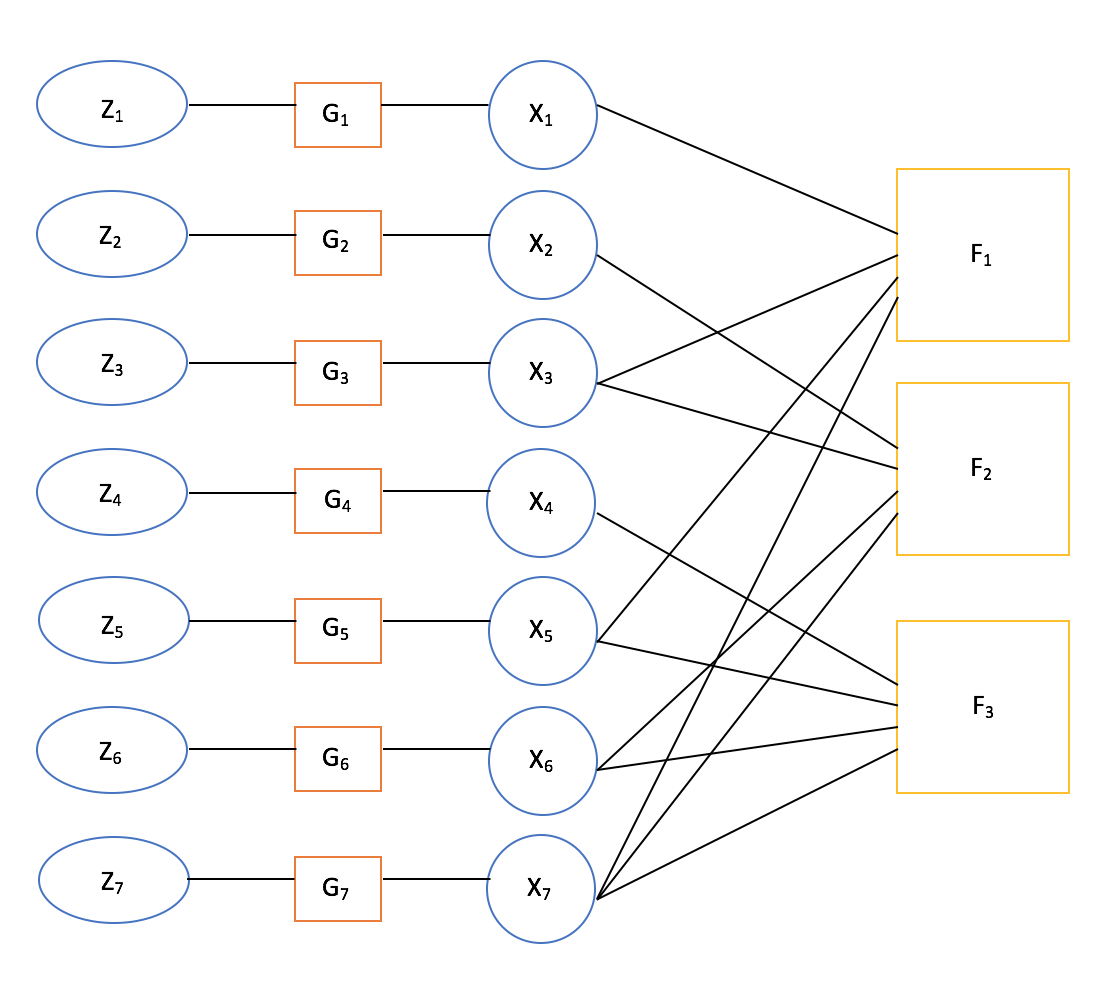
Graphical Models Project 1:

Sum/Max-Product Decoding

# Project Setup

Below is the graphical model to represent the system flow with variable nodes and functions according to the Hamming Code (7,4) encryption.

Converting the graph into probabilistic formulae gives the following:

This implies conditional independence, . For explanation, it says to have knowledge about X1 as well as the probability distribution of Z1, one can compute Z1 and the other Xi will not provide any more useful information for the computation. By definition of independence, this can be equated to the following:

# Implementation Notes

The system setup has been divided into three main components (i.e. classes), namely the transmitter, the decoder, and the simulator.

The simulator initializes constant variables including the parity, decoding and code generator matrix, as well as the number of iteration and list of standard deviations for testing. It calculates and stores bit errors as it iterates through the variances and repeatedly calls transmission and decoder.

The transmitter is responsible for transferring the message through the channel which includes (optional) Hamming encoding, extension of range through bit inversion, and addition of noise (*Ni*) from a Gaussian distribution. The input is *m*, which based on the optional Hamming encryption is either the original message or the Hamming encoded message. The output is *z*, which is the observed code word for which the decoder will try and find the most likely possibility of *X*.

The decoder takes the following as input:

* *z*, a 7-entry array: (z1, z2, z3, z4, z5, z6, z7) as described in the setup
* *use\_maxproduct* to specify the algorithm used for decoding (1 for max product, 0 for sum product)
* *std\_deviation* is the known standard deviation of the channel Ni, also described in the setup

The graph's messages can be partitioned as follows:

1. The messages from the Pz|x function nodes to variable (Xi) nodes -- these are cycle free and therefore are statically described in an array which we arbitrarily call m
2. The messages from the variable (Xi) nodes to the function (fi) nodes, which may or may not be part of cycle (depending if they connect to 1 vs 2+ function nodes). Each variable can potentially send a message to any function node, and these messages change at each iteration of a cycle. We describe these messages in a sparse matrix V (for variable node) -- sparse because the entry for any unconnected Xi->fi node will be zero.
3. The messages from the function (fi) nodes to the variable (Xi) nodes. We describe these messages using a matrix F. Similarly to V (and for the same reasons), F is dynamic and sparse, and its non-zero entries may or may not belong to cycles.

The returned value 'x' is the decoder's best guess for the original code word sent by the transmitter. To compute x, we take the summary message (multiplication of all incoming messages at the variable (xi) nodes), after a prescribed number of iterations.

Finally, because underflow was observed (small fractions multiplied many times become too small to be carried in floating point numbers) we take the log of all original messages, and perform all computations in the log domain.

# Running Instructions & Dependencies

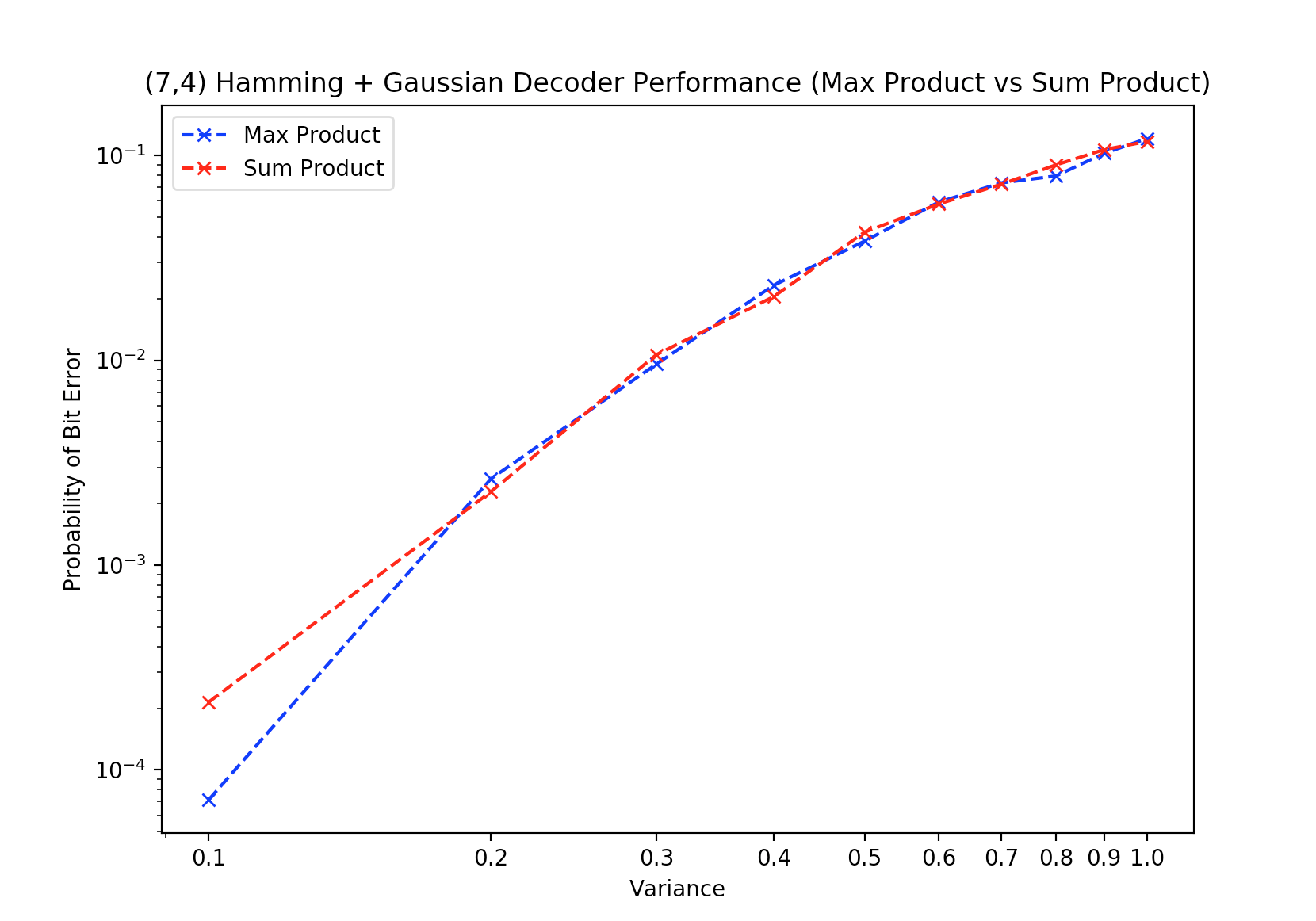
To run this system, you will need:

* Python 3
* Numpy
* Scipy
* Matplotlib

Execute the command *python simulation.py* or *python3 simulation.py* (if using OS X).

# Results

The graph displays the relation between probability of bit error and variance for an execution of the system with 2000 code words and 10 iterations per decode run.



The graph shows that as variance increases, so does the probability of bit error. However, the results follow through with our expectations that max product generally performs better than sum-product.