When coding HMMs, the problem with underflow arises as we multiply probabilities together. With many HMM algorithms logs are used to deal with this problem, but as discussed in class there is no easy way to take the log of a sum, and sums are part of the forward algorithm recurrence relation. Some researchers take an equivalent log, but it is more common to employ scaling to avoid underflow. A common scaling scheme is normalizing each cell: dividing over the sum of the values in the column.

In order to have an easy way to test your HMM. I suggest a different scaling method:

With HMMs, odd scores are frequently calculated to compare the probability of the sequence being generated by the HMM model to the probability of that same sequence being generated randomly. For comparison with our HMM model, we can assume a null model that emits random sequences such that each symbol(nucleotide) has the same probability of being emitted. In this case, emissions probabilities are converted to odds by dividing by the probability of emitting the symbol by chance (ie: dividing by 1/NUM_SYMBOLS). This is the same as multiplying by NUM_SYMBOLS. Note that the emissions probability is a factor of every cell in the array, so what you do to the emissions probabilities you do to every cell. Therefore, using the uniform probability model below, with odds scores for emissions (employed by scaling each cell of your scoring array by NUM_SYMBOLS), you will get equal likelihood of the model generating the sequence ending in each of the 3 states: S1, S2 and S3. This will be true of every subsequence, so your entire scoring array will have .333333333333333333333333333 stored in each cell. This will be true of every sequence you test with, including the long sequence we used for the BLAST assignment.

In sum, you can deal with underflow as you fill in the scoring array by multiplying each cell by NUM_SYMBOLS. Then you can test your forward algorithm by temporarily changing all emissions, transitions and start states to equally likely (uniform probability model). When you run this model on any string, you should get .333333333333333333333333333 in each cell, with a total score of 1.

```
//test situation: all emissions, all transitions and all start states are equally likely (uniform probability model)

const double transProb[NUM_STATES][NUM_STATES] = {
                    //              S1                              S2                              S3
                     { .333333333333333333333333333, .333333333333333333333333333,  .333333333333333333333333333 }, // S1
                     { .333333333333333333333333333, .333333333333333333333333333,  .333333333333333333333333333 }, // S2
                     { .333333333333333333333333333, .333333333333333333333333333,  .333333333333333333333333333 }  // S3
                                                                                                                   };
                                                    //      a     c     t     g
const double emitProb[NUM_STATES][NUM_SYMBOLS] = {
                                                        { .25,  .25,  .25,  .25 }, // S1
                                                        { .25,  .25,  .25,  .25 }, // S2
                                                        { .25,  .25,  .25,  .25 }  // S3
                                                                                                       };

                                                   //   S1                           S2                        S3
const double initProb[NUM_STATES] =  {  .333333333333333333333333333, .333333333333333333333333333,  .333333333333333333333333333 };
```