% This is an implementation of the algorithm described in the Computational

% cognition cheat sheet titied "The Indian Buffet Process."

% Written by Ilker Yildirim, September 2012.

**##A is a coefficient matrix 🡪 X = ZA + error (Latent Feature Linear Gaussian) X = n\*p Z = n\*(K) A = K\*p**

num\_objects = 100;

object\_dim = 36;

sigma\_x\_orig = .5;

**# Ignore above**

% The sampler

% Compute Harmonic number for N.

**#Harmonic function is part of the prior function (X)**

HN = 0;

for i=1:num\_objects HN = HN + 1/i; end;

E = 1000;

BURN\_IN = 0;

SAMPLE\_SIZE = 1000;

#Graphical model of IBP

% Initialize the chain.

sigma\_A = 1;

sigma\_X = 1;

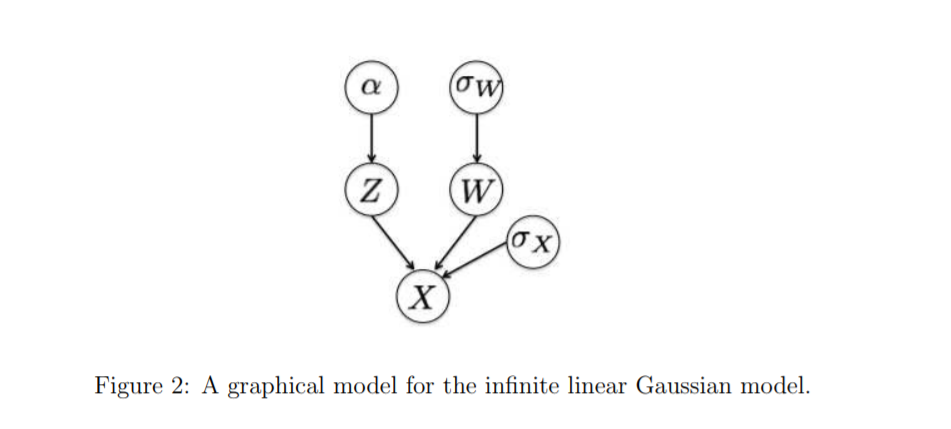
alpha = 1;

K\_inf = 10;

**# Initiate prior (Z)**

[Z K\_plus] = sampleIBP(alpha, num\_objects);

**# More detail on sampleIBP**



**#Store chain of Z K and alpha**

chain.Z = zeros(SAMPLE\_SIZE,num\_objects,K\_inf);

chain.K = zeros(SAMPLE\_SIZE, 1);

chain.sigma\_X = zeros(SAMPLE\_SIZE, 1);

chain.sigma\_A = zeros(SAMPLE\_SIZE, 1);

chain.alpha = zeros(SAMPLE\_SIZE,1);

s\_counter = 0;

**# LOOP each entry**

for e=1:E

% Store samples after the BURN-IN period.

if (e > BURN\_IN)

s\_counter = s\_counter+1;

chain.Z(s\_counter,:,1:K\_plus) = Z(:,1:K\_plus);

chain.K(s\_counter) = K\_plus;

chain.sigma\_X(s\_counter) = sigma\_X;

chain.sigma\_A(s\_counter) = sigma\_A;

chain.alpha(s\_counter) = alpha;

end;

disp(['At iteration ', num2str(e), ': K\_plus is ', num2str(K\_plus), ', alpha is ', num2str(alpha)]);

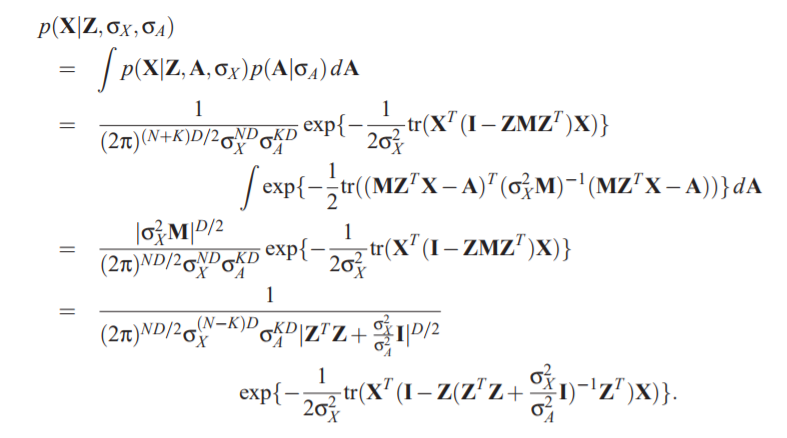
for i=1:num\_objects

% The matrix M will be handy for future likelihood and matrix

% inverse computations.

**M** = (Z(:,1:K\_plus)'\*Z(:,1:K\_plus) + (sigma\_X^2/sigma\_A^2)\*diag(ones(K\_plus,1)))^-1;





**K\_Plus = TOTAL FEATURE**

for k=1:K\_plus

% That can happen, since we may decrease K\_plus inside.

if (k>K\_plus)

break;

end;

if Z(i,k) > 0

% Take care of singular features

if sum(Z(:,k)) - Z(i,k) <= 0

Z(i,k) = 0;

Z(:,k:K\_plus-1) = Z(:,k+1:K\_plus);

K\_plus = K\_plus-1;

M = (Z(:,1:K\_plus)'\*Z(:,1:K\_plus) + ...

(sigma\_X^2/sigma\_A^2)\*diag(ones(K\_plus,1)))^-1;

continue;

end;

end;

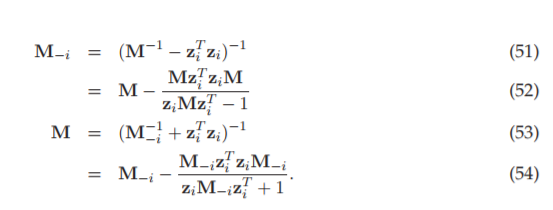
% This equations are for computing the inverse efficiently.

% It is an implementation of the trick from Griffiths and

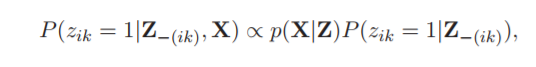
% Ghahramani (2005; Equations 51 to 54).

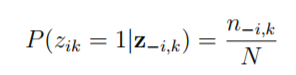
M1 = calcInverse(Z(:,1:K\_plus), M, i, k, 1);

M2 = calcInverse(Z(:,1:K\_plus), M, i, k, 0);



% Compute conditional distributions for the current cell in Z.





Z(i,k) = 1;

P(1) = likelihood(X, Z(:,1:K\_plus), M1, sigma\_A, sigma\_X, K\_plus, num\_objects, ...

object\_dim) + log(sum(Z(:,k))- Z(i,k)) -log(num\_objects);

Z(i,k) = 0;

P(2) = likelihood(X, Z(:,1:K\_plus), M2, sigma\_A, sigma\_X, K\_plus, num\_objects, ...

object\_dim) + log(num\_objects - sum(Z(:,k))) - log(num\_objects);

P = exp(P - max(P));

% Sample from the conditional.

if rand < P(1)/(P(1)+P(2))

Z(i,k) = 1;

M = M1;

else

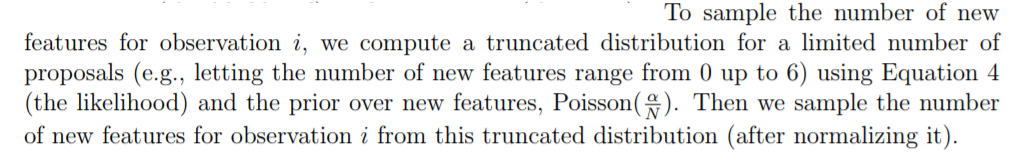
Z(i,k) = 0;

M = M2;

end;

end;

**% Sample the number of new dishes for the current object.**



trun = zeros(1,5);

alpha\_N = alpha / num\_objects;

for k\_i=0:4

Z(i,K\_plus+1:K\_plus+k\_i) = 1;

M = (Z(:,1:K\_plus+k\_i)'\*Z(:,1:K\_plus+k\_i) + (sigma\_X^2/sigma\_A^2)\*diag(ones(K\_plus+k\_i,1)))^-1;

trun(k\_i+1) = k\_i\*log(alpha\_N) - alpha\_N - log(factorial(k\_i)) + ...

likelihood(X, Z(:,1:K\_plus+k\_i), M, sigma\_A, sigma\_X, K\_plus+k\_i, num\_objects, object\_dim);

end;

Z(i,K\_plus+1:K\_plus+4) = 0;

trun = exp(trun - max(trun));

trun = trun/sum(trun);

p = rand;

t = 0;

for k\_i=0:4

t = t+trun(k\_i+1);

if p < t

new\_dishes = k\_i;

break;

end;

end;

Z(i,K\_plus+1:K\_plus+new\_dishes) = 1;

K\_plus = K\_plus + new\_dishes;

end;

**% Metropolis steps for sampling sigma\_X and sigma\_A**

M = (Z(:,1:K\_plus+new\_dishes)'\*Z(:,1:K\_plus+new\_dishes) + ...

(sigma\_X^2/sigma\_A^2)\*diag(ones(K\_plus+new\_dishes,1)))^-1;

l\_curr = likelihood(X, Z(:,1:K\_plus+new\_dishes), M, sigma\_A, sigma\_X, ...

K\_plus+new\_dishes, num\_objects, object\_dim);

if rand < .5

pr\_sigma\_X = sigma\_X - rand/20;

else

pr\_sigma\_X = sigma\_X + rand/20;

end;

M = (Z(:,1:K\_plus+new\_dishes)'\*Z(:,1:K\_plus+new\_dishes) + ...

(pr\_sigma\_X^2/sigma\_A^2)\*diag(ones(K\_plus+new\_dishes,1)))^-1;

l\_new\_X = likelihood(X, Z(:,1:K\_plus+new\_dishes), M, sigma\_A, pr\_sigma\_X, ...

K\_plus+new\_dishes, num\_objects, object\_dim);

acc\_X = exp(min(0, l\_new\_X - l\_curr));

if rand < .5

pr\_sigma\_A = sigma\_A - rand/20;

else

pr\_sigma\_A = sigma\_A + rand/20;

end;

M = (Z(:,1:K\_plus+new\_dishes)'\*Z(:,1:K\_plus+new\_dishes) + ...

(sigma\_X^2/pr\_sigma\_A^2)\*diag(ones(K\_plus+new\_dishes,1)))^-1;

l\_new\_A = likelihood(X, Z(:,1:K\_plus+new\_dishes), M, pr\_sigma\_A, sigma\_X, ...

K\_plus+new\_dishes, num\_objects, object\_dim);

acc\_A = exp(min(0, l\_new\_A - l\_curr));

if rand < acc\_X

sigma\_X = pr\_sigma\_X;

end;

if rand < acc\_A

sigma\_A = pr\_sigma\_A;

end;

% Sample alpha from its conditional posterior.

alpha = mygamrnd(1+K\_plus, 1/(1+HN),1);

% Save the chain at every 1000th iteration.

if mod(e,1000) == 0

s = strcat('chain\_ibp\_',num2str(e));

save(s, 'chain');

end;

end;