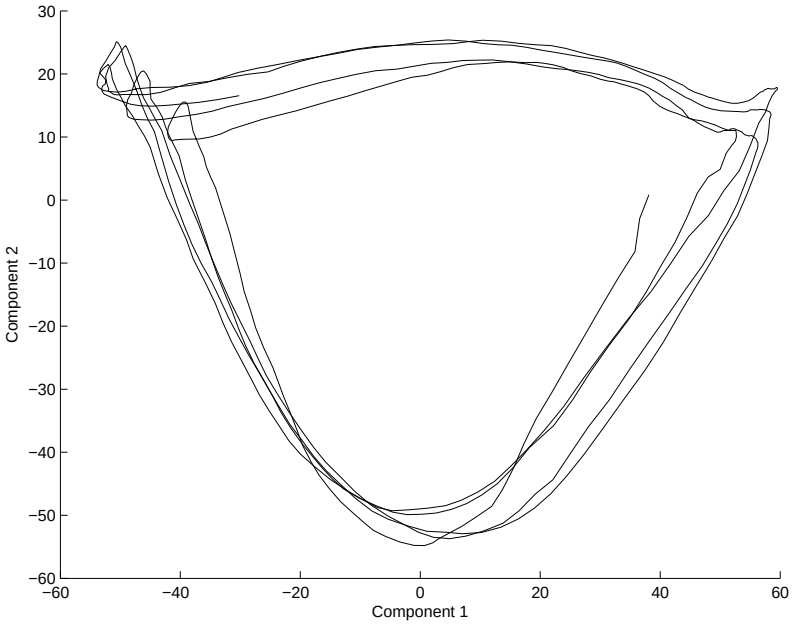


Figure 3: Sequence projected in 2-component state space

```
% Z: [NFrames x 2]
Z = projectSequence(Mu, E, sequence_X, 2);
figure;
line('XData', Z(:, 1), 'YData', Z(:, 2));
xlabel('Component 1');
ylabel('Component 2');
writeFigurePDF('p1q4.pdf');
```

Listing 6: script.m

The trajectory is repeating, and we can identify four complete walk cycles. The corners in the motion correspond to the points in the walk cycle where the feet touch the ground and the direction of the main component reverses. As we can see, the walk cycle is asymmetric; there is much less motion along the second component on one step as opposed to the other. Looking back to the original animation, we notice the gait is indeed asymmetric; for example, the left ankle does not bend at all whereas the right ankle shows clear compliance and toe-off during the walk.

2 Generative Topographic Mapping

Question 1

We use the `gtm1dinittrain` function to train a GTM model with a 1D latent space using the EM algorithm. In Figure 4 we project the frames of the sequence and the 50 gaussians of the GTM mixture model into the space defined by the first two principal components of the data, as computed in Part 1.

The 1D latent space coils back on itself when projected into this 2D space. There is generally good coverage of the data, so it might serve well as a model for the probability of a given pose being a walking pose. As a model of walking motion however, it is lacking any notion of time: continuously varying the latent variable will give us a reconstructed path in data space which does not correspond to a forward or backward walking motion. Ideally, the GTM model's latent space would loop around the trajectory once, coming back where it started. This might be achievable if we initialised the model such that the projection in data space was a cycle, and then ran the EM algorithm to optimise the fit.

```
figure;
Net = gtm1dinittrain(sequence_X, E, 50, 7, 200);
writeFigurePDF('p2q1.pdf');
```

Listing 7: script.m

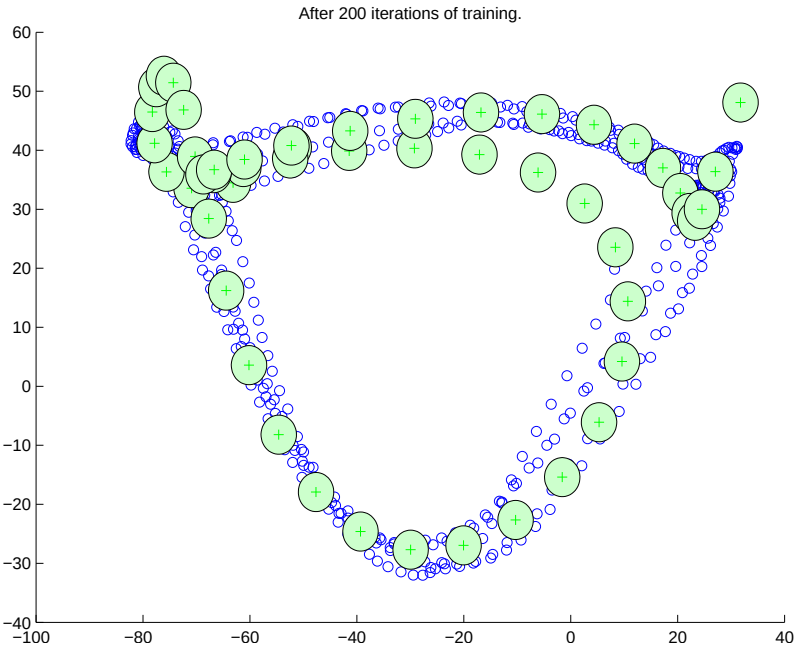


Figure 4: 1D GTM model after 200 iterations of EM

Question 2

We use the `gtmprob` function to evaluate the probability of each frame in the sequence in the model. Under the assumption that the frames are IID the probability of the sequence of frames is the product of these probabilities. To evaluate the log likelihood, we use `sum(log(P))` rather than `log(prod(P))` because the latter would cause numerical underflow under IEEE floating-point arithmetic.

```
% P: [NFrames x 1]
P = gtmprob(Net, sequence_X);
LL = sum(log(P));
[Log, Cleanup] = makeLogFile('p2q2.log');
fprintf(Log, 'Log likelihood of GTM model: %1.3e\n', LL);
```

Listing 8: script.m

We calculate a log likelihood of -7.939×10^4, which we can use to compare our model with other models and choices of parameter values.

```
Log likelihood of GTM model: -7.939e+004
```

Question 3

We train the same GTM model, varying the parameters of the model an measuring the effect on the log likelihoods. In the first experiment we vary the number of RBF centers. The results are shown in Figure 5. As we increase the number of RBF centers, the quality of the fit as measured by the log likelihood of our training sequence increases smoothly, though with diminishing returns.

In the second experiment we vary the number of latent sample points. The results are shown in Figure 6. Although the overall shape is similar, there is a notable discontinuity in the range of 60-100 sample points.

```
function [LL] = gtmTrainAndReport(Sequence, E, Pts, Ctrs)
% gtmTrainAndReport trains a gtm model, displaying a figure.
% INPUT Sequence: [NFrames x NFeatures]
%           Matrix with instances as rows and features as columns.
%       E: [NFeatures x NFeatures]
%           Matrix whose columns are the eigenvectors of the covariance
%           matrix in corresponding order with the eigenvalues in lambda.
%       Pts: [1 x 1]
%           The number of latent points to use.
%       Ctrs: [1 x 1]
%           The number of rbf centers to use.
% OUTPUT LL: [1 x 1]
%           The log likelihood of the Sequence under the trained model.

figure;
Net = gtm1dinittrain(Sequence, E, Pts, Ctrs, 200);
% P: [NFrames x 1]
P = gtmprob(Net, Sequence);
LL = sum(log(P));
```

Listing 9: gtmTrainAndReport.m

```
% Vary the number of RBF Centers
LLs_byNCtrs = [];
CtrValues = [2 3 5 7 11 23];
for NCtrs = CtrValues
    LLs_byNCtrs(end + 1) = gtmTrainAndReport(sequence_X, E, 50, NCtrs);
    writeFigurePDF(sprintf('p2q3-%ds-%dc.pdf', 50, NCtrs));
end
figure;
plot(CtrValues, LLs_byNCtrs);
writeFigurePDF('p2q3-plotByCtrs.pdf');
% Vary the number of sample points
LLs_ByNPts = [];
PtValues = [10 30 50 100 150];
for NPts = PtValues
    LLs_ByNPts(end + 1) = gtmTrainAndReport(sequence_X, E, NPts, 7);
    writeFigurePDF(sprintf('p2q3-%ds-%dc.pdf', NPts, 7));
end
figure;
plot(PtValues, LLs_ByNPts);
writeFigurePDF('p2q3-plotByPts.pdf');
```

Listing 10: script.m

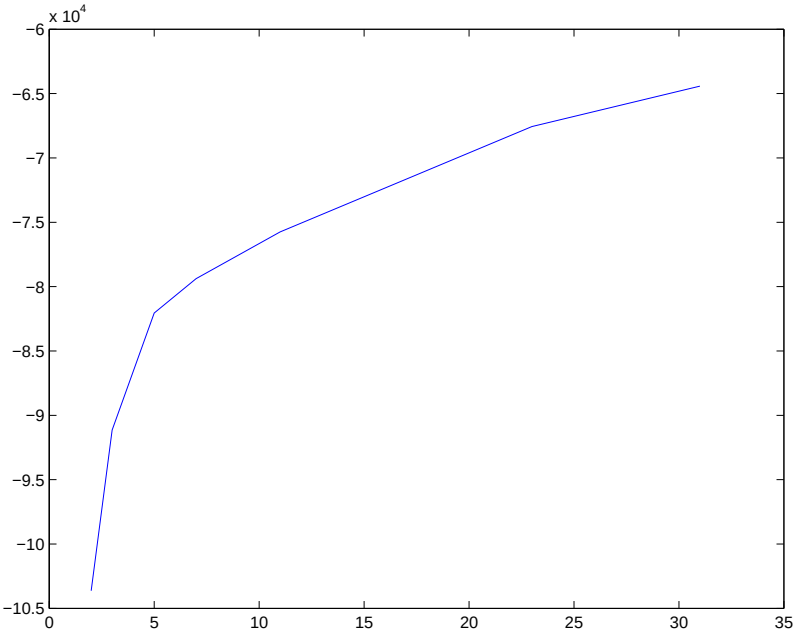


Figure 5: Log likelihood against number of RBF centers

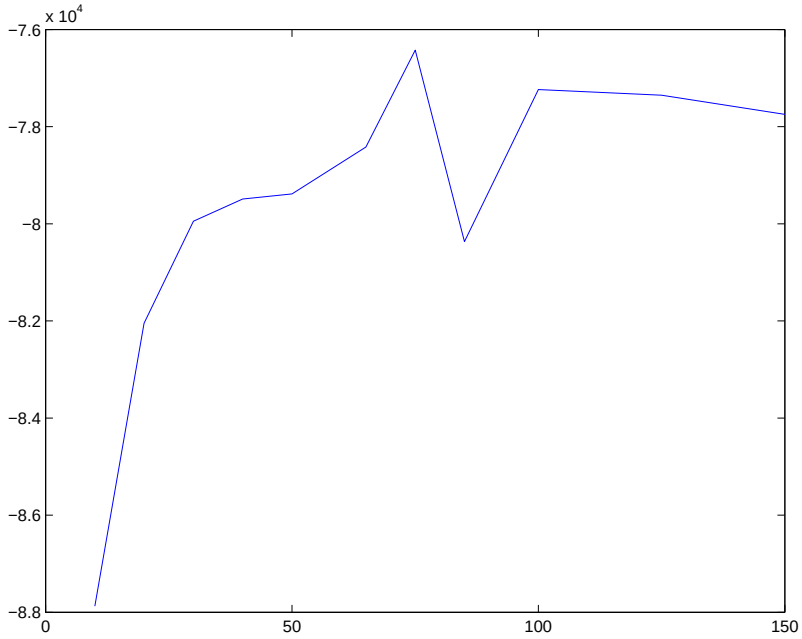


Figure 6: Log likelihood against number of sample points

Question 4

This time we train a GTM model with a 2D latent space. We compute the mean responsibility for each frame in the latent space. This projection has the same overall shape as the PCA projection, but it is less smooth; it visibly jumps between points on discrete grid coordinates.

```
Net2D = gtm2dinittrain(sequence_X, 50, 10, 50);
% Compute the mean latent projection of the sequence
Means = gtmlmean(Net2D, sequence_X);
figure;
line('XData', Means(:, 1), 'YData', Means(:, 2));
writeFigurePDF('p2q4.pdf');
```

Listing 11: script.m

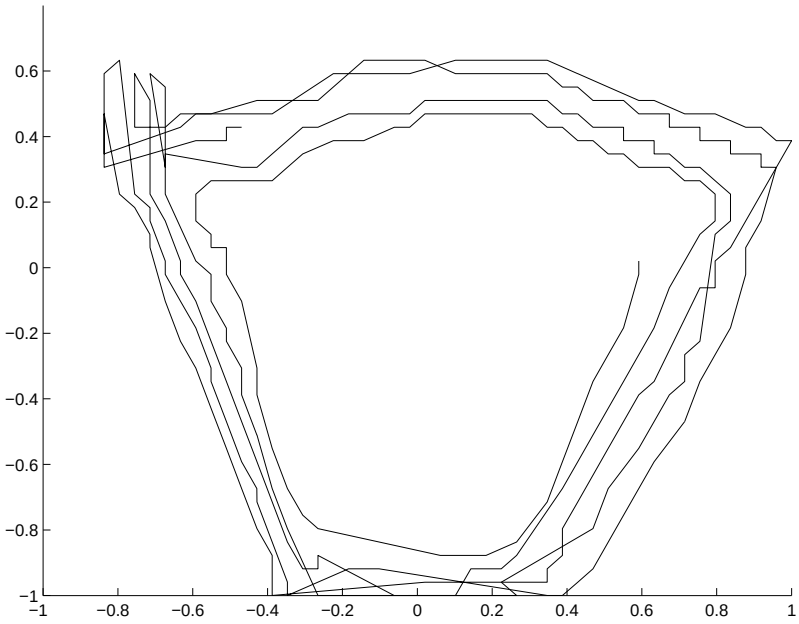


Figure 7: 2D latent space projection for 2D GTM model

3 Factor Analysis

Question 1

We wish to compute the mean of the posterior projection of the original sequence into the FA model's latent space. In other words, we wish to compute $E(\mathbf{z} \mid \mathbf{x})$. Regrettably I tried to derive this from first principles and erroneously concluded using the pseudo-inverse of the matrix W would be sufficient, and this is what the code below uses. I do not have time left to fix this, but the resulting plot should be correct as far as the overall structure is concerned.

```
[W,Mu,DiagPsi,~] = fa(sequence_X, 2, 50);
MMu = repmat(Mu', [NFrames 1]);
sequence_Z_FA = (sequence_X - MMu) * pinv(W)';
figure;
line('XData', sequence_Z_FA(:, 1), 'YData', sequence_Z_FA(:, 2));
writeFigurePDF('p3q1.pdf');
```

Listing 12: script.m

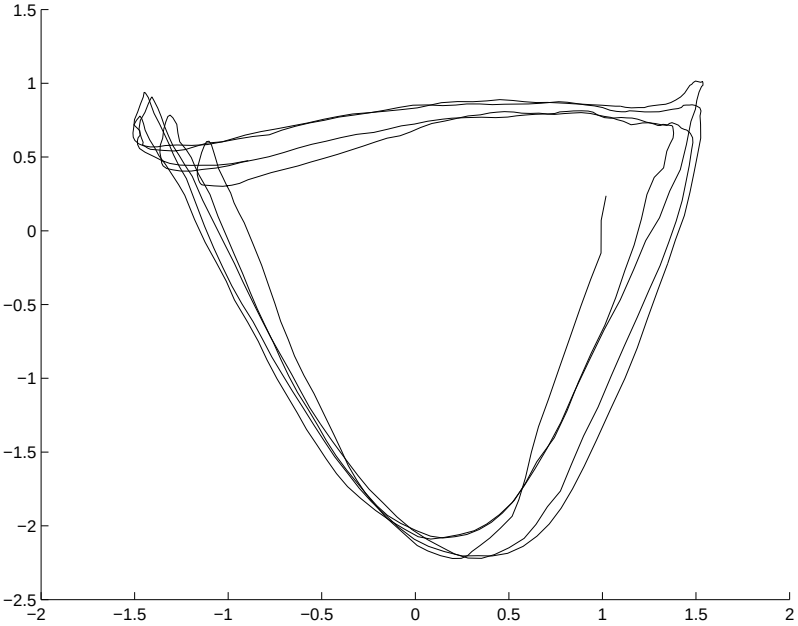


Figure 8: 2D latent space projection for FA model

Question 2

[Deleted]

Question 3

After the PCA transformation, the data space will have been rotated to remove covariance in the data. Since FA attempts to discover the covariance structure of the data, it would not make sense to run PCA then FA on a dataset.

Question 4

We implement the provided equation in MATLAB to compute the log likelihood of a datapoint under the learned model:

$$\log p(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T C^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}\log|C| - \frac{D}{2}\log(2\pi)$$

```
function LL = fallikelihood(Sequence, W, Psi, Mu)
% INPUT Sequence: [NFrames x NFeatures]
%           Matrix with instances as rows and features as columns.

[NFrames NFeatures] = size(Sequence);
C = W * W' + diag(Psi);
LL = 0;
```

```
for i = 1:NFrames
    FX = Sequence(i, :) - Mu';
    FL = -0.5 * FX/C * FX';
    LL = LL + FL;
end
LL = LL - 0.5 * NFrames * (logdet(C) + NFeatures * log(2 * pi));
```

Listing 13: fallikelihood.m

```
LL = fallikelihood(sequence_X, F, DiagPsi, Mu);
[Log, Cleanup] = makeLogFile('p3q4.log');
fprintf(Log, 'Log likelihood of FA model: %1.3e\n', LL);
```

Listing 14: script.m

The log likelihood of our data under the learned model is -1.521×10^4, which is much better than the log likelihood under the GTM model.

```
Log likelihood of FA model: -1.521e+004
```

Question 5

In order to choose the number of latent factors M, we vary this parameter and use cross-validation to evaluate the performance of the model using log likelihood. We plot the log likelihood of the test set and the training set in Figure 9. In order to make them comparable, they have been scaled. To show why this is necessary, consider taking a data set A and constructing a new dataset A2 by duplicating every instance in A. The log likelihood of A2 will necessarily be twice that of A, but the model does not fit A2 twice as well as A in any useful sense.

We see that the fit to the training data is always slightly better than to the test data, as would be expected, but the two continue to increase as we increase M, and so we are not overfitting the data. There are clear diminishing returns in the quality of the fit as M increases.

In general, it makes sense to shuffle the data before splitting it into folds, in order to ensure that each fold is representative of the distribution of the data as a whole. In particular, for our dataset if we don't do this then the data within a fold will be strongly correlated since the frames are in time order, and some particular part of the walk cycle will be over-represented in the fold.

Cross-validation gives us a way of selecting models that generalise well. Picking a model which is too complex, even if it fits our data, has several disadvantages: it may be computationally much more expensive than a simpler model, and it is more likely to generalise poorly - to overfit to the training data. Cross-validation specifically addresses the overfitting problem. Alternative methods specifically penalise more complex models, or include a prior over the possible parameter values of the model. In the latter case we are performing Bayesian model selection and computing the probability of the model given the data, rather than the likelihood which is the probability of the data given the model.

```
% Compute a random permutation of the frames
% NFrames = NFrames;
% Idx : 1 x NFrames
Idx = randperm(NFrames);
% RandSeq: [NFrames x NFeatures]
RandSeq = sequence_X(Idx, :);

NFolds = 7;
FoldSize = NFrames/NFolds;
Ms = [1 2 5 10 15 20 25]';
LLTrain = zeros(size(Ms));
LLTest = zeros(size(Ms));
for MIdx = 1:size(Ms)
    M = Ms(MIdx);
    LLTrainTot = 0;
    LLTestTot = 0;
    for V = 1:NFolds
        TestSet = RandSeq(1 : FoldSize, :);
        TrainSet = RandSeq(FoldSize + 1 : NFrames, :);
        RandSeq = circshift(RandSeq, 83);

        [W,Mu,DiagPsi,~] = fa(TrainSet, M, 50);

        LLTestTot = LLTestTot + fallikelihood(TestSet, W, DiagPsi, Mu);
        LLTrainTot = LLTrainTot + fallikelihood(TrainSet, W, DiagPsi, Mu);
    end
    % Compute the average across the NFolds runs
    LLTest(MIdx) = LLTestTot/NFolds;
    LLTrain(MIdx) = LLTrainTot/NFolds;
end
% We need to scale the log likelihoods by the number of instances in order
% to make them comparable.
LLTest = LLTest * NFolds;
LLTrain = LLTrain * NFolds / (NFolds-1);
figure;
plot(Ms, LLTest, 'g');
hold on;
plot(Ms, LLTrain, 'b');
writeFigurePDF('p3q5.pdf');
```

Listing 15: script.m

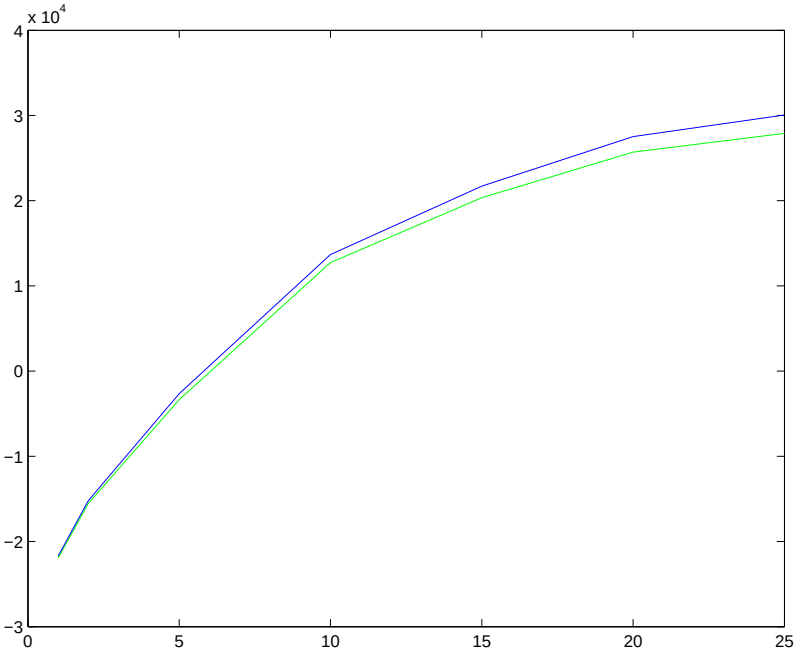


Figure 9: Log likelihood of training set (blue) and test set (green)

Question 6

We start with the expression for the log likelihood of a datapoint:

$$\log p(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T C^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}\log|C| - \frac{D}{2}\log(2\pi)$$

We are given two expansions for terms in this expression:

$$\mathbf{d}^T C^{-1}\mathbf{d} = \mathbf{d}^T\Psi^{-1}\mathbf{d} - \mathbf{d}^T\Psi^{-1}WA^{-1}W^T\Psi^{-1}\mathbf{d}$$

$$\log|C| = \sum_{i=1}^{D}\log\Psi_{ii} + \log|A|$$

where $\mathbf{d} = \mathbf{x}-\boldsymbol{\mu}$, $A = I_M + W^T\Psi^{-1}W$, and I_M is the $M \times M$ identity matrix.

We will consider computing the log likelihood for a single instance of $\mathbf{x}$. For each expression, we give the total path cost of computation in the order we have performed it; other orders can yield more expensive computations.

Ψ^{-1}	:	$O(D)$	since $\Psi$is diagonal.
$\mathbf{d}^T\Psi^{-1}\mathbf{d}$	:	$O(D)$	
$W^T\Psi^{-1}W$	:	$O(DM^2)$	
$I_M + W^T\Psi^{-1}W = A$	:	$O(DM^2)$	
A^{-1}	:	$O(DM^2 + M^3)$	
$\mathbf{d}^T\Psi^{-1}W$	:	$O(DM)$	
$\mathbf{d}^T\Psi^{-1}WA^{-1}$	:	$O(DM^2 + M^3)$	
$\mathbf{d}^T\Psi^{-1}WA^{-1}W^T\Psi^{-1}\mathbf{d}$	:	$O(DM^2 + M^3)$	
$\mathbf{d}^TC^{-1}\mathbf{d}$	:	$O(DM^2 + M^3)$	
$\log\lvert A\rvert$	:	$O(DM^2 + M^3)$	
$\sum_{i=1}^{D}\log\Psi_{ii}$	:	$O(D)$	
$\log\lvert C\rvert$	:	$O(DM^2 + M^3)$	
$\log p(\mathbf{x})$	:	$O(DM^2 + M^3)$	

The costs of computing $\mathbf{d}^TC^{-1}\mathbf{d}$ and $\log|C|$ are both $O(DM^2 + M^3)$, and therefore the cost of computing the log likelihood is the same. As long as $M < D$, this reduces to $O(DM^2)$ and is an improvement on the original cost.

4 Linear Dynamical System

Question 1

LDS is similar to FA, except that it models the change of the latent variables over time. Since we have a sequence of data over time where one frame is clearly related to and correlated with the next, LDS seems like a promising model to investigate.

Question 2

We use the provided functions to train an LDS model on the dataset. We compute the log likelihood of the data under the model, and compute the subspace angle between the C matrix of the LDS model and the W matrix of the FA model from the last part. This is an interesting comparaison because the matrices both map from a 2D latent space to the data space.

```
rand('seed', 0);
randn('seed', 0);
Net = lds(sequence_X, 2);
LL = lds_cl(Net, sequence_X, 2);
[Log, Cleanup] = makeLogFile('p4q2.log');
fprintf(Log, 'Log_likelihood_of_LDS_model:_%1.3e\n', LL);
% Compute the angle between the subspaces defined by C and W
Theta = subspace(Net.C, W);
fprintf(Log, 'Angle_between_subspaces_of_C_and_W:_%3.3e_(radians)\n', Theta);
```

Listing 16: script.m

The resulting log likelihood is -1.21×10^4, the best achieved so far. The angle between the two matrices is very small, suggesting that the improvement in the LDS model over the FA model comes from modelling the latent variables

over time, rather than by improving the learned mapping between latent and data space.

```
Log likelihood of LDS model: -1.210e+004
Angle between subspaces of C and W: 7.392e-003 (radians)
```

Question 3

Naive approaches to cross-validation would not perform well, since a shuffling of the input would lose the temporal aspect which LDS tries to model; and as mentioned in Part 3 Question 5, without shuffling the data within one fold is strongly self-correlated and not representative of the dataset as a whole. One possibility would be to use one walk cycle for each fold; in the provided dataset, we have only 4 complete cycles, but with more complete cycles cross-validation might be workable.

Question 4

We use the provided function `ldspost` to infer the posterior distribution of the latent states. The result is shown in Figure 10. We mirror and rotate the latent space to make the similarity to previous plots more apparent.

```
[sequence_Z_LDS, V] = ldspost(sequence_X, Net);
figure;
line('XData', -sequence_Z_LDS(:, 2), 'YData', -sequence_Z_LDS(:, 1));
writeFigurePDF('p4q4.pdf');
```

Listing 17: script.m

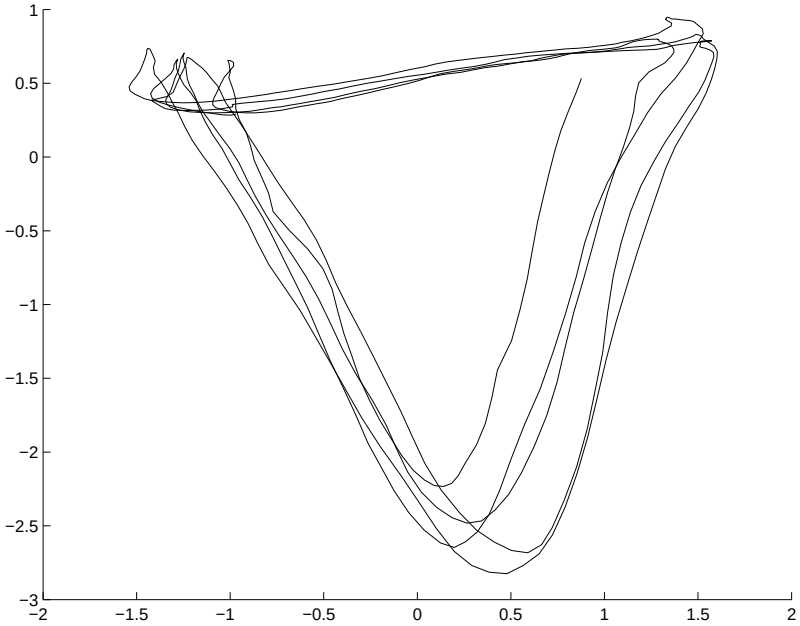


Figure 10: 2D latent space projection for LDS model

Question 5

Based on the sequence of inferred latent states, we can reconstruct the original movie by projecting back into data space.

```
MMu = repmat(Net.Mu, [NFrames 1]);
sequence_Y_reconstructed = sequence_Z_LDS * Net.C' + MMu;
figure;
skelPlayData(skeleton, sequence_Y_reconstructed, frame_length);
```

Listing 18: script.m

The resulting walk cycle is reasonably close to the original, although it suffers from some oscillations at the start and end of each footstep when the direction of movement reverses.

Question 6

Unlike previously examined models, the LDS model allows us to generate a new sequence. We initialise the latent variables to their mean value and run the model forward, both with and without noise. The sequence without noise, shown in Figure 11, is a clear spiral. This is not a close approximation of the more triangular shape of the latent space reconstruction above. The model will eventually settle down to a stable point.

The sequence with noise, shown in Figure 12, produces very jerky motion. The magnitude of the noise corresponds to the error in modelling the original trajectory of our data in latent space.

```
% Without noise
[sequence_Y_sampled, sequence_Z_sampled] = ldsSample(NFrames, Net, 0);
figure;
line('XData', -sequence_Z_sampled(:, 2), 'YData', -sequence_Z_sampled(:, 1));
writeFigurePDF('p4q6-nonoise.pdf');
figure;
skelPlayData(skeleton, sequence_Y_sampled, frame_length);
% With noise
[sequence_Y_sampled, sequence_Z_sampled] = ldsSample(NFrames, Net, 1);
figure;
line('XData', -sequence_Z_sampled(:, 2), 'YData', -sequence_Z_sampled(:, 1));
writeFigurePDF('p4q6-withnoise.pdf');
figure;
skelPlayData(skeleton, sequence_Y_sampled, frame_length);
```

Listing 19: script.m

```
function [SeqY, SeqZ] = ldsSample(NFrames, Net, Noise)
% ldsSample samples a sequence from the provided LDS model.
% INPUT NFrames: [1 x 1]
%           The number of frames to sample.
%       Net: [1 x 1] structure
%           The structure containing the trained model.
%       Noise: [1 x 1]
%           A scaling to apply to the noise in the model.
% OUTPUT SeqY: [NFrames x NFeatures]
%           The sampled sequence in data space.
%        SeqZ: [NFrames x NLatent]
%           The sampled sequence in latent space.

[NFeatures NLatent] = size(Net.C);
SeqY = zeros(NFrames, NFeatures);
SeqZ = zeros(NFrames, NLatent);
SeqZ(1, :) = Net.x0;
for I = 2 : NFrames
    SeqZ(I, :) = Net.A * SeqZ(I - 1, :)' + Noise * gsamp(zeros([1 NLatent]), Net.Q
        , 1)';
end
for I = 1 : NFrames
    SeqY(I, :) = Net.C * SeqZ(I, :)' + Net.Mu';
end
```

Listing 20: ldsSample.m

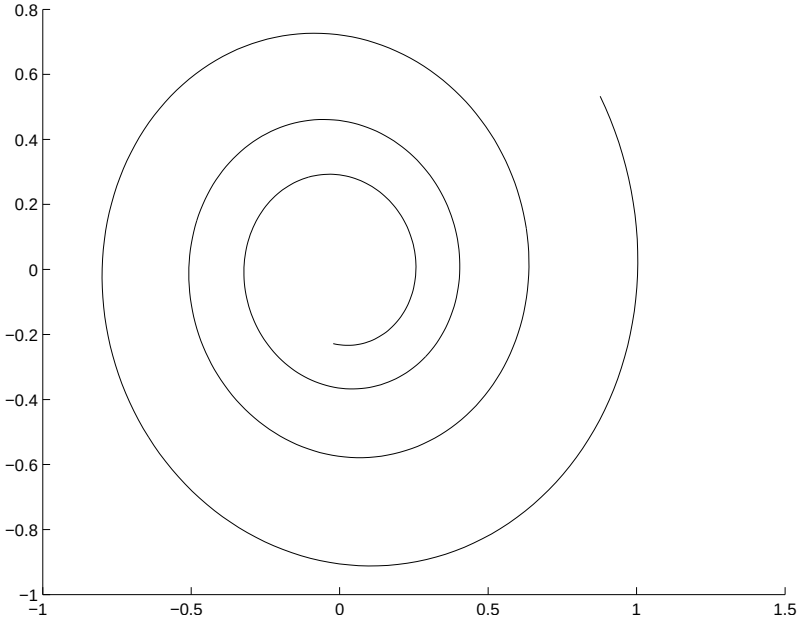


Figure 11: Sampled sequence in latent space, no noise

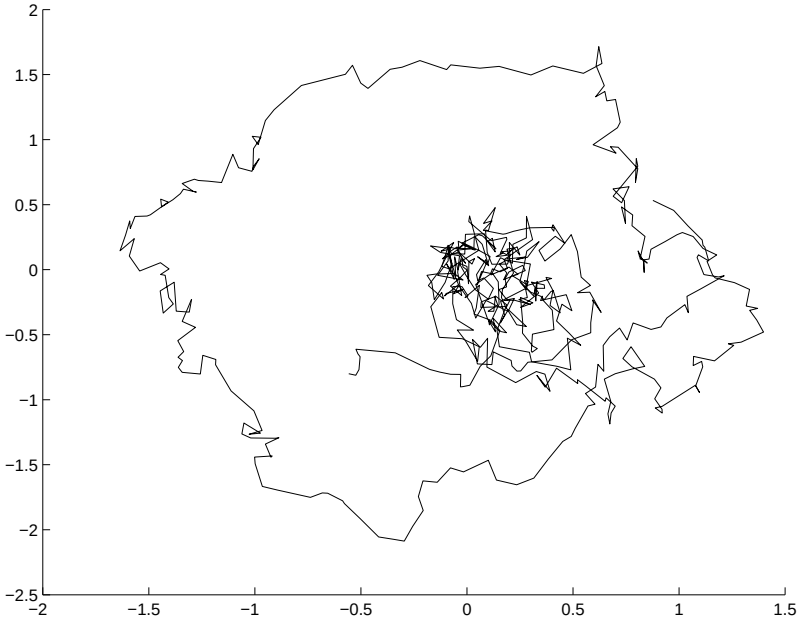


Figure 12: Sampled sequence in latent space, with noise

Question 7

If we compare the log likelihoods of the data under each of our models, we can see that 2D FA is a much better model for the data than 1D GTM. 2D LDS improves on the performance of FA by modelling the change in latent variables over time.

Model	Log likelihood
2D FA	-1.521×10^4
1D GTM	-7.939×10^4
2D LDS	-1.210×10^4

Question 8

In a HMM with a discrete state space, the posterior distribution is a mixture of gaussians, and we are free to change the parameters of each one to maximise likelihood. We will have a certain probability of transitioning from each component to each other component. The ideal fit will have means along the trajectory of our walk cycle, variance related to the variance between successive cycles, and a transition matrix which gives us low probabilities of all transitions except to the current state and the next state along the cycle. The relative probabilities of staying in the current state and moving to the next state would be related to the number of discrete states and the speed of the walk cycle.

Appendix A - Additional Code

```
function [] = writeFigurePDF(Fig, FileName)
% writeFigure writes the current figure to a pdf file
% INPUT Fig: [optional] the figure to write; default is the current figure.
%       FileName: a string containing the path of the file to save to.

if nargin < 2
    FileName = Fig;
    Fig = gcf;
end
FileNameRoot = regexp(FileName, '(.*)\.pdf$', 'tokens');
FileNameEPS = [FileNameRoot{1}{1} '.eps'];
print(Fig, '-depsc', FileNameEPS);
[Status, ~] = unix(['epstopdf ' FileNameEPS]);
if Status ~= 0
    fprintf(2, 'Error running epstopdf!\n');
end
```

Listing 21: writeFigurePDF.m

```
function [F, C] = makeLogFile(FileName)
% makeLogFile opens a log file and makes an onCleanup object to close it.
F = fopen(FileName, 'w');
C = onCleanup(@()fclose(F));
```

Listing 22: makeLogFile.m