Alignment of pairs of sequences

Outline:

I. Relationship between the tour and sequence alignment

II. What is Blast?
- A. Overview
- B. Blast alignment scoring parameters – nucleic acids
- C. Blast alignment scoring parameters – amino acids
- D. Filtering
- E. Word-based searches

III. Interpretation of Blast results

I. Relationship between tour and sequence alignment

We were left with two mysteries. The first mystery, you'll recall, was why a nucleotide search through BlastN failed to come up with a match while a protein search through BlastX found many good matches. Perhaps that's not so mysterious. In any event, the comparison between DG29 and the lethal factor gene (in the second mystery) may provide you with some insight into the problem. When translated (i.e., using BlastX), DG29 matches exactly the protein encoded by the lethal factor gene. But there are many differences in the DNA. Your knowledge of the translation process and the nature of the genetic code is probably enough to dispel any concerns you may have concerning the first mystery.

The second mystery was more puzzling. The alignment between DG29 and the lethal factor gene looks awfully good, and yet BlastN couldn't find what is obvious by eye. Why not? To answer that question, we need to understand how Blast works. It obviously does not work the way we do if we were in charge of the search, and that's because it has different priorities. It was not designed to compare just one sequence with another but rather to compare one sequence with millions of others. It was designed for speed. We need to understand the non-obvious costs entailed by its design.

II. What is Blast?

II.A. Overview

All BLAST (Basic Local Alignment of Sequences Tool) knows how to do is to align two sequences as best it can and score its best effort. How you interpret that information is up to you. Do not confuse the following pairs of concepts:

- Alignment of pairs of sequences vs multiple sequence alignment
  Alignment of pairs of sequences (which is what Blast does) is a much easier problem than multiple sequence alignment. The former is useful in identifying sequences. The latter is useful in finding conserved regions in a set of similar sequences. We'll consider here only ***pairwise*** alignment.
- Global alignment vs local alignment
  Global alignment compares two sequences throughout their lengths. This is useful if you have reason to believe that the two sequences ought be similar from one end to the other.

This is clearly not the case when you're comparing a short DNA sequence against an entire genome, and it is seldom the case with protein comparisons either. We'll consider here only ***local*** alignment, in which the ends of the two sequences being compared are not forced into the alignment.

How does BlastN decide whether two sequences match? The program declares a match if it can find a good alignment of the two sequences (you define "good"). Let's consider the problems Blast faces by considering the two pairs of sequences below. It seems pretty obvious that the first pair match well (identity fraction of 16/17) and the second pair match not much better than what you'd expect from two random sequences (identity fraction of 6/21).

a. `AATATTGACGCTTTACT`
`||||||  ||||||||||`
`AATATTCACGCTTTACT`

b. `TTTACTACATCAGTCCATCGG`
`|  | |  | |  |`
`GTACTTCCGATGGACTGATGT`

But wait! The latter pair actually isn't a bad match. If you invert the second sequence (i.e., take it's complement and turn it upside down) and shift it a bit you get:

b. `TTTACTACATCAGTCCATCGG`
`||||||||||||||||`
`ACATCAGTCCATCGGAAGTAC`

Pretty good! Clearly we need some machine help, to contort the sequences in all possible ways. And what about gaps? Which of the following would you consider to be the ***best*** match of GACT in the given sequence:

`GTAGCGTGATTAATACTGACA` `GTAGCGTGATTAATACTGACA` `GTAGCGTGATTAATACTGACA`
`G-A-C-T` `GACT` `GA------CT`

There's no right answer to that question. It depends on how much weight you place on discontinuity and on mismatches.

We ask Blast to consider inverted sequences, mismatches, gaps,… and to do this for the several billion possible matches amongst known sequences. In Mystery #2 you may have carped how BlastN didn't give you the answer you wanted... you should instead have marveled that it gave you any answer at all!

II.B. Blast alignment scoring parameters – nucleic acids

Blast is derived from a more general procedure, known as the Smith-Waterman algorithm, which examines every possible alignment of one sequence with respect to another, scoring the alignments according to a set of penalties for mismatches and gaps and a reward for matches.

The details are most easily understood by example. Suppose you want to find the best alignment of query sequence (the sequence you provide) within a subject sequence (also called target, the sequence found by Blast):

Query: `AGATACCTACA` **Subject$_1$:** `TTAGATAAGCCTAGAG`

We'll use the (rather atypical) scoring parameters of:

Match reward = +1 per match
Mismatch penalty = -2 per mismatch
Gap existence (or opening) penalty = -3 per gap
Gap length (or extension) penalty = -2 per letter

With these scoring parameters, the alignment:

```
#1    AGATA--CCTACA     #2    AGATA
        |||||  ||||  |          |||||
    TTAGATAAGCCTAGAG       TTAGATAAGCCTAGAG
```

would produce the score of:

#1: (10 matches)(+1) + (1 mismatch)(–2) + (1 gap)(–3) + (2 gap letters)(–2)

= +1

#2: (5 matches)(+1)

= +5

So with this scoring system, Blast would rank the second possibility over the first.

You can specify what values you want for the reward and penalties, but few people do, instead accepting the default values given by an implementation of Blast.

SQ1. Copy the sequence below and use nucleotide Blast (BlastN) to find at least somewhat similar sequences within all available sequences (click Others for Database; click Somewhat similar for Program Selection).

ACTCTTCTGGTCCCCACAGACTCAGATAGAATCCAC
CATGAAAGTGCTGTCTCCTGCCGACAAGACCAACGTC

…but before running the Blast, set the scoring parameters (click Algorithm Parameters at the bottom of the page). Set Match/Mismatch to +2/-3. Set Gap Costs to Existence 6, Extension 2. When you get the results, scroll down to the best hit and calculate yourself what you think the alignment score should. Then check your calculation by looking at the score report ("score = x bits (y)"). Your score should be the same as y. Is it? Play around with the scoring parameters until you're convinced you can predict the alignment score.

II.C. Blast alignment scoring parameters – amino acids (you can skip this if you like)

Protein amino acid sequences are scored in a very different fashion, with the motivation to take into account that some amino acid changes are more drastic than others, hence less common. Consider the example in **Fig 1**. Fewer than half the amino acids of the query match the subject sequence, but the remaining query-subject pairs are amino acids of the same type (e.g. both positively charged), and such amino acid changes are relatively common.

```
Query:          KLEIWWARE
(type of match) ++E+WW R+
Subject:      GSRVELWWGRQTV
```

+ charged	hydrophilic/uncharged	hydrophobic/aliphatic
K lysine	N asparagine	I isoleucine
R arginine	Q glutamine	L leucine
		V valine
– charged	small	M methionine
D aspartate	A alanine	
E glutamate	G glycine	

Figure 1: Sample alignment of amino acid sequences

Who's to decide what changes are common and which are not? And how can the frequency be quantified? This has been done in different ways. The most frequently used quantification are BLOSUM tables (***Bl***ock ***Su***bstitution ***M***atrices), an example of which is shown in **Fig. 2**. To make the BLOSUM62 table, a large number of conserved regions from proteins, with 62% identity, were compared to each other to determine what amino acid substitutions have taken place over evolutionary history.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Figure 2: BLOSUM62. Each number represents the $\log_2$ of the odds ratio of the observed to expected number of transitions of one amino acid to another. Since the table is symmetrical (the direction of change is irrelevant), only half of it is shown.

Each number within a BLOSUM table represents the relative likelihood that an amino acid at a conserved position within a protein changes to another amino acid (or remains unchanged). The numbers are calculated as follows:

1. The frequency of each amino acid-amino acid transition is rendered as an odds ratio. For example, for the transition from amino acid X to amino acid Y, or the reverse:

   $$\frac{\text{observed frequency of X-Y transitions}}{\text{predicted frequency of X-Y transitions if X \& Y appear by chance}}$$

   In the blocks of conserved protein segments with 62% identity, leucine (L) residues remain constant about 4% of the time (the observed frequency of L-L transitions is 4%), and the predicted frequency is the frequency that one residue is L (which for this common amino acid is about 10%) and the other is also L (also 10%). So the odds ration is: (0.04)/(0.1*0.1) = 4

2. The log base 2 is taken of the odds ratio, giving what is called the log-odds ratio, or the **lod** score, and that number is multiplied by an arbitrary scaling factor, 2, to increase the range of integer values (making the table easier to read). In the example above, this gives 2 * $\log_2[4]$ = 4, which is the number given in the BLOSUM62 table. Adding the **lod** scores is equivalent to multiplying the odds ratios, and since addition is faster than multiplication, the program becomes more efficient.

Positive numbers indicate transitions that occur more often than expected by chance, indicating selection for these transitions. The magnitude of the number indicates the significance to be attached to the figure. For example, the significance of the rare amino acid tryptophan appearing at the same position in an alignment by chance is much greater than the significance of a conserved leucine residue (11 vs 4 in the BLOSUM62 table).

Note that the values in the table do not arise from theory but from actual observation. You might expect, for example, that a transition from the negatively charged amino acid glutamate (E) to the positively charged amino acid arginine (R) would be selected against, but this is not the case.

Perhaps the fact that both are large often allows one to substitute for the other, despite their difference in charges.

Using this table, we're in a position to score an amino acid alignment. Considering the previous example shown in **Fig 1**, and reading the scores off the BLOSUM62 table shown in **Fig. 2**, we get a total score of 39, as shown in **Fig. 3**. Blast ranks alignments by scores obtained in this way. The amino acid residues to either side of the aligned region do not contribute to the score because adding them to the alignment would make the total score lower.

Query	Subject	BLOSUM62 score
	G	
	S	
K	R	2
L	V	1
E	E	5
I	L	2
W	W	11
W	W	11
A	G	0
R	R	5
E	Q	2
	T	
	V	
TOTAL		39

Figure 3: Scoring of alignment of Fig. 1.

SQ2. Go back to the main Blast page of NCBI, scroll down to Specialized Blast, and select "Align two (or more) sequences". Click the BlastP tab (to compare protein sequences), and paste the query shown in the example on the page 3 into the first box and the subject into the second box. Click Algorithm Parameters, and notice that the scoring matrix used is BLOSUM62. Set Compositional Adustments to "No adjustment" (so that it will use the table without modification) and unclick the Automatically Adjust Parameters box (so that Blast will listen to the modifications you make in the algorithm parameters!). Finally click Blast to run the comparison.

When you get the results, compare the score (in parentheses) with what you calculate yourself from the BLOSUM62 table on the previous page. Play around with the query and subject sequences until you're convinced you can predict the alignment score.

II.D. Filtering

Searching a huge database for sequence similarities, as Blast does, is useful so long as the sequence you're searching for arises only when true functional similarity exists. You would be disappointed to find hits that are trivial, arising just by chance. Ordinarily, you avoid hits do to chance by giving a query that is so long that the odds of finding a random hit are prohibitive. But there are exceptional cases. Let's try one.

SQ3. Fragile X syndrome, the most common form of inherited mental retardation, is determined by a mutation in the FMR-1 gene. Find its DNA sequence by going to the main NCBI page, selecting Nucleotide as the Search field, and M67468 (the accession number of the gene) in the for field. You should come to the record for the FMR-1 gene. Hold on to that page, and in a separate browser window, go back to NCBI, and get to Nucleotide Blast. Enter the accession number M67468 in the big white box and specify a query subrange of From 20 To 120. Finally, run blastn (select Somewhat similar sequences, and click BLAST).

You probably got "No significant similarity found." How could that be? You got the sequence directly from GenBank. How could the sequence not find itself?

SQ4. Go back to the FMR-1 record. Is there anything special about the nucleotides of the sequence from 20 to 120?

This is normal instance of the FMR-1 gene. In people with Fragile X syndrome, this region can be an order of magnitude larger!

SQ5. Go back to the Nucleotide Blast page and display the Algorithm Parameters. Notice that there is a section called Filters and Masking. Uncheck both of the two filter boxes, and run the Blast search again. Any difference?

SQ6. Go back to the Nucleotide Blast page again. Restore the filtering (check both filter boxes), but this time run the search with the entire FMR-1 sequence (delete the numbers in the From and To boxes). Now what do you get? Pay special attention to the sequence from 20 to 120.

The filter boxes take care of two different situations:

- Regions of low complexity
  Complexity is a term that describes how much information is required to specify a sequence. If you have a random sequence (e.g. `CTCACTATCGAACTCTCGTA`) there's no useful way to avoid specifying each letter individually. On the other hand, if you have a sequence with repeats, such as `AAAAAAAAAAA`, you can specify just one letter and say repeat it so many times. There is less information in the second case than in the first. The second case is less complex. Since regions of low complexity arise in natural DNA sequences far more frequently than you'd expect by chance (because errors in DNA replication tend to produce repeated sequences), Blast gives you the opportunity to filter them out.
- Species-specific repeats
  There are also sequences that occur in humans thousands of times. They are not of low complexity but rather arise by transposition, the movement of a chunk of DNA to a new location. Other genomes also have repeated sequences of this type. Blast gives you the opportunity to filter them out, so long as there exists an existing table of such repeated sequences. There is such a table for the human genome, but you can't count on one in other organisms.

In the Blast search you did in SQ6, you probably saw the repeated sequence (GAGGAG…) represented in lower case letters,… it was not filtered out, even though you had filtering of low complexity regions on, so long as there were neighboring high-complexity regions of similarity. But try this:

SQ7. Repeat SQ6, but this time unchecking the box labeled "Mask for lookup table only". How does this change the results?

What is this "lookup table" they're talking about? To understand this, and much else about Blast, you need to understand what lies under the hood, discussed in the next section.

II.E. Word-based searches

The Smith-Waterman algorithm guarantees the best local match (according to the specific set of parameters used), but it does so by demanding huge chunks of computer memory and time. If you had used the unmodified Smith-Waterman algorithm to run the search for FMR-1-like sequences (SQ3), you'd still be waiting for the result and would probably continue to wait for days to come. Instead, you used a modification of the Smith-Waterman algorithm, called Blast,

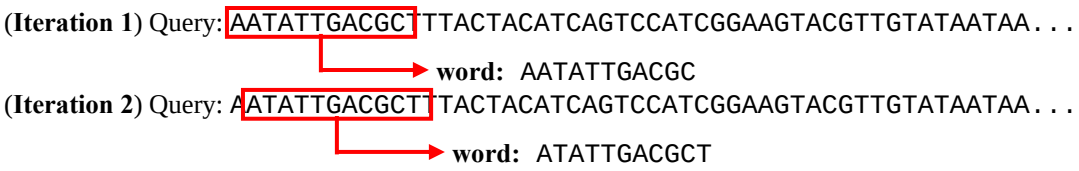

Figure 4: Extraction of words from query sequence, beginning from the beginning of the sequence and proceeding by sliding the window to the right, to the end.

which can give the result you want in about 1/1000th the time. The main trick comes from the realization that the majority of the query-subject comparisons done with a large database are with subject sequences that are completely dissimilar. Better to throw these sequences away as soon as possible and move on. This is what Blast tries to do.

Blast defines a sequence as hopeful if it has a minimal exact match between a part of the query and a part of the subject. Exact matches are easy to find – no complicated scoring systems, no introduction of variable length gaps – either the sequence matches or it doesn't. If Blast finds a minimal match, called a *word*, then and only then does it resort to the time-consuming Smith-Waterman algorithm to extend the match (perhaps with gaps and mismatches) as far as it can go.

The algorithm used by Blast proceeds by the following steps:

1. (Optional) Filter the query sequence to remove repetitive regions, as explained in Section II.D.
2. Find all exact matches between the query sequence and the target sequence In BioBIKE-speak: `FOR-EACH word FROM` *beginning-of-query TO end-of-query*
    - **a.** Extract a subsequence from the query, called a **word**, by sliding a window along the length of the query, as shown in Figure 4.
    - **b.** Find all *exact* matches of the word in the target sequence. If one or more matches are found, store them in a table, go back to Step **a** and get the next word from the query.
    - **c.** Repeat steps **a** and **b** for each possible word.
3. Extend all exact matches. `FOR-EACH word-match IN` *table-from-previous-step*
    - **a.** (Optional) Filter the query sequence (again) to remove repetitive regions, as explained in Section II.D.
    - **b.** Use a modified Smith-Waterman algorithm to extend the word match in both directions, according to a set of reward and penalties.
    - **c.** Calculate the expected number of matches as good or better than the final match (the Expect Score; see next section).
    - **d.** Save those matches whose Expect Scores are better than a given threshold.
    - **e.** Repeat Steps **a** through **d** until the word-matches in the table are exhausted.
3. Rank the saved matches by their Expect Scores.
4. Display the top matches.

Step 2 of the procedure can save an enormous amount of memory and calculation time, depending on the size of the word. If you set the word-size to 1 (a very bad idea, if indeed it's possible), you would apply the full Smith-Waterman algorithm to every match of a single nucleotide, achieving almost no savings in time. On the other hand, if you set the word-size to 100, you will have a very rapid search but one that finds only those matches with a huge region of identity. The NCBI implementation provides a default value for the word size, which few users know about or care to change.

SQ8: What word size would you use in order to detect the match shown in Figure 2?

SQ9. Go back to the main Blast page of NCBI, scroll down to Specialized Blast, and select "Align two (or more) sequences". Click the BlastN tab, if necessary, to get you to nucleotide-nucleotide comparisons. Make up a random 20-nucleotide sequence and paste it into both boxes. Specify "Somewhat similar… BlastN" for the Program Selection, and open up Algorithm Parameters, and uncheck the Short Queries box. Now run Blast – presumably it will detect the similarity. Introduce a single nucleotide change that causes Blast to miss the 19-nt of identity. Finally, modify the word size so that Blast does find the 19-nt of identity.

III. Interpretation of Blast results

The Smith-Waterman algorithm will find you the best match to a query sequence. Blast will usually find you the best match to a query sequence. But in either case, "best" may be awful. How do you decide how good a match needs to be before it's worth looking at it? (and be assured, it is you who needs to decide this).

One simple minded way is to look at the percent identities. 100% identity between two sequences sounds very good. 70% identity sounds considerably worse. But this approach is highly misleading. Consider the following two sequence comparisons:

A) 100% identical

```
AG
||
AG
```

B) 70% identical

```
GTGGGTTCGGTTTGTATTGAAAACGTTGAGGGGAATCCCCATCTGAGGTGG
  |||||| |||| |||||||  |||||| ||| |||| | |  | | | |
AAGGGTTCAGTTTTTATTGAAATCGTTGATGGGTATCCACCTGCGCGCTAG
```

There's no doubt that the second is more impressive, i.e. not expected to have arisen by chance, hence indicative of biological significance. That last description seems to get to the heart of the issue: how can we judge whether a match is expected to have arisen by chance? We need to talk about how to quantify expectation.

Suppose you're following Blast as it compares a 100-nucleotide sequence with a random 100,000-nucleotide sequence (**Fig. 5)**. The first step is to extract a word from the query (let's say that the word size had been set to 11 nucleotides). How often would we expect to see such a match, a match of this particular 11 nucleotides and the first 11 nucleotides of the 100,000-nucleotide sequence? Not much, sure, but we need a number. Let's simplify the situation and say that the probability of a match of one nucleotide is ¼ . If so, then we get an expectation of $(¼)^{11}$, aabout 1 time in 4 million. That doesn't sound like much, but consider we have the very same expection of nucleotides 2 through 12, 2 through 13, and so forth, all the way up to 99,990 through 100,000. They're all the same expectation, so I can add them up: 99,990 * $(¼)^{11}$.

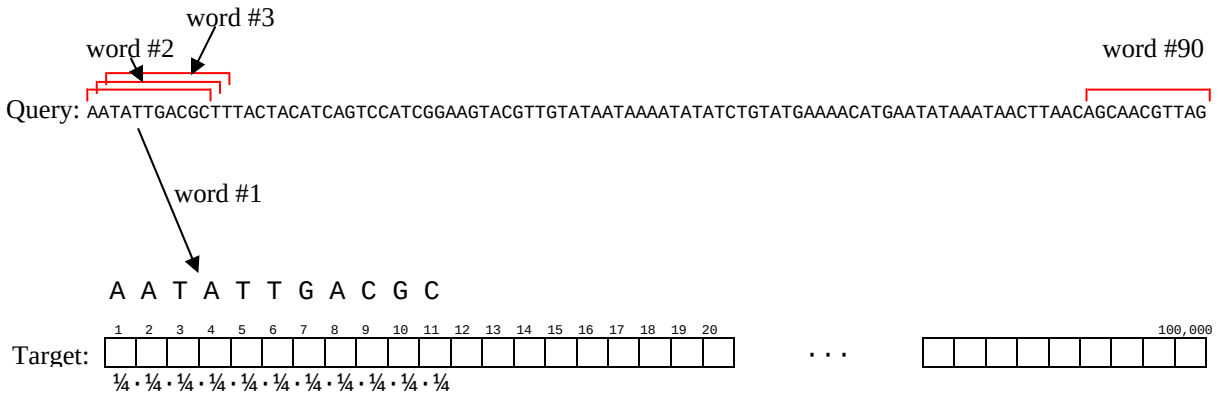

Figure 5. Probability of match of specific word with specific region of target.

That's still a relatively low expectation, now one chance in about 40, but we're not done yet. Everything I've said is just as true for word #2, and word #3, and word #4,… all the way to the last word #90. So the expectation is 90 * 99,990 * $(¼)^{11}$, which is about 2. An 11-nucleotide match is far from unexpected. I'd expect such a match twice just by chance!

More generally, I can calculate the expected number of matches (**E**) by taking into account the probability (**p**) of a match of a single nucleotide, the number (S**)** of nucleotides in a single match, and the lengths (**m** and **n**) of the query and the target:

$$\mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot \mathbf{p}^{\mathbf{S}} \qquad (= 100 \cdot 100{,}000 \cdot ¼^{11} \text{ in the previous example})^{*}$$

$$= \mathbf{m} \cdot \mathbf{n} \cdot e^{\ln(\mathbf{p}) \cdot \mathbf{S}}$$

This is fine if the probability of a nucleotide match is constant and if mismatches and gaps are forbidden, but BlastN is more complicated. This means that the expression has to be modified. I don't understand half the math necessary to tell you how the ultimate expression arises, but you can see that this expression used by BlastN to calculate expected frequency is pretty close to the expression we derived intuitively from first principles. In the actual expression:

$$\mathbf{E} = \mathbf{K} \cdot \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$$

Where **E, m**, and **n** have the same meanings as before, **K** and **λ** are empirically derived constants (the latter depending on the alignment scoring system used), and **S** is the score from all the matches, mismatches, and gaps. To minimize the number of parts in this equation without intuitive meaning, **K**, **λ**, and **S**, may be combined into a single quantity:

$$\mathbf{S'} = [\lambda \mathbf{S} - \ln(\mathbf{K})] / \ln(2)$$

to get:

$$\mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot 2^{-\mathbf{S'}}$$

S' is what is called the bit score, how many base-two digits of information are represented by the match. It is relatively independent of the alignment scoring system, and you'll note that Blast reports this along with the raw score for every match.

---

* Actually, **m** and **n** are not the full lengths of the query and subject but rather their lengths reduced by the length of the match.

But it is the **E** score, also called the **Expect score**, that is most widely used to judge the quality of a match. Recapping (and extending a bit) what has gone before, the E score is:

(a) the number of matches you would expect to find

(b) of the same or better quality as what you actually observed

(c) if you used your query

(d) and a database the same size and composition as the one you used

(e) but randomized.

Going back to the two examples we considered earlier, the two nucleotide 100% match would have an expect score of about 10^{10} in a nucleotide database the size of GenBank, while the 50-nt 70% matach would have an expect score of about 10^{-10}. Clearly the second is more interesting than the first.

Still, the question of threshold remains. Many people draw the line at an E score of 10^{-3}, figuring that a match that would arise in a random database at this, but there are problems with any mindless, arbitrary cutoff, as we'll see below.

SQ10: Is the E score a probability? Does it make any sense to have E scores greater than 1?

SQ11: Generate a few random 50-nt DNA sequences. You can use BioBIKE to do this, using the RANDOM-DNA function and the LENGTH option. Then Blast these sequences against the nr (non-redundant) database (i.e. clicking Others as the Database), using BlastN as the program. What kind of E-values do you get?

SQ12: According to the formula for Expect Score, the value should vary according to the size of the database. Let's test this. Blast the sequence below against the nr database, using BlastN, then do the same, specifying bacteria as the Organism and then cyanobacteria as the organism. What are the E-values you get in each of these three cases? Do their relative values make sense?

```
GTGGGTTCGGTTTGTATTGAAAACGTTGAGGGGAATCCCCATCTGAGGTG
GTTGCTGGGTTGGCACTTGCAAGAATTGGAATACCGTGTGCATCAAGCCG
```

SQ13: And finally, hearkening back to Mystery #2, why couldn't Blast find what seemed by eye to be a wonderful match between the PCR fragment DG29 and a lethal toxin gene from *Bacillus anthracis*?