Sequence information UMF018
Examination 07-12-20

No aids allowed except for dictionaries. For information contact Tore Samuelsson, tel 773 3468 / mobile 0736-169629.

Sequence alignment (GK)

1. What is a z-score? Explain how a z-score can be used to indicate whether the similarity found when globally aligning two sequences is significant.

   [3p]

2. The PAM250 matrix is shown below. Comment on the scores between W and W; A and A; I and L; F and D.

```
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V
A  2
R -2  6
N  0  0  2
D  0 -1  2  4
C -2 -4 -4 -5  4
Q  0  1  1  2 -5  4
E  0 -1  1  3 -5  2  4
G  1 -3  0  1 -3 -1  0  5
H -1  2  2  1 -3  3  1 -2  6
I -1 -2 -2 -2 -2 -2 -2 -3 -2  5
L -2 -3 -3 -4 -6 -2 -3 -4 -2  2  6
K -1  3  1  0 -5  1  0 -2  0 -2 -3  5
M -1  0 -2 -3 -5 -1 -2 -3 -2  2  4  0  6
F -4 -4 -4 -6 -4 -5 -5 -5 -2  1  2 -5  0  9
P  1  0 -1 -1 -3  0 -1 -1  0 -2 -3 -1 -2 -5  6
S  1  0  1  0  0 -1  0  1 -1 -1 -3  0 -2 -3  1  3
T  1 -1  0  0 -2 -1  0  0 -1  0 -2  0 -1 -2  0  1  3
W -6  2 -4 -7 -8 -5 -7 -7 -3 -5 -2 -3 -4  0 -6 -2 -5 17
Y -3 -4 -2 -4  0 -4 -4 -5  0 -1 -1 -4 -2  7 -5 -3 -3  0 10
V  0 -2 -2 -2 -2 -2 -2 -1 -2  4  2 -2  2 -1 -1 -1  0 -6 -2  4
```

   [2p]

3. Draw a suffix tree for the string "ACTCTCA".

   [3p]

Patterns (TS)

4a. Explain the syntax of the following PROSITE pattern.

GAx(2,3)Cx[KR](2)[NQED]G

4b. What *flexibility* does the pattern in 4a have?

[2p]

5. Explain the concept of *specificity* in the context of PROSITE patterns. Present a likely explanation why the following two patterns are different with respect to specificity.

Zinc finger	C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H	
Asn Glycosylation	N - {P} - ST - {P}	(where {P} refers to any symbol except P)

[2p]

Profiles (TS)

6. For the simple nucleotide sequence alignment below, show the calculations leading to a sequence logo.

```
Sequence 1      AAC
Sequence 2      AAG
Sequence 3      ATT
Sequence 4      ACA
```

[4p]

Genome bioinformatics (TS)

7. Show with a drawing the relationship between the following concepts in shotgun sequencing and sequence asssembly : a) cloned fragments of genomic DNA, b) sequence reads, c) contigs and d) scaffolds

[2p]

8. List the different properties of eukaryotic genes that are exploited in the *Genscan* method of gene prediction.

[2p]

Databases (MDL)

9. Explain what is meant by *gene ontology*. What are the three organizing principles of gene ontology?

   [2p]

10. This is the schema for the Evofold track at the Table Browser from UCSC (which gives you RNA Secondary Structure predictions). What is the SQL syntax to retrieve the number of sequences that are in Chromosome 5 on the forward strand?

field	example	SQL type	info	description
bin	591	smallint(6)	range	Indexing field to speed chromosome range queries.
chrom	chr1	varchar(255)	values	Reference sequence chromosome or scaffold
chromStart	886773	int(10) unsigned	range	Start position in chromosome
chromEnd	886798	int(10) unsigned	range	End position in chromosome
name	608_0_-_96	varchar(255)	values	Name of item
score	96	int(10) unsigned	range	Score from 0-1000
strand	-	char(1)	values	+ or -
size	25	int(10) unsigned	range	Size of element.
secStr	(((((.....(((...)))))))))	longblob		Parentheses and '.'s which define the secondary structure
conf	0.97,0.98,0.99,0.99,0.9,0.7...	longblob		Confidence of secondary-structure annotation per position (0.0-1.0).

   [2p]

RNA bioinformatics (TS)

11. Below is an alignment of three non-coding RNA sequences. Suggest a secondary structure that is consistent with all sequences. On the basis of this structure, choose one pair of columns of the alignment that displays *covariation* (a pattern of mutation consistent with base-pair) and select one pair of columns that does not. For these two pairs calculate the *mutual information*.

    A. `GCGGGCUUUCUUUCCCCGC`
    B. `GAGGGCUUU-UUUCCCCUC`
    C. `GUGGCCUUUUUUUGCCCAC`

    [4p]

Molecular phylogeny (TS)

12. Consider the multiple alignment below of four nucleotide sequences. Use a method of *maximum parsimony* to deduce the most likely phylogenetic tree.

```
Human       GAACGGACTTCA
Mouse       GAACGGACTTGA
Frog        AACCGGGCTAGA
Zebrafish   AATCGGCCTACA
```

    [4p]

Perl programming (GK)

13 a) Suppose `$s = "TTCCTACAACT".` Do the following patterns match and, if so, what are the values of the match variables ($1, $2, etc.)?

i) `$s =~ /^(.*)C/ ;`
ii) `$s =~ /^([^A]*)A/ ;`
iii) `$s =~ /C(..C)/ ;`
iv) `$s =~ /(.{2})(.*)\1/ ;`

b) Describe how each of the following statements modifies $s, assuming that $s has the value "GTAGGTTATT" before each statement is executed.

i) `$s =~ s/.TT/CTT/g ;`
ii) `$s =~ s/A[TG]+/X/;`
iii) `$s =~ s/T[^A]*/G/g ;`
iv) `$s =~ tr/TA/GC/ ;`

[4p]

14. A five-residue motif, "GPGXX", is believed to be important for the elastic properties of spider silk. This motif occurs 64 times in UniProt entry SPD2_NEPCL (see attached sheet).

a) Write a Perl program that reads a UniProt file whose name is specified on the command line and prints out the number of occurrences of this motif in the UniProt entry.

[6p]

b) Suppose we want a program that finds what actual amino acid residues correspond to the "XX" positions in this motif (e.g. "QQ", "GY", etc.), counts how often each "XX" pair occurs, and prints the most common pair. The output of the program should look as follows:

```
QQ 25
RY 1
SA 12
SQ 1
GY 24
IA 1
Most common pair is QQ
```

i) Describe an approach for performing this task in Perl (you are not required to write any Perl code in answering this part of the question).

ii) Write a Perl program that performs this task.
(You may assume that the sequence can be read from the UniProt file in the same way as in part (a), so you do not need to repeat that code in your answer to this part of the question.)

[6p]

Hidden Markov Models, December 20, 2007

Remember: When you are asked to calculate something, it is the *solution* that is asked for, not only an answer.

15. Consider a game where, at each play of the game, you either win 1 euro with probability p or loose 1 euro with probability $1 - p$. Say that you decide to participate in 10 rounds of the game. Let X_n= your profit (which is negative if you have lost more than won) after n plays, $n = 1, 2, ..., 10$.

    a) Explain why $\{X_n\}$ is a Markov chain.
    b) If your profit after 6 rounds is -2, what is the probability that your profit is 0 after 10 rounds? (4p)

16. Two protein sequences
    x=WHANRIGFLSAK and y=PFETHARIINAVNDLQVTV
    have been aligned using the local pair HMM (see figure 4.3 formula sheet). The alignment is given by the π-path,
    $\pi = Begin, RX_1, S, RY_1, RY_1, RY_1, RY_1, S, M, M, X, M, M, Y, M, M, Y, Y, Y, M, RX_2, RX_2, RX_2, S, RY_2, RY_2, RY_2, RY_2, End$.
    Which part of x is aligned with which part of y? Draw the alignment of the subsequences so that you can see which symbol in x is aligned with which symbol in y. (2p)

17. Tim has two dice A and B. A is a regular square die with six sides numbered 1, 2, ..., 6, and B is a pyramid-shaped die with four sides numbered 1, 2, 3, 4. Die B is fair (all four outcomes are equally likely). For die A the outcomes 5 and 6 each have probability 0.3, and the rest of the outcomes has probability 0.1. Tim has produced a sequence of numbers by using these dice in the following way: For each roll of a die, he uses die A with probability 0.5 if the previous roll was done with die A, and die B with probability 0.8 if the previous roll was done with die B. For the first roll, die A is used with probability 0.8.

    The sequence x= [3 5 6 2 1 3], was generated this way. In the table below you have the result for the Viterbi algorithm on the sequence x.

i	$v_A(i)$	$ptr_i(A)$	$v_B(i)$	$ptr_i(B)$
1	0.08	0	0.05	0
2	0.012	A	0	B
3	0.0018	A	0	A
4	$9 * 10^{-5}$	A	0.000225	A
5	$4.5 * 10^{-6}$	B	$4.5 * 10^{-5}$	B
6	$9 * 10^{-7}$	B	$9 * 10^{-6}$	B

    a) What are the transition- and emisson-probabilities in this HMM?
    b) Show how $v_B(5)$ is calculated.
    c) Derive the path π^* which has the highest probability of having generated x. (7p)

18. The table below shows a part of a multiple alignment of 8 protein sequences. The columns marked by M_1, M_2,M_3, and M_4 are the four first match states in the profile HMM for this particular family of sequences.

-	-	-	-	N	S
P	Y	-	-	L	S
N	E	-	-	T	S
N	E	-	-	N	S
T	S	-	-	F	S
T	Q	-	-	N	T
A	D	L	G	A	S
P	-	-	-	G	T
M_1	M_2			M_3	M_4

a) Calculate estimates of the transition probabilities $a_{M_2M_3}$, and the emission probability $e_{M_4}(S)$. Use simple pseudo-counts (Laplace rule).
b) Why are pseudo-counts used in the parameter estimation of a HMM (for instance in the profile HMM)? (3p)

```
ID   SPD2_NEPCL              Reviewed;         627 AA.
AC   P46804;
DT   01-NOV-1995, integrated into UniProtKB/Swiss-Prot.
DT   01-NOV-1995, sequence version 1.
DT   11-SEP-2007, entry version 38.
DE   Spidroin-2 (Dragline silk fibroin 2) (Fragment).
OS   Nephila clavipes (Golden silk orbweaver).
OC   Eukaryota; Metazoa; Arthropoda; Chelicerata; Arachnida; Araneae;
OC   Araneomorphae; Entelegynae; Araneoidea; Tetragnathidae; Nephila.
OX   NCBI_TaxID=6915;
RN   [1]
RP   NUCLEOTIDE SEQUENCE [MRNA].
RX   MEDLINE=92406876; PubMed=1527052;
RA   Hinman M.B., Lewis R.V.;
RT   "Isolation of a clone encoding a second dragline silk fibroin. Nephila
RT   clavipes dragline silk is a two-protein fiber.";
RL   J. Biol. Chem. 267:19320-19324(1992).
CC   -!- FUNCTION: Spiders major ampullate silk possesses unique
CC       characteristics of strength and elasticity. Fibroin consists of
CC       pseudocrystalline regions of antiparallel beta-sheet interspersed
CC       with elastic amorphous segments.
CC   -!- SUBUNIT: Major subunit, with spidroin 1, of the dragline silk.
CC   -!- SUBCELLULAR LOCATION: Secreted, extracellular space.
CC   -!- DOMAIN: Highly repetitive protein characterized by regions of
CC       polyalanine and glycine-rich repeating units.
CC   -!- SIMILARITY: Belongs to the silk fibroin family.
CC   -!- WEB RESOURCE: NAME=Protein Spotlight; NOTE=The tiptoe of an airbus
CC       - Issue 24 of July 2002;
CC       URL="http://www.expasy.org/spotlight/back_issues/sptlt024.shtml".
CC   ---------------------------------------------------------------------
CC   Copyrighted by the UniProt Consortium, see http://www.uniprot.org/terms
CC   Distributed under the Creative Commons Attribution-NoDerivs License
CC   ---------------------------------------------------------------------
DR   EMBL; M92913; AAA29381.1; -; mRNA.
DR   PIR; A44112; A44112.
PE   2: Evidence at transcript level;
KW   Repeat; Secreted; Silk protein.
FT   CHAIN        <1    627       Spidroin-2.
FT                                 /FTId=PRO_0000221447.
FT   REPEAT        1     36       1.
FT   REPEAT       37     79       2.
FT   REPEAT       80    121       3.
FT   REPEAT      122    172       4.
FT   REPEAT      173    213       5.
FT   REPEAT      214    252       6.
FT   REPEAT      253    283       7.
FT   REPEAT      284    317       8.
FT   REPEAT      318    359       9.
FT   REPEAT      360    391       10.
FT   REPEAT      392    428       11.
FT   REPEAT      429    464       12.
FT   REPEAT      465    488       13.
FT   REPEAT      489    515       14.
FT   REPEAT      516    530       15.
FT   REGION        1    530       15 X approximate tandem repeats.
FT   NON_TER       1      1
SQ   SEQUENCE   627 AA;  54184 MW;  CB9B63779B2C594B CRC64;
     PGGYGPGQQG PGGYGPGQQG PSGPGSAAAA AAAAAAGPGG YGPGQQGPGG YGPGQQGPGR
     YGPGQQGPSG PGSAAAAAAG SGQQGPGGYG PRQQGPGGYG QGQQGPSGPG SAAAASAAAS
     AESGQQGPGG YGPGQQGPGG YGPGQQGPGG YGPGQQGPSG PGSAAAAAAA ASGPGQQGPG
     GYGPGQQGPG GYGPGQQGPS GPGSAAAAAA AASGPGQQGP GGYGPGQQGP GGYGPGQQGL
     SGPGSAAAAA AAGPGQQGPG GYGPGQQGPS GPGSAAAAAA AAAGPGGYGP GQQGPGGYGP
     GQQGPSGAGS AAAAAAAGPG QQGLGGYGPG QQGPGGYGPG QQGPGGYGPG SASAAAAAAG
     PGQQGPGGYG PGQQGPSGPG SASAAAAAAA AGPGGYGPGQ QGPGGYAPGQ QGPSGPGSAS
     AAAAAAAAGP GGYGPGQQGP GGYAPGQQGP SGPGSAAAAA AAAAGPGGYG PAQQGPSGPG
     IAASAASAGP GGYGPAQQGP AGYGPGSAVA ASAGAGSAGY GPGSQASAAA SRLASPDSGA
     RVASAVSNLV SSGPTSSAAL SSVISNAVSQ IGASNPGLSG CDVLIQALLE IVSACVTILS
     SSSIGQVNYG AASQFAQVVG QSVLSAF
//
```

Perl 5 Reference Guide (Extract)

Literals

Numeric:
- **123**
- **1_234**
- **123.4**
- **5E-10**
- **0xff** (hex)
- **0377** (octal)

String:
- **'abc'**
  literal string, no variable interpolation or escape characters, except **\'** and ****. Also: **q/abc/**. Almost any pair of delimiters can be used instead of **/.../**.
- **"abc"**
  Variables are interpolated and escape sequences are processed. Also: **qq/abc/**.
  Escape sequences: **\t** (Tab), **\n** (Newline), **\r** (Return), **\f** (Formfeed), **\b** (Backspace), **\a** (Alarm), **\e** (Escape), **\033** (octal), **\x1b** (hex), **\c[** (control)
  \l and **\u** lowercase/uppercase the following character. **\L** and **\U** lowercase/uppercase until a **\E** is encountered. **\Q** quote regular expression characters until a **\E** is encountered.
- **'**COMMAND**'**
  evaluates to the output of the COMMAND. Also: **qx/**COMMAND**/**.

Array:
- **(1, 2, 3)**. **()** is an empty array.
- **(1..4)** is the same as **(1,2,3,4)**,
  likewise **('a'..'z')**.
- **qw/foo bar .../** is the same as **('foo','bar',...)**.

Array reference:
- **[1,2,3]**

Hash (associative array):
- **(**KEY1**,** VAL1**,** KEY2**,** VAL2**,...)**
  Also **(**KEY1**=>** VAL1**,** KEY2**=>** VAL2**,...)**

Hash reference:
- **{**KEY1**,** VAL1**,** KEY2**,** VAL2**,...}**

Code reference:
- **sub {**STATEMENTS**}**

Filehandles:
- **<STDIN>**, **<STDOUT>**, **<STDERR>**, **<ARGV>**, **<DATA>**.
  User-specified: HANDLE, **$**VAR.

Globs:
- **<**PATTERN**>** evaluates to all filenames according to the pattern. Use **<${**VAR**}>** or **glob $**VAR to glob from a variable.

Here-Is:
- **<<**IDENTIFIER
  Shell-style "here document."

Special tokens:
- **__FILE__**: filename; **__LINE__**: line number; **__END__**: end of program; remaining lines can be read using the filehandle **DATA**.

Variables

$var	a simple scalar variable.
$var[28]	29th element of array **@var**.
$p = \@var	now **$p** is a reference to array **@var**.
$$p[28]	29th element of array referenced by **$p**. Also, **$p->[28]**.
$var[-1]	last element of array **@var**.
$var[$i][$j]	**$j**th element of the **$i**th element of array **@var**.
$var{'Feb'}	one value from hash (associative array) **%var**.
$p = \%var	now **$p** is a reference to hash **%var**.

$$p{'Feb'}	a value from hash referenced by **$p**. Also, **$p->{'Feb'}**.
$#var	last index of array **@var**.
@var	the entire array; in a scalar context, the number of elements in the array.
@var[3,4,5]	a slice of array **@var**.
@var{'a','b'}	a slice of **%var**; same as **($var{'a'},$var{'b'})**.
%var	the entire hash; in a scalar context, **true** if the hash has elements.
$var{'a',1,...}	emulates a multidimensional array.
('a'...'z')[4,7,9]	a slice of an array literal.
PKG**::**VAR	a variable from a package, e.g., **$pkg::var**, **@pkg::ary**.
\OBJECT	reference to an object, e.g., **\$var**, **\%hash**.
***NAME**	refers to all objects represented by NAME. ***n1 = *n2** makes **n1** an alias for **n2**. ***n1 = $n2** makes **$n1** an alias for **$n2**.

You can always use a **{**BLOCK**}** returning the right type of reference instead of the variable identifier, e.g., **${...}**, **&{...}**. **$$p** is just a shorthand for **${$p}**.

Operators

******	Exponentiation
+ - * /	Addition, subtraction, multiplication, division
%	Modulo division
& | ^	Bitwise AND, bitwise OR, bitwise exclusive OR
>> <<	Bitwise shift right, bitwise shift left
|| &&	Logical OR, logical AND
.	Concatenation of two strings
x	Returns a string or array consisting of the left operand (an array or a string) repeated the number of times specified by the right operand

All of the above operators also have an assignment operator, e.g., **.=**

->	Dereference operator
****	Reference (unary)
! ~	Negation (unary), bitwise complement (unary)
++ --	Auto-increment (magical on strings), auto-decrement
== !=	Numeric equality, inequality
eq ne	String equality, inequality
< >	Numeric less than, greater than
lt gt	String less than, greater than
<= >=	Numeric less (greater) than or equal to
le ge	String less (greater) than or equal to
<=> cmp	Numeric (string) compare. Returns -1, 0, 1.
=~ !~	Search pattern, substitution, or translation (negated)
..	Range (scalar context) or enumeration (array context)
?:	Alternation (if-then-else) operator
,	Comma operator, also list element separator. You can also use **=>**.
not	Low-precedence negation
and	Low-precedence AND
or xor	Low-precedence OR, exclusive OR

All Perl functions can be used as list operators, in which case they have very high or very low precedence, depending on whether you look at the left or the right side of the operator. Only the operators **not**, **and**, **or** and **xor** have lower precedence.

A "list" is a list of expressions, variables, or lists. An array variable or an array slice may always be used instead of a list.

Parentheses can be added around the parameter lists to avoid precedence problems.

Statements

Every statement is an expression, optionally followed by a modifier, and terminated by a semicolon. The

semicolon may be omitted if the statement is the final one in a BLOCK.

Execution of expressions can depend on other expressions using one of the modifiers **if**, **unless**, **while** or **until**, for example:

```
EXPR1 if EXPR2 ;
EXPR1 until EXPR2 ;
```

The logical operators **||**, **&&** or **?:** also allow conditional execution:

```
EXPR1 || EXPR2 ;
EXPR1 ? EXPR2 : EXPR3 ;
```

Statements can be combined to form a BLOCK when enclosed in **{}**. Blocks may be used to control flow:

```
if (EXPR) BLOCK [ [ elsif (EXPR) BLOCK ... ] else BLOCK ]
unless (EXPR) BLOCK [ else BLOCK ]
[ LABEL: ] while (EXPR) BLOCK [ continue BLOCK ]
[ LABEL: ] until (EXPR) BLOCK [ continue BLOCK ]
[ LABEL: ] for (EXPR; EXPR; EXPR) BLOCK
[ LABEL: ] foreach VAR² (LIST) BLOCK
[ LABEL: ] BLOCK [ continue BLOCK ]
```

Program flow can be controlled with:

goto LABEL	Continue execution at the specified label.
last [LABEL]	Immediately exits the loop in question. Skips continue block.
next [LABEL]	Starts the next iteration of the loop.
redo [LABEL]	Restarts the loop block without evaluating the conditional again.

Special forms are:

```
do BLOCK while EXPR ;
do BLOCK until EXPR ;
```

which are guaranteed to perform BLOCK once before testing EXPR, and

```
do BLOCK
```

which effectively turns BLOCK into an expression.

Structure Conversion

pack TEMPLATE, LIST
: Packs the values into a binary structure using TEMPLATE.

unpack TEMPLATE, EXPR
: Unpacks the structure EXPR into an array, using TEMPLATE.

TEMPLATE is a sequence of characters as follows:

a / **A**	ASCII string, null- / space-padded
b / **B**	Bit string in ascending / descending order
c / **C**	Native / unsigned char value
f / **d**	Single / double float in native format
h / **H**	Hex string, low / high nybble first
i / **I**	Signed / unsigned integer value
l / **L**	Signed / unsigned long value
n / **N**	Short / long in network (big endian) byte order
s / **S**	Signed / unsigned short value
u / **P**	Uuencoded string / pointer to a string
v / **V**	Short / long in VAX (little endian) byte order
x / **@**	Null byte / null fill until position
X	Backup a byte

Each character may be followed by a decimal number that will be used as a repeat count; an asterisk (*****) specifies

all remaining arguments. If the format is preceded with **%**N, **unpack** returns an N-bit checksum instead. Spaces may be included in the template for readability purposes.

String Functions

chomp LIST²
: Removes line endings from all elements of the list; returns the (total) number of characters removed.

chop LIST²
: Chops off the last character on all elements of the list; returns the last chopped character.

crypt PLAINTEXT, SALT
: Encrypts a string.

eval EXPR²
: EXPR is parsed and executed as if it were a Perl program. The value returned is the value of the last expression evaluated. If there is a syntax error or runtime error, an undefined string is returned by **eval**, and **$@** is set to the error message. See also **eval** in section Miscellaneous.

index STR, SUBSTR [, OFFSET]
: Returns the position of SUBSTR in STR at or after OFFSET. If the substring is not found, returns **-1** (but see **$[** in section Special Variables).

length EXPR²
: Returns the length in characters of the value of EXPR.

lc EXPR
: Returns a lowercase version of EXPR.

lcfirst EXPR
: Returns EXPR with the first character lowercase.

quotemeta EXPR
: Returns EXPR with all regular expression metacharacters quoted.

rindex STR, SUBSTR [, OFFSET]
: Returns the position of the last SUBSTR in STR at or before OFFSET.

substr EXPR, OFFSET [, LEN]
: Extracts a substring of length LEN out of EXPR and returns it. If OFFSET is negative, counts from the end of the string. May be assigned to.

uc EXPR
: Returns an uppercased version of EXPR.

ucfirst EXPR
: Returns EXPR with the first character uppercased.

Array and List Functions

delete $HASH**{**KEY**}**
: Deletes the specified value from the specified hash. Returns the deleted value unless HASH is **tie**d to a package that does not support this.

each %HASH
: Returns a 2-element array consisting of the key and value for the next value of the hash. Entries are returned in an apparently random order. After all values of the hash have been returned, a null array is returned. The next call to **each** after that will start iterating again.

exists EXPR²
: Checks if the specified hash key exists in its hash array.

grep EXPR, LIST

grep BLOCK LIST
: Evaluates EXPR or BLOCK for each element of the LIST, locally setting **$_** to refer to the element. Modifying **$_** will modify the corresponding element from LIST. Returns the array of elements from LIST for which EXPR returned **true**.

join EXPR, LIST
: Joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string.

keys %HASH
: Returns an array with all of the keys of the named hash.

map EXPR, LIST

map BLOCK LIST
: Evaluates EXPR or BLOCK for each element of the LIST, locally setting **$_** to refer to the element. Modifying **$_** will modify the corresponding element from LIST. Returns the list of results.

pop @ARRAY
: Pops off and returns the last value of the array.

push @ARRAY, LIST

Pushes the values of LIST onto the end of the array.

reverse LIST
: In array context, returns the LIST in reverse order. In scalar context: returns the first element of LIST with bytes reversed.

scalar @ARRAY
: Returns the number of elements in the array.

scalar %HASH
: Returns a **true** value if the hash has elements defined.

shift [@ARRAY]
: Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If @ARRAY is omitted, shifts **@ARGV** in main and **@_** in subroutines.

sort [SUBROUTINE] LIST
: Sorts the LIST and returns the sorted array value. SUBROUTINE, if specified, must return less than zero, zero, or greater than zero, depending on how the elements of the array (available to the routine as **$a** and **$b**) are to be ordered. SUBROUTINE may be the name of a user-defined routine, or a BLOCK.

splice @ARRAY, OFFSET [, LENGTH [, LIST]]
: Removes the elements of @ARRAY designated by OFFSET and LENGTH, and replaces them with LIST (if specified). Returns the elements removed.

split [PATTERN [, EXPR[2] [, LIMIT]]]
: Splits a string into an array of strings, and returns it. If LIMIT is specified, splits into at most that number of fields. If PATTERN is also omitted, splits at the whitespace. If not in array context, returns number of fields and splits to @_. See also Search and Replace Functions.

unshift @ARRAY, LIST
: Prepends list to the front of the array, and returns the number of elements in the new array.

values %HASH
: Returns a normal array consisting of all the values of the named hash.

Regular Expressions

Each character matches itself, unless it is one of the special characters **+ ? . * ^ $ ( ) [ ] { } |** \. The special meaning of these characters can be escaped using a \.

.	matches an arbitrary character, but not a newline unless it is a single-line match (see **m//s**).
(...**)**	groups a series of pattern elements to a single element.
^	matches the beginning of the target. In multiline mode (see **m//m**) also matches after every newline character.
$	matches the end of the line. In multiline mode also matches before every newline character.
[... **]**	denotes a class of characters to match. **[^** ... **]** negates the class.
(... **|** ... **|** ... **)**	matches one of the alternatives.
(?# TEXT **)**	Comment.
(?: REGEXP **)**	Like **(**REGEXP**)** but does not make back-references.
(?= REGEXP **)**	Zero width positive look-ahead assertion.
(?! REGEXP **)**	Zero width negative look-ahead assertion.
(? MODIFIER **)**	Embedded pattern-match modifier. MODIFIER can be one or more of **i**, **m**, **s**, or **x**.

Quantified subpatterns match as many times as possible. When followed with a **?** they match the minimum number of times. These are the quantifiers:

+	matches the preceding pattern element one or more times.
?	matches zero or one times.
*****	matches zero or more times.
{N**,**M**}**	denotes the minimum N and maximum M match count. **{**N**}** means exactly N times; **{**N**,}** means at least N times.

A \ escapes any special meaning of the following character if non-alphanumeric, but it turns most alphanumeric characters into something special:

\w	matches alphanumeric, including _, **\W** matches non-alphanumeric.
\s	matches whitespace, **\S** matches non-whitespace.
\d	matches numeric, **\D** matches non-numeric.
\A	matches the beginning of the string, **\Z** matches the end.
\b	matches word boundaries, **\B** matches non-boundaries.
\G	matches where the previous **m//g** search left off.
\n, **\r**, **\f**, **\t**	etc. have their usual meaning.

\w, **\s** and **\d** may be used within character classes, **\b** denotes backspace in this context.

Back-references:

\1 ... **\9**	refer to matched subexpressions, grouped with **()**, inside the match.
\10	and up can also be used if the pattern matches that many subexpressions.

See also **$1** ... **$9**, **$+**, **$&**, **$`**, and **$'** in section Special Variables.

With modifier **x**, whitespace can be used in the patterns for readability purposes.

Search and Replace Functions

[EXPR **=~**] [**m**] **/**PATTERN**/** [**g**] [**i**] [**m**] [**o**] [**s**] [**x**]
: Searches EXPR (default: **$_**) for a pattern. If you prepend an **m** you can use almost any pair of delimiters instead of the slashes. If used in array context, an array is returned consisting of the subexpressions matched by the parentheses in the pattern, i.e., **($1,$2,$3,...)**.
Optional modifiers: **g** matches as many times as possible; **i** searches in a case-insensitive manner; **o** interpolates variables only once. **m** treats the string as multiple lines; **s** treats the string as a single line; **x** allows for regular expression extensions.
If PATTERN is empty, the most recent pattern from a previous match or replacement is used.
With **g** the match can be used as an iterator in scalar context.

?PATTERN**?**
: This is just like the **/**PATTERN**/** search, except that it matches only once between calls to the **reset** operator.

[**$**VAR **=~**] **s/**PATTERN**/**REPLACEMENT**/** [**e**] [**g**] [**i**] [**m**] [**o**] [**s**] [**x**]
: Searches a string for a pattern, and if found, replaces that pattern with the replacement text. It returns the number of substitutions made, if any; if no substitutions are made, it returns **false**.
Optional modifiers: **g** replaces all occurrences of the pattern; **e** evaluates the replacement string as a Perl expression; for any other modifiers, see **/**PATTERN**/** matching. Almost any delimiter may replace the slashes; if single quotes are used, no interpretation is done on the strings between the delimiters, otherwise the strings are interpolated as if inside double quotes.
If bracketing delimiters are used, PATTERN and REPLACEMENT may have their own delimiters, e.g., **s(foo)[bar]**. If PATTERN is empty, the most recent pattern from a previous match or replacement is used.

[**$**VAR **=~**] **tr/**SEARCHLIST**/**REPLACEMENTLIST**/** [**c**] [**d**] [**s**]
: Translates all occurrences of the characters found in the search list with the corresponding character in the replacement list. It returns the number of characters replaced. **y** may be used instead of **tr**.
Optional modifiers: **c** complements the SEARCHLIST; **d** deletes all characters found in SEARCHLIST that do not have a corresponding character in REPLACEMENTLIST; **s** squeezes all sequences of characters that are translated into the same target character into one occurrence of this character.

pos SCALAR
: Returns the position where the last **m//g** search left off for SCALAR. May be assigned to.

study [**$**VAR[2]]
: Study the scalar variable **$**VAR in anticipation of performing many pattern matches on its contents before the variable is next modified.

Input / Output

In input/output operations, FILEHANDLE may be a filehandle as opened by the **open** operator, a predefined filehandle (e.g., **STDOUT**) or a scalar variable that evaluates to the name of a filehandle to be used.

<FILEHANDLE**>**
: In scalar context, reads a single line from the file opened on FILEHANDLE. In array context, reads the whole file.

< >
: Reads from the input stream formed by the files specified in **@ARGV**, or standard input if no arguments were supplied.

close FILEHANDLE
: Closes the file or pipe associated with the filehandle.

open FILEHANDLE [, FILENAME]
: Opens a file and associates it with FILEHANDLE. If FILENAME is omitted, the scalar variable of the same name as the FILEHANDLE must contain the filename.
The following filename conventions apply when opening a file.

"FILE" open FILE for input. Also "<FILE".
">FILE" open FILE for output, creating it if necessary.
">>FILE" open FILE in append mode.
"+<FILE" open FILE with read/write access (file must exist).
"+>FILE" open FILE with read/write access (file truncated).
"|CMD" opens a pipe to command CMD; forks if CMD is -.
"CMD|" opens a pipe from command CMD; forks if CMD is -.
FILE may be &FILEHND in which case the new filehandle is connected to the (previously opened) filehandle FILEHND. If it is &=N, FILE will be connected to the given file descriptor. **open** returns **undef** upon failure, **true** otherwise.
Returns a pair of connected pipes.

print [FILEHANDLE] [LIST[2]]
Equivalent to **print** FILEHANDLE **sprintf** LIST.

Special Variables

The following variables are global and should be localized in subroutines:

$_	The default input and pattern-searching space.
$.	The current input line number of the last filehandle that was read.
$/	The input record separator, newline by default. May be multicharacter.
$,	The output field separator for the print operator.
$"	The separator that joins elements of arrays interpolated in strings.
$\	The output record separator for the print operator.
$#	The output format for printed numbers. Deprecated.
$*	Set to 1 to do multiline matching within strings. Deprecated, see the **m** and **s** modifiers in section Search and Replace Functions.
$?	The status returned by the last `...` command, pipe **close** or **system** operator.
$]	The perl version number, e.g., 5.001.
$[	The index of the first element in an array, and of the first character in a substring. Default is 0. Deprecated.
$;	The subscript separator for multidimensional array emulation. Default is "\034".
$!	If used in a numeric context, yields the current value of **errno**. If used in a string context, yields the corresponding error string.
$@	The Perl error message from the last **eval** or **do** EXPR command.
$:	The set of characters after which a string may be broken to fill continuation fields (starting with ^) in a format.
$0	The name of the file containing the Perl script being executed. May be assigned to.
$$	The process ID of the currently executing Perl program. Altered (in the child process) by **fork**.
$<	The real user ID of this process.
$>	The effective user ID of this process.
$(	The real group ID of this process.
$)	The effective group ID of this process.
$^A	The accumulator for **formline** and **write** operations.
$^D	The debug flags as passed to perl using -D.
$^F	The highest system file descriptor, ordinarily 2.
$^I	In-place edit extension as passed to Perl using -i.
$^L	Formfeed character used in formats.
$^P	Internal debugging flag.
$^T	The time (as delivered by **time**) when the program started. This value is used by the file test operators -M, -A and -C.
$^W	The value of the -w option as passed to Perl.
$^X	The name by which the currently executing program was invoked.

The following variables are context dependent and need not be localized:

$%	The current page number of the currently selected output channel.
$=	The page length of the current output channel. Default is 60 lines.
$-	The number of lines remaining on the page.
$~	The name of the current report format.
$^	The name of the current top-of-page format.
$|	If set to nonzero, forces a flush after every write or print on the currently selected output channel. Default is 0.
$ARGV	The name of the current file when reading from <>.

The following variables are always local to the current block:

$&	The string matched by the last successful pattern match.
$`	The string preceding what was matched by the last successful match.
$'	The string following what was matched by the last successful match.
$+	The last bracket matched by the last search pattern.
$1...$9...	Contain the subpatterns from the corresponding sets of parentheses in the last pattern successfully matched. $10... and up are only available if the match contained that many subpatterns.

Special Arrays

@ARGV	Contains the command-line arguments for the script (not including the command name).
@EXPORT	Names the methods a package exports by default.
@EXPORT_OK	Names the methods a package can export upon explicit request.
@INC	Contains the list of places to look for Perl scripts to be evaluated by the **do** FILENAME and **require** commands.
@ISA	List of base classes of a package.
@_	Parameter array for subroutines. Also used by **split** if not in array context.
%ENV	Contains the current environment.
%INC	List of files that have been included with **require** or **do**.
%OVERLOAD	Can be used to overload operators in a package.
%SIG	Used to set signal handlers for various signals.

Hidden Markov Models

Formula Collection

$$a_{kl} = P(\pi_i = l | \pi_{i-1} = k).$$

$$e_k(b) = P(x_i = b | \pi_i = k),$$

$$P(x,\pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}},$$

Algorithm: Viterbi

Initialisation ($i = 0$): $v_0(0) = 1$, $v_k(0) = 0$ for $k > 0$.

Recursion ($i = 1 \ldots L$): $v_l(i) = e_l(x_i) \max_k(v_k(i-1)a_{kl})$;
$\text{ptr}_i(l) = \text{argmax}_k(v_k(i-1)a_{kl})$.

Termination: $P(x,\pi^*) = \max_k(v_k(L)a_{k0})$;
$\pi_L^* = \text{argmax}_k(v_k(L)a_{k0})$.

Traceback ($i = L \ldots 1$): $\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$.

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k), \tag{3.10}$$

Algorithm: Forward algorithm

Initialisation ($i = 0$): $f_0(0) = 1$, $f_k(0) = 0$ for $k > 0$.

Recursion ($i = 1 \ldots L$): $f_l(i) = e_l(x_i) \sum_k f_k(i-1)a_{kl}$.

Termination: $P(x) = \sum_k f_k(L)a_{k0}$.

$$b_k(i) = P(x_{i+1} \ldots x_L | \pi_i = k). \tag{3.13}$$

Algorithm: Backward algorithm

Initialisation ($i = L$): $b_k(L) = a_{k0}$ for all k.

Recursion ($i = L-1, \ldots, 1$): $b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i+1)$.

Termination: $P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$.

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \quad \text{and} \quad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}. \tag{3.18}$$

$$P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{P(x)}, \tag{3.14}$$

$$P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{f_k(i) a_{kl} e_l(x_{i+1}) b_l(i+1)}{P(x)}. \tag{3.19}$$

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1), \tag{3.20}$$

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i | x_i^j = b\}} f_k^j(i) b_k^j(i), \tag{3.21}$$

Algorithm: Baum–Welch

Initialisation: Pick arbitrary model parameters.

Recurrence:

Set all the A and E variables to their pseudocount values r (or to zero).

For each sequence $j = 1 \ldots n$:

Calculate $f_k(i)$ for sequence j using the forward algorithm (p. 58).

Calculate $b_k(i)$ for sequence j using the backward algorithm (p. 59).

Add the contribution of sequence j to A (3.20) and E (3.21).

Calculate the new model parameters using (3.18).

Calculate the new log likelihood of the model.

Termination:

Stop if the change in log likelihood is less than some predefined threshold or the maximum number of iterations is exceeded. ◁

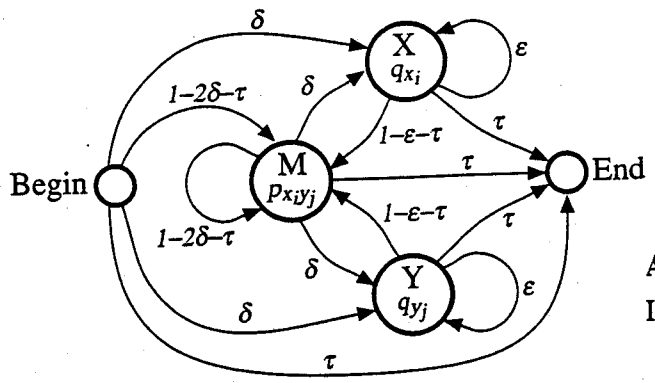
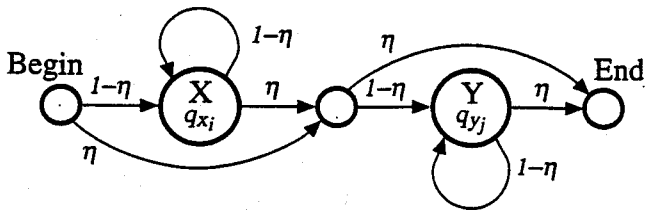


Figure 4.2 *The full probabilistic version of Figure 4.1.*



Algorithm: Forward calculation for pair HMMs

Initialisation:

$f^{\mathrm{M}}(0,0) = 1.\ f^{\mathrm{X}}(0,0) = f^{\mathrm{Y}}(0,0) = 0.$

All $f^{*}(i,-1), f^{*}(-1,j)$ are set to 0.

Recursion: $i = 0,\ldots,n, j = 0,\ldots,m$ except (0,0);

$$f^{\mathrm{M}}(i,j) = p_{x_i y_j}\left[(1-2\delta-\tau)f^{\mathrm{M}}(i-1,j-1)+(1-\varepsilon-\tau)(f^{\mathrm{X}}(i-1,j-1)+f^{\mathrm{Y}}(i-1,j-1))\right]$$

$$f^{\mathrm{X}}(i,j) = q_{x_i}\left[\delta f^{\mathrm{M}}(i-1,j)+\varepsilon f^{\mathrm{X}}(i-1,j)\right];$$

$$f^{\mathrm{Y}}(i,j) = q_{y_j}\left[\delta f^{\mathrm{M}}(i,j-1)+\varepsilon f^{\mathrm{Y}}(i,j-1)\right].$$

Termination:

$$f^{\mathrm{E}}(n,m) = \tau\left[f^{\mathrm{M}}(n,m)+f^{\mathrm{X}}(n,m)+f^{\mathrm{Y}}(n,m)\right].$$

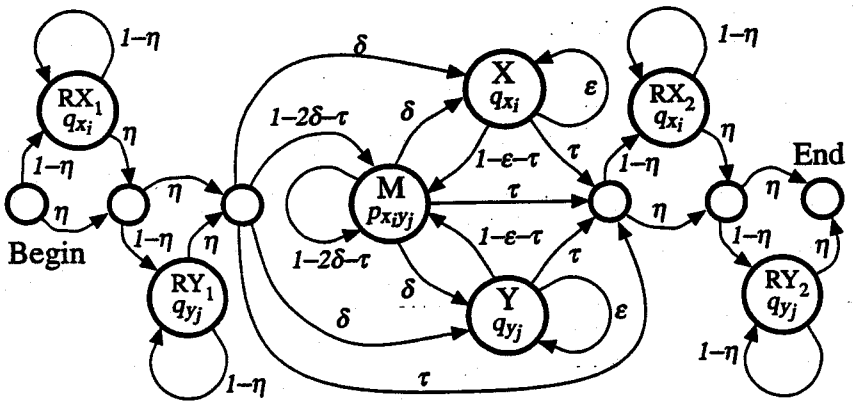


Figure 4.3 *A pair HMM for local alignment. This is composed of the global model (states* M, X *and* Y*) flanked by two copies of the random model (states* RX_1, RY_1 *and* RX_2, RY_2*).*

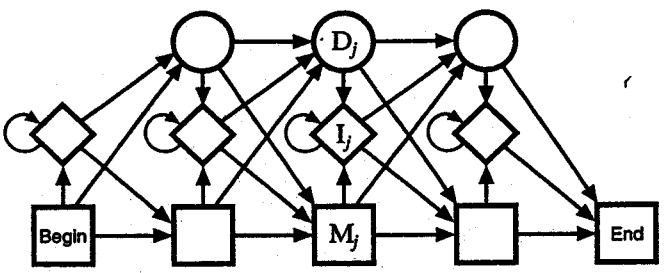


Figure 5.2 *The transition structure of a profile HMM. We use diamonds to indicate the insert states and and circles for the delete states.*