Genome 540 Discussion

Conor Camplisson

February 9th, 2023

Genome Sciences
UNIVERSITY OF WASHINGTON

Outline

- Homework 6 overview
- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation
- Homework 5 questions

Outline

- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

Homework 6 Overview

Science

Current Issue First release papers Archive About

HOME > SCIENCE > VOL. 376, NO. 6588 > THE COMPLETE SEQUENCE OF A HUMAN GENOME

SPECIAL ISSUE RESEARCH ARTICLE | HUMAN GENOMICS

The complete sequence of a human genome

SERGEY NURK, SERGEY KOREN, ARANG RHIE, MIKKO RAUTIAINEN, ANDREY V. BZIKADZE, ALLA MIKHEENKO, MITCHELL R. VOLLGER, NICOLAS ALTEMOSE, LEV URALSKY, [...], AND ADAM M. PHILLIPPY +90 authors Authors Info & Affiliations

First 'Telomere to Telomere' Human Genome Reveals Secrets of the Centromere

Researchers generate the first complete, gapless sequence of a human genome

Mitchell Vollger

GS 540 TA 2019

NIH National Human Genome Research Institute

Begin your search here

About Genomics Research Funding Research at NHGRI Hea

Telomere-to-Telomere (T2T)

The first complete, gapless sequence of a human genome.

Homework 6 Overview

chm13.chr16.txt

```
16        1         0
16        2         0
16        3         0
16        4         0
16        5         0
      [ ... ]
16        14793     0
16        14794     1
16        14795     3
16        14796     0
      [ ... ]
```

Goal: to find CNVs using D-segments

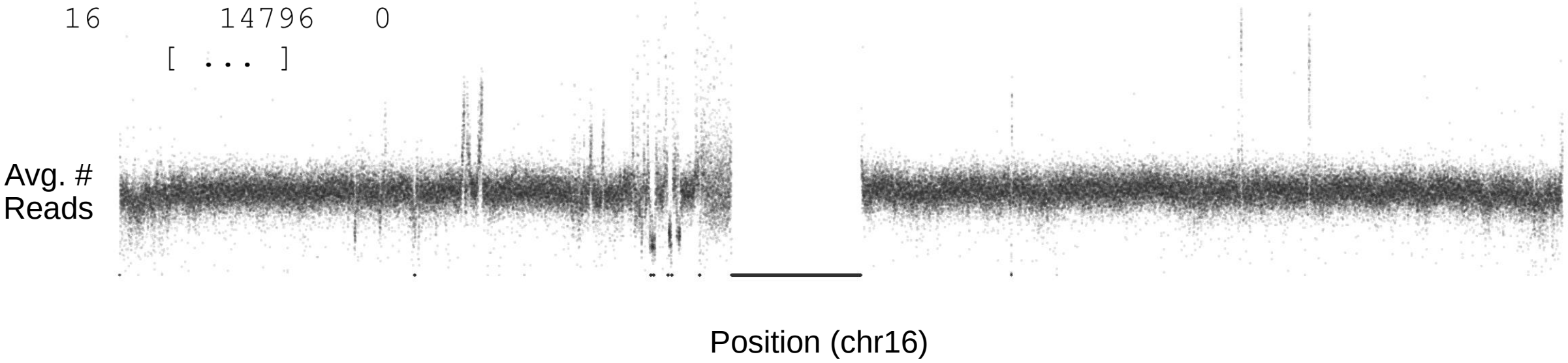
Data: next-gen read alignments to genome, CHM13 chr16

Observed symbols: counts of read starts at each position

- Frequencies from Poisson dist. with appropriate mean

Target regions: heterozygous duplications

- One chrom = ref allele, other = dup, Poisson mean 1.5X background

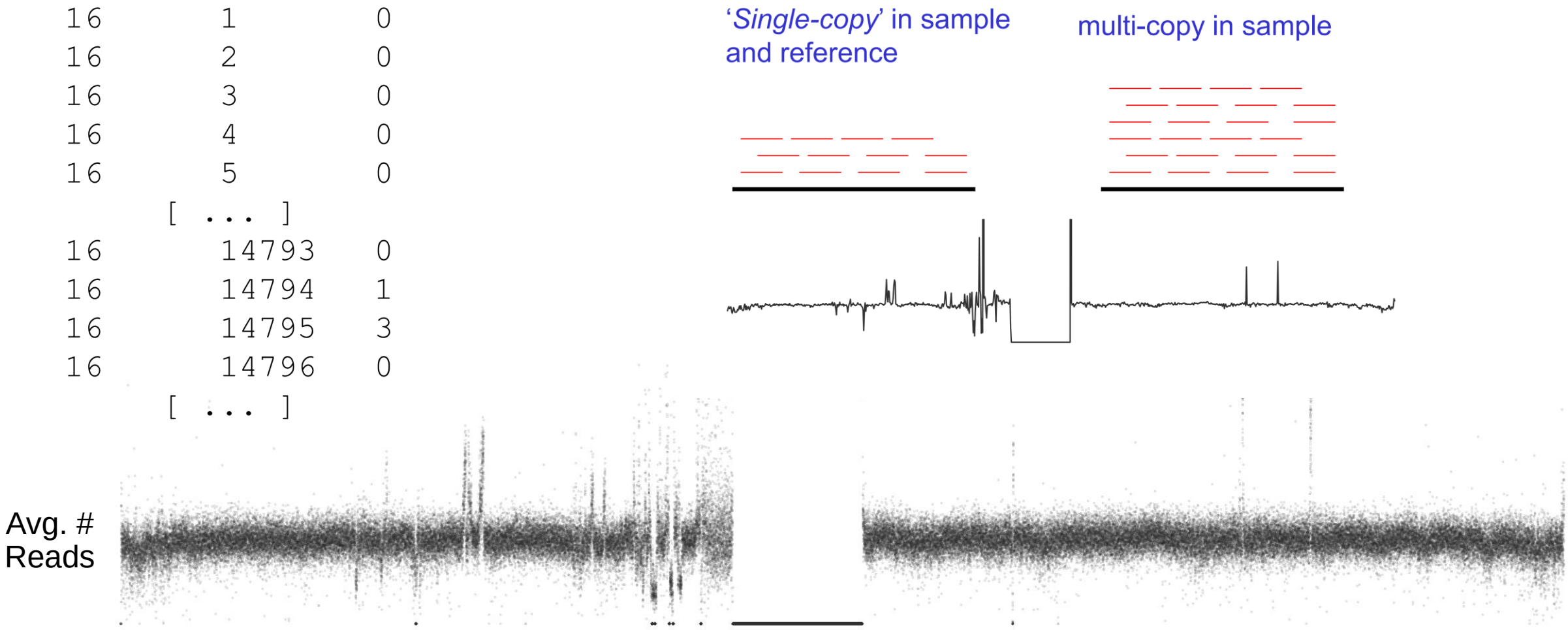


Homework 6 Overview

chm13.chr16.txt

```
16        1          0
16        2          0
16        3          0
16        4          0
16        5          0
      [  ...  ]
16        14793      0
16        14794      1
16        14795      3
16        14796      0
      [  ...  ]
```

Goal: to find CNVs using D-segments



Homework 6 Overview

chm13.chr16.txt

```
16      1       0
16      2       0
16      3       0
16      4       0
16      5       0
      [ ... ]
16      14793   0
16      14794   1
16      14795   3
16      14796   0
      [ ... ]
```

D-segment algorithm

$O(N)$ algorithm to find all maximal D-segs:

```
cumul = max = 0; start = 1;
for (i = 1; i ≤ N; i++) {
    cumul += s[i];
    if (cumul ≥ max)
        {max = cumul; end = i;}
    if (cumul ≤ 0 or cumul ≤ max + D or i == N) {
        if (max ≥ S)
            {print start, end, max; }
        max = cumul = 0; start = end = i + 1; /* NO BACKTRACKING
          NEEDED! */
    }
}
```

http://bozeman.mbt.washington.edu/compbio/mbt599_2022/Lec12.pdf

Homework 6 Overview

D-segment info

```
Assignment: GS 540 HW6
Name: Conor Camplisson
Email: concamp@uw.edu
Language: Python
Runtime: 1.058 sec

Segment Histogram:
Non-Elevated CN Segments=8
Elevated CN Segments=7

Segment List:
48164 48273 66.76
67646 68115 97.51
105528 106003 63.04
106904 107345 41.67
122792 123034 66.56
164376 164665 62.09
165086 166103 225.95
```

Annotations for top 3 scoring segments

```
Annotations:

Start: 165086
End: 166103
Description: Something interesting (e.g., "Overlaps with exon5 of the protein coding gene cMyc")

Start: 67646
End: 68115
Description: Something interesting (e.g., "Overlaps with exon5 of the protein coding gene cMyc")

Start: 48164
End: 48273
Description: Something interesting (e.g., "Overlaps with exon5 of the protein coding gene cMyc")
```

Read start count histograms

```
Read start histogram for non-elevated copy-number segments:
0=331908
1=19439
2=4272
>=3=1332

Read start histogram for elevated copy-number segments:
0=1656
1=542
2=352
>=3=499
```

Outline

- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

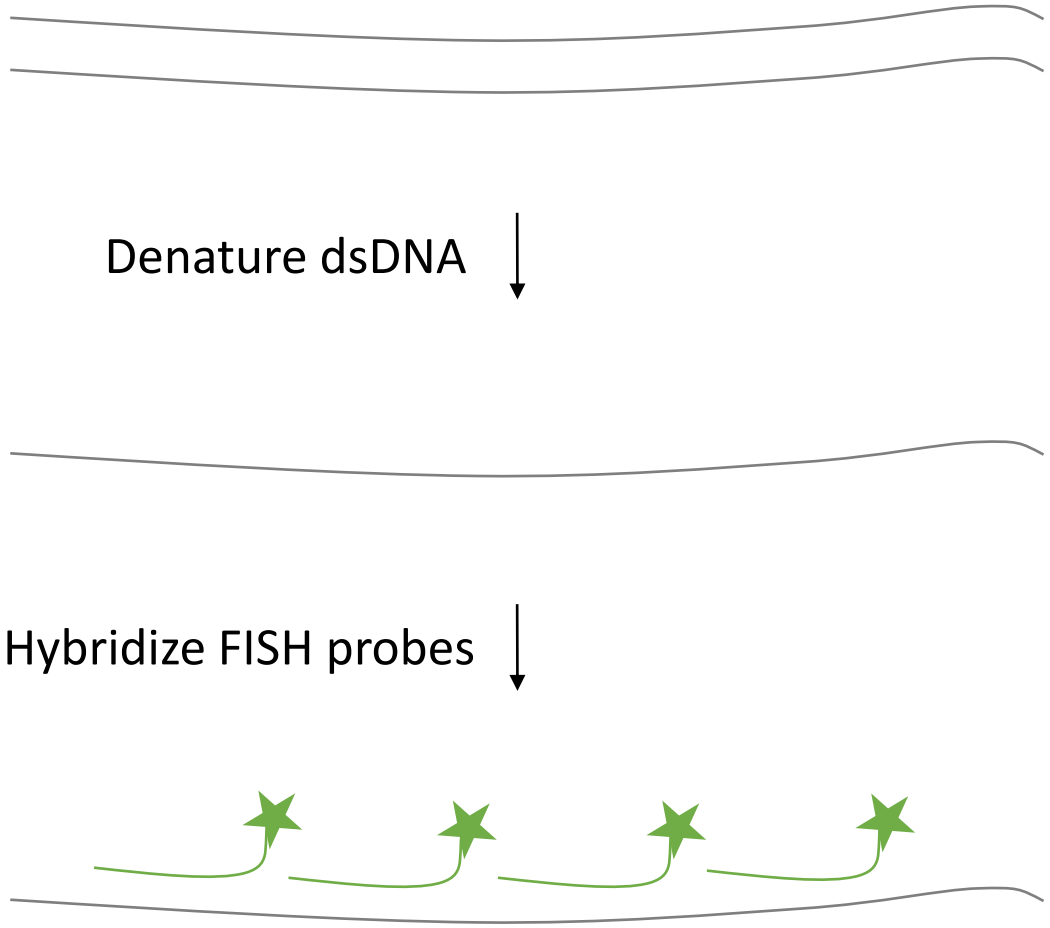
Outline

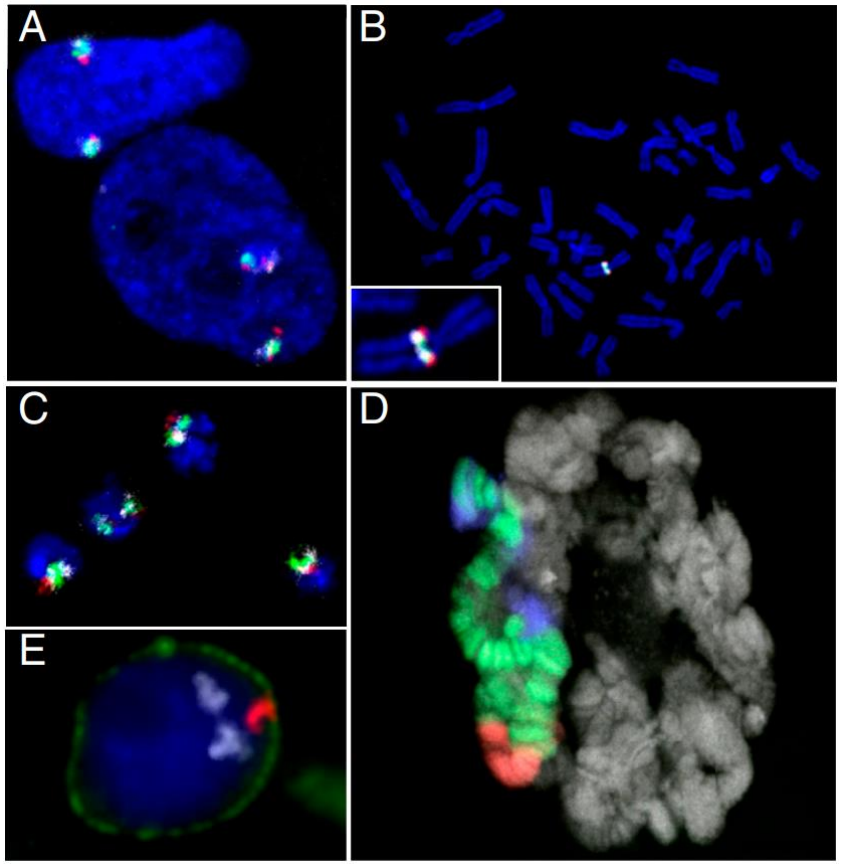
- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
    - GC-based linear model for DNA Tm
    - 8-bit integer encoding DNA sequences
    - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

Fluorescence *in situ* Hybridization (FISH)

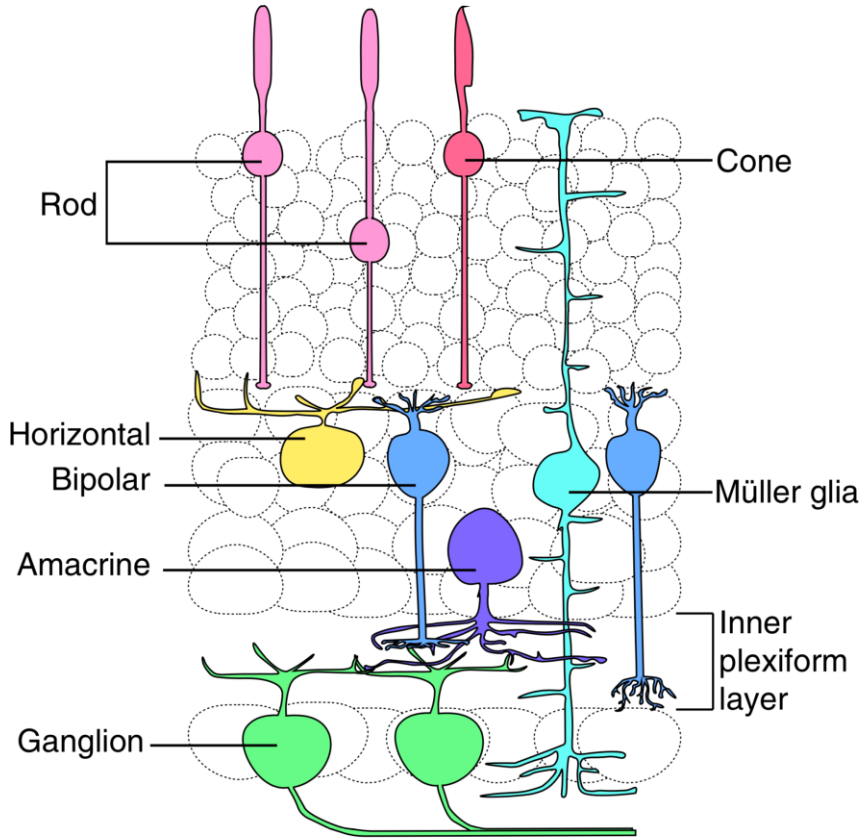


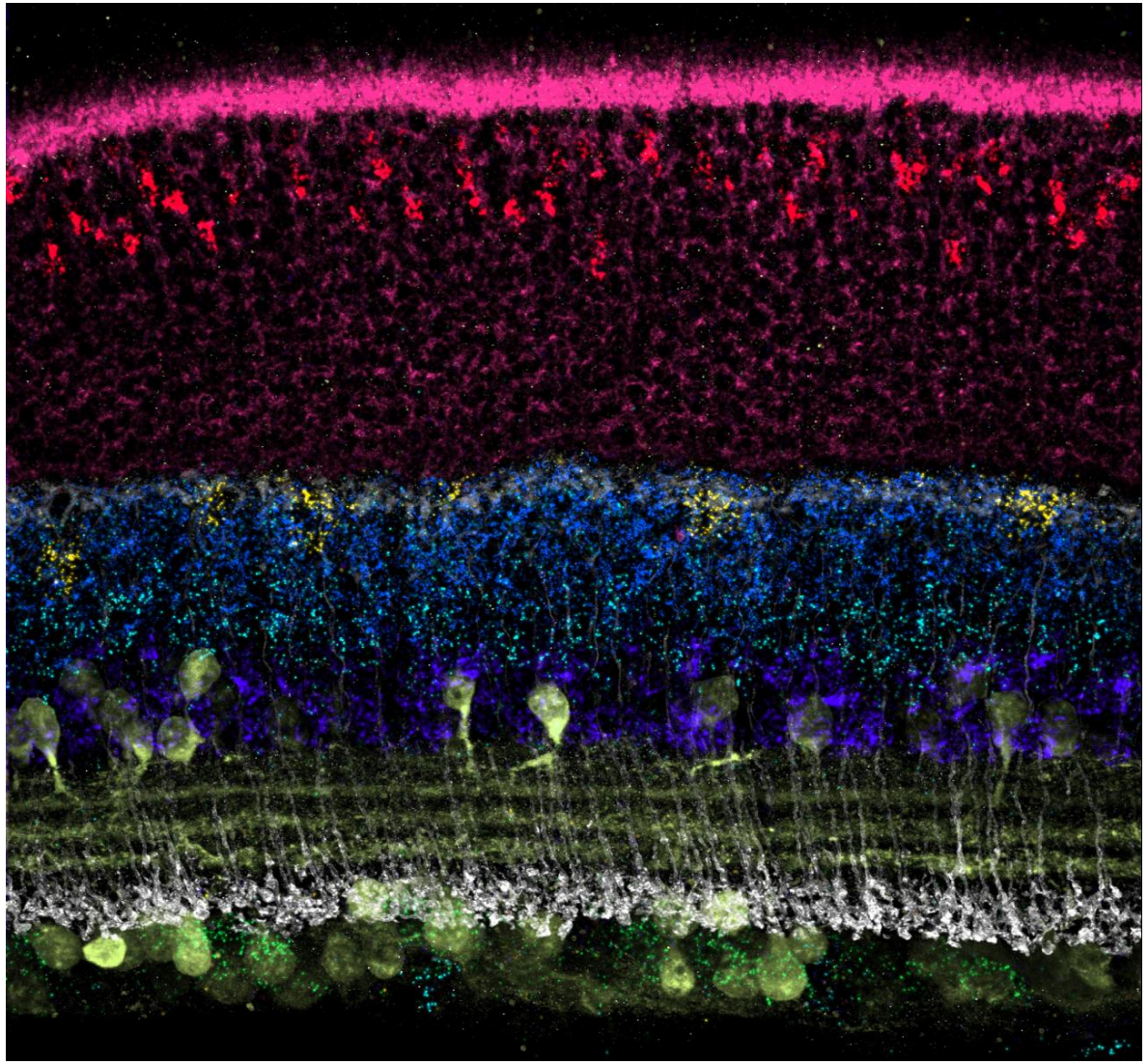


Beliveau, B.J. Et al. *PNAS* **109 (52)**, 21301-6 (2012)

11

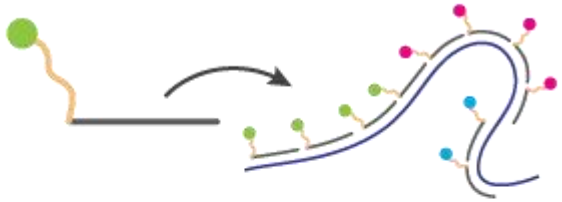
Targeting RNA & Proteins

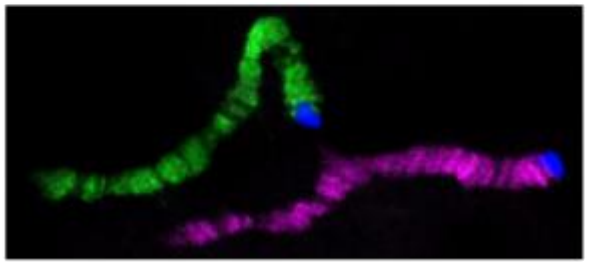




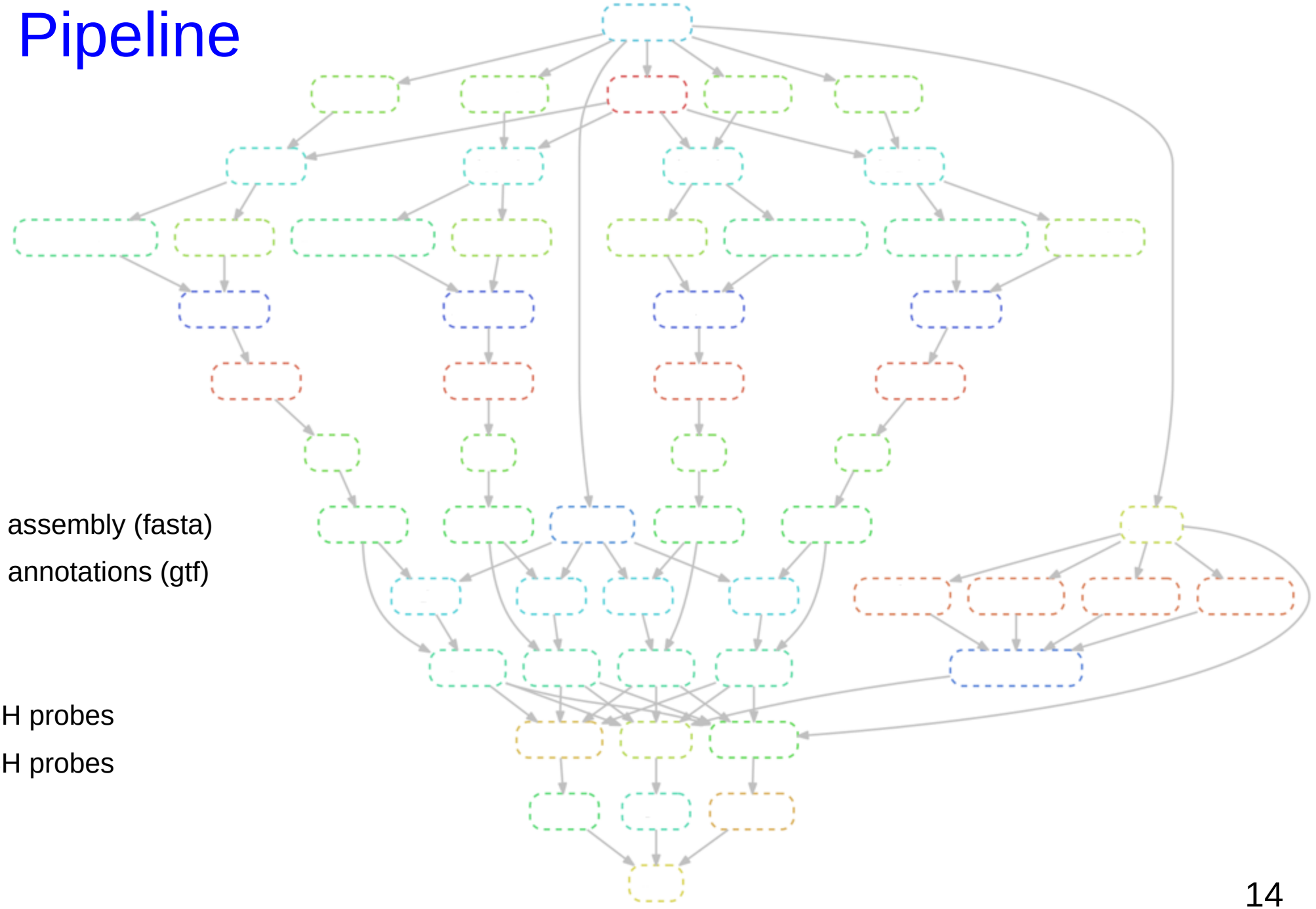
Kishi, J.Y., Lapan, S.W., Beliveau, B.J. et al. *Nat Methods* **16**, 533–544 (2019)

Where do FISH probes come from?





PaintSHOP Pipeline



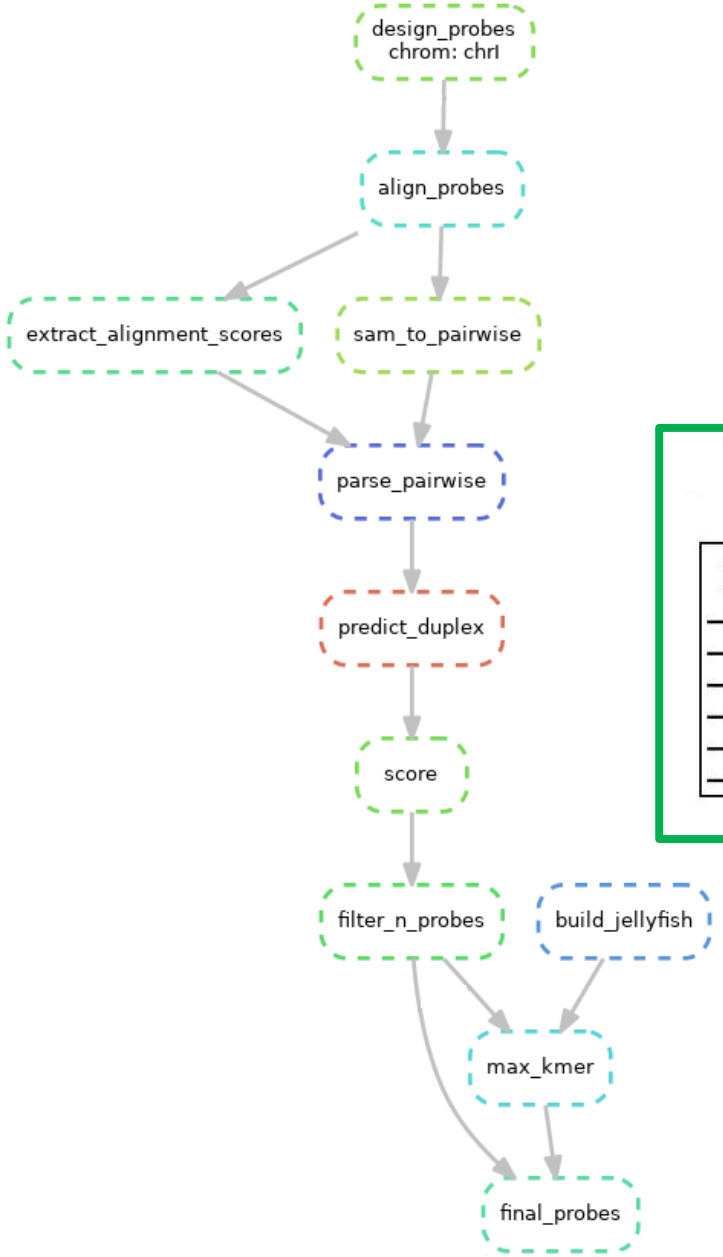
Input:

- Genome assembly (fasta)
- Genome annotations (gtf)

Output:

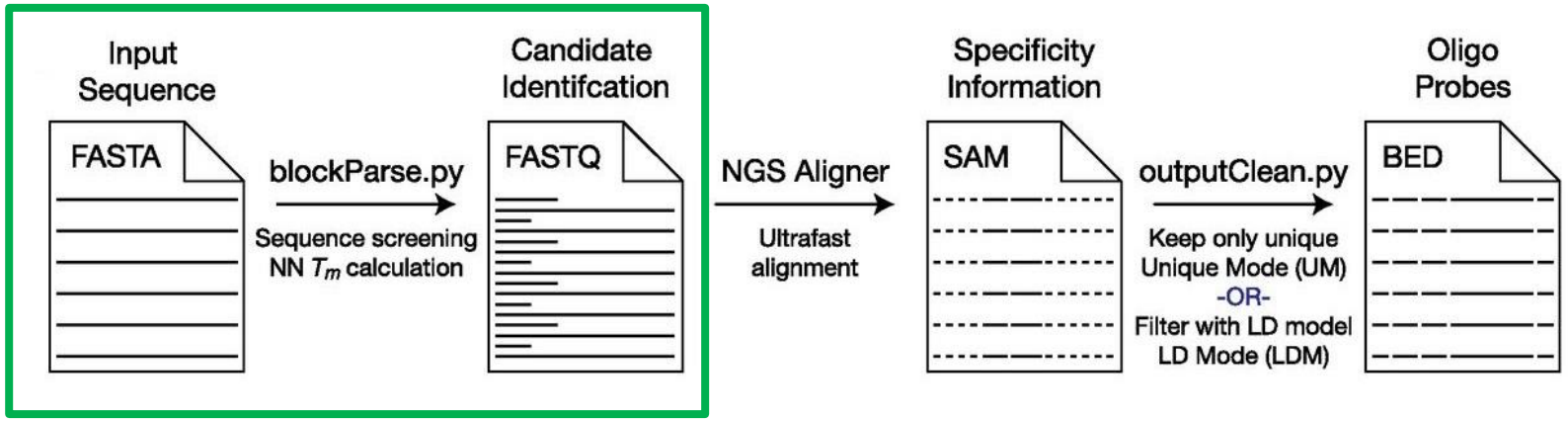
- DNA FISH probes
- RNA FISH probes

Design Probe Sequences



OligoMiner provides a rapid, flexible environment for the design of genome-scale oligonucleotide in situ hybridization probes

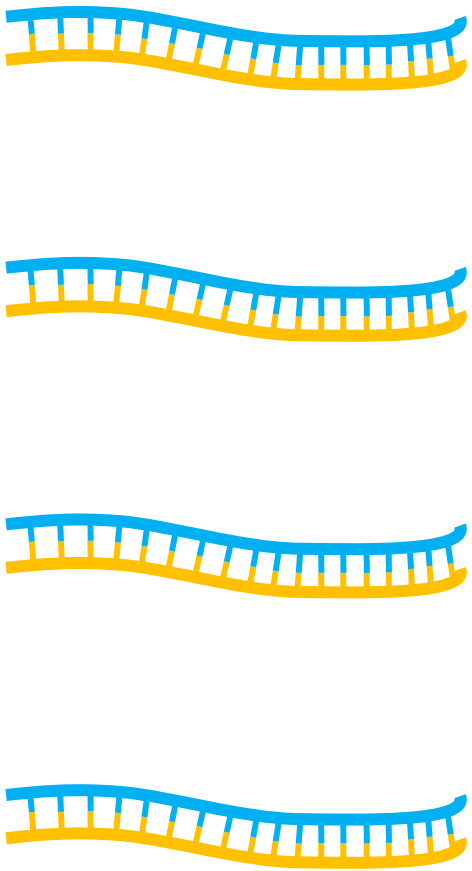
Brian J. Beliveau[a,b,1], Jocelyn Y. Kishi[a,b], Guy Nir[c], Hiroshi M. Sasaki[a,b], Sinem K. Saka[a,b], Son C. Nguyen[c,2], Chao-ting Wu[c], and Peng Yin[a,b,1]



Duplex probability (pdup)

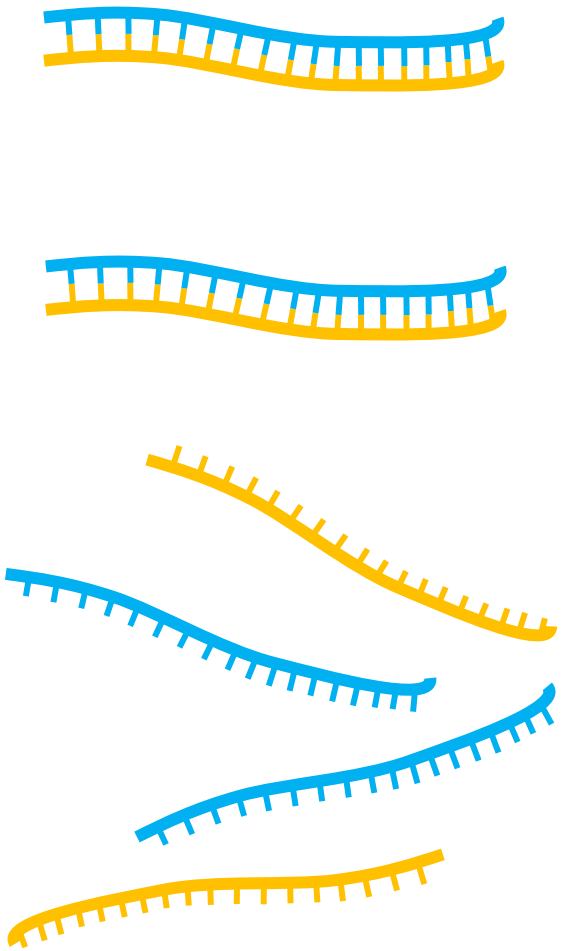
4 °C

pdup ~ 1.00



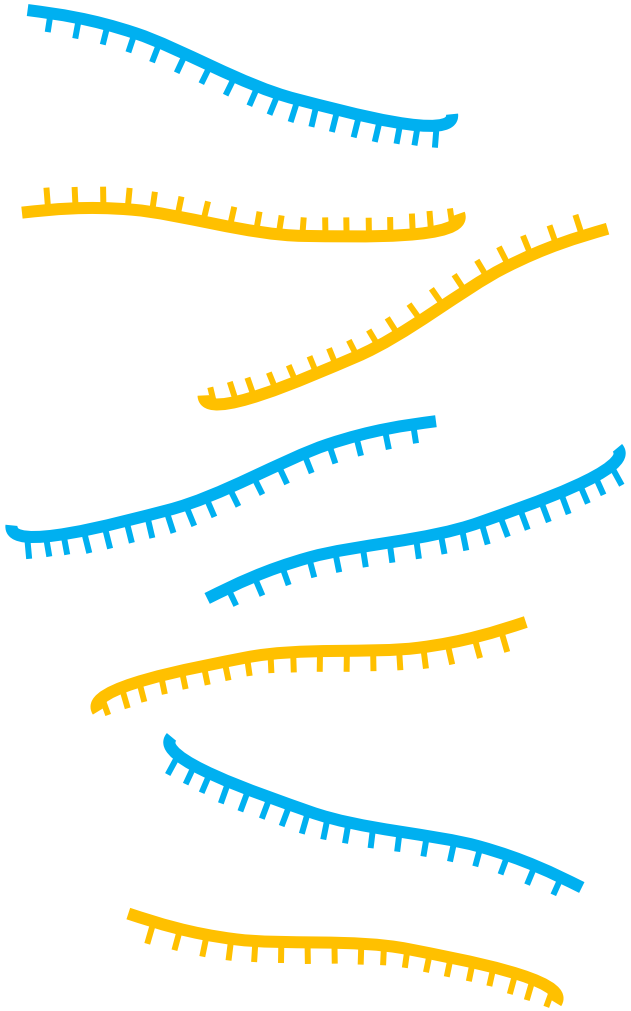
60 °C

pdup ~ 0.50



98 °C

pdup ~ 0.00



Tm = Temperature at which pdup == 0.5

Outline

- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

Tm linear model

Bio.SeqUtils.MeltingTemp.Tm_GC(*seq, check=True, strict=True, valueset=7, userset=None, Na=50, K=0, Tris=0, Mg=0, dNTPs=0, saltcorr=0, mismatch=True*)

Return the Tm using empirical formulas based on GC content.

General format: Tm = A + B(%GC) - C/N + salt correction - D(%mismatch)

A, B, C, D: empirical constants, N: primer length D (amount of decrease in Tm per % mismatch) is often 1, but sometimes other values have been used (0.6-1.5). Use 'X' to indicate the mismatch position in the sequence. Note that this mismatch correction is a rough estimate.

```
>>> from Bio.SeqUtils import MeltingTemp as mt
>>> print("%0.2f" % mt.Tm_GC('CTGCTGATXGCACGAGGTTATGG', valueset=2))
69.20
```

Arguments:

- valueset: A few often cited variants are included:

  1. Tm = 69.3 + 0.41(%GC) - 650/N (Marmur & Doty 1962, J Mol Biol 5: 109-118; Chester & Marshak 1993), Anal Biochem 209: 284-290)
  2. Tm = 81.5 + 0.41(%GC) - 675/N - %mismatch 'QuikChange' formula. Recommended (by the manufacturer) for the design of primers for QuikChange mutagenesis.
  3. Tm = 81.5 + 0.41(%GC) - 675/N + 16.6 x log[Na+] (Marmur & Doty 1962, J Mol Biol 5: 109-118; Schildkraut & Lifson 1965, Biopolymers 3: 195-208)
  4. Tm = 81.5 + 0.41(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). This is the standard formula in approximative mode of MELTING 4.3.
  5. Tm = 78 + 0.7(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). For RNA.
  6. Tm = 67 + 0.8(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). For RNA/DNA hybrids.
  7. Tm = 81.5 + 0.41(%GC) - 600/N + 16.6 x log[Na+] Used by Primer3Plus to calculate the product Tm. Default set.
  8. Tm = 77.1 + 0.41(%GC) - 528/N + 11.7 x log[Na+] (von Ahsen et al. 2001, Clin Chem 47: 1956-1961). Recommended 'as a tradeoff between accuracy and ease of use'.

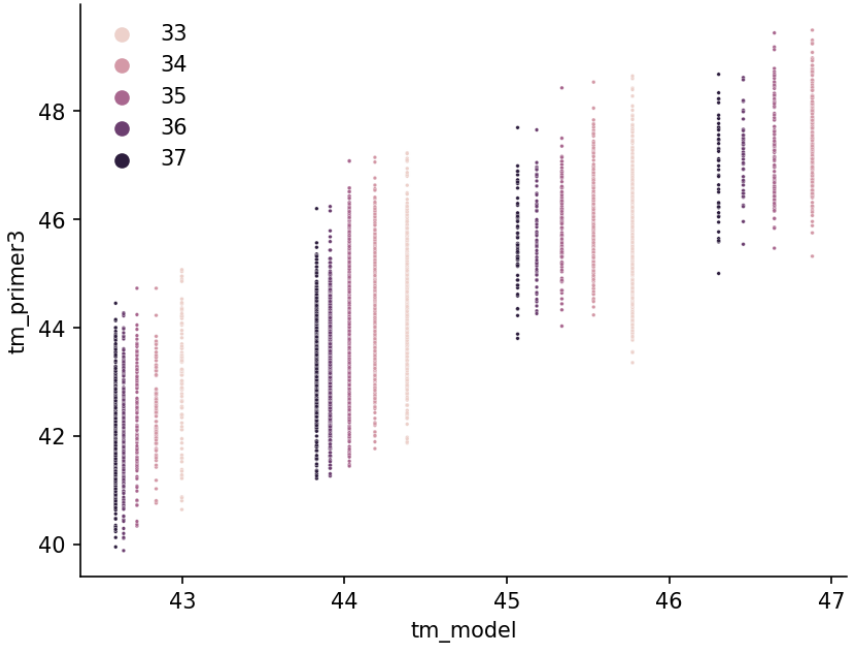
```
]: TRAINING_LENGTH_RANGES = [
    (10, 12),
    (20, 40),
    (30, 37),
    (60, 80),
    (80, 100),
]

for min_len, max_len in TRAINING_LENGTH_RANGES:

    # train a linear tm model on random data
    n_per_l = 1000
    training_seqs = gen_test_seqs(min_len, max_len, n_per_l)

    # train a tm linear model
    tm_model = fit_linear_model(training_seqs)
    print(min_len, max_len, tm_model)
```

```
10 12 TmModel(len_param=3.187835328256666, gc_param=4.057706333755891, intercept=-21.45680965217028, score=0.9345150743569315)
20 40 TmModel(len_param=0.02676952406550645, gc_param=1.5113619367336089, intercept=48.36501361210223, score=0.9363268218093916)
30 37 TmModel(len_param=-0.08850041646535836, gc_param=1.3639873481975235, intercept=55.08006277860753, score=0.9451095795387607)
60 80 TmModel(len_param=-0.15275288137376797, gc_param=0.5892562606305541, intercept=76.57753202553275, score=0.9748179968126922)
80 100 TmModel(len_param=-0.15342726230216325, gc_param=0.45625613513792596, intercept=81.7928942823475, score=0.9953643959868029)
```



numpy example

GC binary mask → GC k-mer grid

```
seq = 'TTTTCATCTTTGGGCGTGCTTACCCCTAGATACACATTCAGTAGTTACAA'
seq
```

```
'TTTTCATCTTTGGGCGTGCTTACCCCTAGATACACATTCAGTAGTTACAA'
```

```
gc_array = gc_array_from_seq(seq)
print(gc_array)
```

```
[0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0
 0 1 0 1 0 0 1 0 0 0 1 0 0]
```

```
idx_grid = np.arange(config['max_length']) + probe_starts[:,np.newaxis]
gc_grid = gc_array[idx_grid]
print(gc_grid)
```

```
[[0 0 0 0 1 0 0 1 0 0 0 1]
 [0 0 0 1 0 0 1 0 0 0 1 1]
 [0 0 1 0 0 1 0 0 0 1 1 1]
 [0 1 0 0 1 0 0 0 1 1 1 1]
 [1 0 0 1 0 0 0 1 1 1 1 1]
 [0 0 1 0 0 0 1 1 1 1 1 0]
 [0 1 0 0 0 1 1 1 1 1 0 1]
 [1 0 0 0 1 1 1 1 1 0 1 1]
 [0 0 0 1 1 1 1 1 0 1 1 0]
 [0 0 1 1 1 1 1 0 1 1 0 0]
 [0 1 1 1 1 1 0 1 1 0 0 0]
 [1 1 1 1 1 0 1 1 0 0 0 1]
 [1 1 1 1 0 1 1 0 0 0 1 1]
```

Rolling cumulative sum of GC

```
gc_grid = np.cumsum(gc_grid, axis=1)
gc_grid
```

```
array([[0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 3],
       [0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 4],
       [0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 4, 5],
       [0, 1, 1, 1, 2, 2, 2, 2, 3, 4, 5, 6],
       [1, 1, 1, 2, 2, 2, 2, 3, 4, 5, 6, 7],
       [0, 0, 1, 1, 1, 1, 2, 3, 4, 5, 6, 6],
       [0, 1, 1, 1, 1, 2, 3, 4, 5, 6, 6, 7],
       [1, 1, 1, 1, 2, 3, 4, 5, 6, 6, 7, 8],
       [0, 0, 0, 1, 2, 3, 4, 5, 5, 6, 7, 7],
       [0, 0, 1, 2, 3, 4, 5, 5, 6, 7, 7, 7],
       [0, 1, 2, 3, 4, 5, 5, 6, 7, 7, 7, 7],
       [1, 2, 3, 4, 5, 5, 6, 7, 7, 7, 7, 8],
       [1, 2, 3, 4, 4, 5, 6, 6, 6, 6, 7, 8],
       [1, 2, 3, 3, 4, 5, 5, 5, 5, 6, 7, 8],
```

Vectorized, multi-length Tm predictions

```
array([[37.754, 39.754, 41.754, 43.754, 47.131],
       [37.754, 39.754, 41.754, 45.131, 48.508],
       [37.754, 39.754, 43.131, 46.508, 49.885],
       [37.754, 41.131, 44.508, 47.885, 51.262],
       [39.131, 42.508, 45.885, 49.262, 52.639],
       [39.131, 42.508, 45.885, 49.262, 51.262],
       [40.508, 43.885, 47.262, 49.262, 52.639],
       [41.885, 45.262, 47.262, 50.639, 54.016],
       [41.885, 43.885, 47.262, 50.639, 52.639],
       [41.885, 45.262, 48.639, 50.639, 52.639],
       [43.262, 46.639, 48.639, 50.639, 52.639],
```

Testing run time on chm13 chr1

Job 290600758 (om1_mining_benchmark) Complete

root <root@gs.washington.edu>
to concamp

Job 290600758 (om1_mining_benchmark) Complete
User = concamp
Queue
Host
Start Time = 10/16/2022 21:31:56.834
End Time = 10/17/2022 09:31:50.716
User Time = 11:56:12
System Time = 00:00:27
Wallclock Time = 11:59:53
CPU = 11:56:40
Max vmem = 33.009G
Max rss = 32.672G
Exit Status = 0

Job 290669128 (om2_mining_benchmark

root <root@gs.washington.edu>
to concamp

Job 290669128 (om2_mining_benchmark_16_640ch) Complete
User = concamp
Queue
Host
Start Time = 10/18/2022 02:34:08.207
End Time = 10/18/2022 02:39:40.177
User Time = 00:20:29
System Time = 00:02:21
Wallclock Time = 00:05:31
CPU = 00:22:50
Max vmem = 103.109G
Max rss = 31.259G
Exit Status = 0

Outline

- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

8bit-int encoding DNA

Encoding scheme:

A = 0
C = 1
G = 2
T = 3

Input: ‘ACGTAACCGGTT’

(as FASTA file)

Output: [0, 1, 2, 3, 0, 0, 1, 1, 2, 2, 3, 3]

8-bit Integers

Can store 256 values (0 – 255)

```
[[0 0 0 0 0 0 0 0]   0
 [0 0 0 0 0 0 0 1]   1
 [0 0 0 0 0 0 1 0]   2
 [0 0 0 0 0 0 1 1]   3
 [0 0 0 0 0 1 0 0]   4

       [...]

 [1 1 1 1 1 1 0 0]   252
 [1 1 1 1 1 1 0 1]   253
 [1 1 1 1 1 1 1 0]   254
 [1 1 1 1 1 1 1 1]]  255
```

Overflow example

```python
# start with int arrays of both data types
a = np.arange(5, dtype=int)
b = np.arange(5, dtype=np.uint8)

# 8-bit overflow demo
print(f'{a} (a)')
print(f'{b} (b)\n')
print(f'{a * 100} (a * 100)')
print(f'{b * 100} (b * 100)')
```

```
[0 1 2 3 4] (a)
[0 1 2 3 4] (b)

[  0 100 200 300 400] (a * 100)
[  0 100 200  44 144] (b * 100)
```

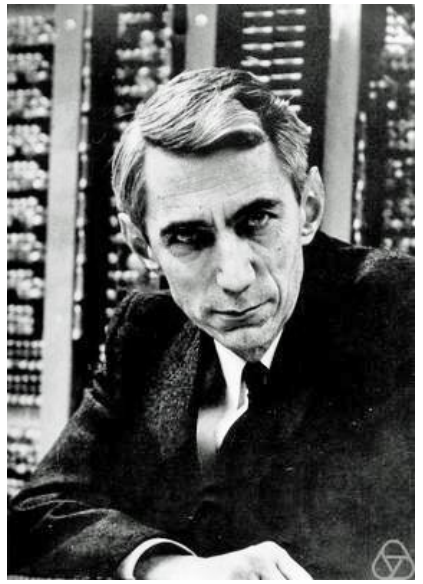
8bit-int encoding DNA

8-bit encoding DNA, 1 byte per nucleotide

```python
# generate a random int DNA sequence using different data types
k = 1000
print(f'DNA sequence length = {k}. Size in RAM:\n')
print(f'default ({np.random.randint(0, 4, k).dtype}):\t{np.random.randint(0, 4, k).nbytes} bytes')
print(f'\tint:\t\t{np.random.randint(0, 4, k, dtype=int).nbytes} bytes')
print(f'\tint32:\t\t{np.random.randint(0, 4, k, dtype=np.int32).nbytes} bytes')
print(f'\tint16:\t\t{np.random.randint(0, 4, k, dtype=np.int16).nbytes} bytes')
print(f'\tint8:\t\t{np.random.randint(0, 4, k, dtype=np.int8).nbytes} bytes')
```

```
DNA sequence length = 1000. Size in RAM:

default (int64):        8000 bytes
        int:            8000 bytes
        int32:          4000 bytes
        int16:          2000 bytes
        int8:           1000 bytes
```



Claude Shannon

- 4 x (0.25 * log2(0.25)) = 2.0 bits

Note: could use 2 bits to store nucleotides in theory, but 1 byte (8 bits) is practical in python.

Encoding scheme:

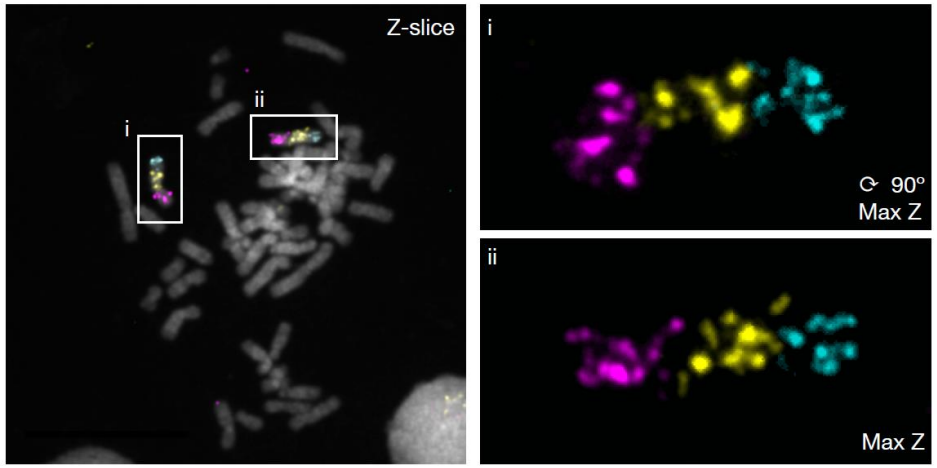
A = 0 = 00
C = 1 = 01
G = 2 = 10
T = 3 = 11

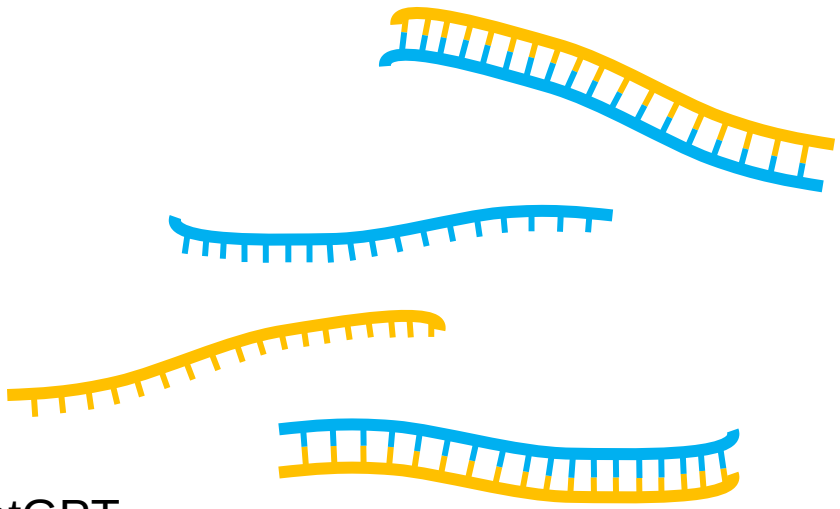
Information entropy

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

8bit encoding DNA sequences

Pattern 1: 3-color side-by-side





write python code to efficiently load a single DNA sequence contained in a fasta file and encode it as an 8-bit integer array using numpy

```python
import numpy as np

def load_dna_sequence(filename):
    """
    Loads a single DNA sequence from a fasta file and encodes it as an 8-bit integer array using numpy
    :param filename: the name of the fasta file
    :return: an 8-bit integer numpy array representing the DNA sequence
    """
    # Create a dictionary mapping each nucleotide to its encoding
    encoding = {'A': 0, 'C': 1, 'G': 2, 'T': 3}

    # Initialize an empty list to store the encoded sequence
    sequence = []

    # Read the fasta file line by line
    with open(filename, 'r') as file:
        for line in file:
            # Skip the header line starting with '>'
            if line.startswith('>'):
                continue

            # Iterate over each nucleotide in the line and encode it
            for nucleotide in line.strip():
                sequence.append(encoding[nucleotide])

    # Convert the list to a numpy array and return it
    return np.array(sequence, np.uint8)
```

chatGPT

8bit-int encoding DNA

Fasta file to seq string using biopython

```
from Bio import SeqIO

FASTA_FILE = '../data/chrI.fa'

# load fasta sequence using biopython
seq_record = list(SeqIO.parse(FASTA_FILE, 'fasta')).pop()
seq_record
```

```
SeqRecord(seq=Seq('CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACCCAC
A...GTA'), id='chrI', name='chrI', description='chrI', dbxrefs=[])
```

List comprehension; DNA lookup table

```
# encoding scheme lookup table
base_map = {
    'A': 0,
    'C': 1,
    'G': 2,
    'T': 3,
}

# encode seq as 8bit int array
encoded = np.array([base_map[base] for base in seq], dtype=np.uint8)

encoded
```

```
array([1, 1, 0, ..., 2, 3, 0], dtype=uint8)
```

For loop; DNA lookup table

```
# encoding scheme lookup table
base_map = {
    'A': 0,
    'C': 1,
    'G': 2,
    'T': 3,
}

# encode seq as 8bit int array
encoded = []
for base in seq:
    encoded.append(base_map[base])

# convert to numpy array
encoded = np.array(encoded, dtype=np.uint8)

encoded
```

```
array([1, 1, 0, ..., 2, 3, 0], dtype=uint8)
```

chm13 chr1: 28 sec

8bit-int encoding DNA

String translation in python

```
raw = 'abcdefghijklmnopqrstuvwxyz'
encoded = 'defghijklmnopqrstuvwxyzabc'

s = 'hello this is an example string.'

s.translate(s.maketrans(raw, encoded))
```

```
'khoor wklv lv dq hadpsoh vwulqj.'
```

DNA encoding with string translation

```
s = 'AAACCCGGTTT'

s.translate(s.maketrans('ACGT', '0123'))
```

```
'00011122333'
```

```
np.array(list(s.translate(s.maketrans('ACGT', '0123'))), dtype=np.uint8)
```

```
array([0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 3], dtype=uint8)
```

A bit faster

Not robust to non-ACGT characters in FASTA

chm13 chr1: 17 sec

8bit-int encoding DNA

Loading fasta with pyfaidx

```
import pyfaidx

seq_record = pyfaidx.Fasta(FASTA_FILE, sequence_always_upper=True)

np.asarray(seq_record)
```

```
array([[b'C', b'C', b'A', ..., b'G', b'T', b'A']], dtype='|S1')
```

Access fasta sequence as numpy array

problem: ascii byte array

```
print(ord('A'))
print(chr(65))
```

```
65
A
```

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

```
1000001 = A = 65
1000011 = C = 67
1000111 = G = 71
1010100 = T = 84
```

8bit-int encoding DNA

Loading fasta with pyfaidx

```python
import pyfaidx

seq_record = pyfaidx.Fasta(FASTA_FILE, sequence_always_upper=True)

np.asarray(seq_record)
```

```
array([[b'C', b'C', b'A', ..., b'G', b'T', b'A']], dtype='|S1')
```

np.view same bytes as different dtype

```python
seq_record = pyfaidx.Fasta(FASTA_FILE, sequence_always_upper=True)

np.asarray(seq_record).view(np.uint8)
```

```
array([[67, 67, 65, ..., 71, 84, 65]], dtype=uint8)
```

Supports arbitrary encoding of non-ACGT characters in FASTA

ASCII int DNA encoding lookup table

```python
# create ascii lookup table to encode fasta characters as ints
DNA_ASCII_LUT = np.zeros(256, dtype=np.uint8)
DNA_ASCII_LUT[[ord(base) for base in 'ACGT']] = [0, 1, 2, 3]

DNA_ASCII_LUT
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=uint8)
```

```
1000001 = A = 65 → 0
1000011 = C = 67 → 1
1000111 = G = 71 → 2
1010100 = T = 84 → 3
```

8bit-int encoding DNA

For loop; DNA lookup table

```python
# encoding scheme lookup table
base_map = {
    'A': 0,
    'C': 1,
    'G': 2,
    'T': 3,
}

# encode seq as 8bit int array
encoded = []
for base in seq:
    encoded.append(base_map[base])

# convert to numpy array
encoded = np.array(encoded, dtype=np.uint8)

encoded
```

```
array([1, 1, 0, ..., 2, 3, 0], dtype=uint8)
```

chm13 chr1: 28 sec

Load and encode with pyfaidx and numpy

```python
import numpy as np
import pyfaidx

FASTA_FILE = '../data/chrI.fa'

# create ascii lookup table to encode fasta characters as ints
DNA_ASCII_LUT = np.zeros(256, dtype=np.uint8)
DNA_ASCII_LUT[[ord(base) for base in 'ACGT']] = [0, 1, 2, 3]

# load fasta file
seq_record = pyfaidx.Fasta(FASTA_FILE, sequence_always_upper=True)

# convert to 8bit int encoded
DNA_ASCII_LUT[np.asarray(seq_record).view(np.uint8)]
```

```
array([[1, 1, 0, ..., 2, 3, 0]], dtype=uint8)
```

16 sec

1 sec

String translation in python

```python
raw = 'abcdefghijklmnopqrstuvwxyz'
encoded = 'defghijklmnopqrstuvwxyzabc'

s = 'hello this is an example string.'

s.translate(s.maketrans(raw, encoded))
```

```
'khoor wklv lv dq hadpsoh vwulqj.'
```

Outline

- Homework 6 overview

- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

- Homework 5 questions

Tm linear model

Bio.SeqUtils.MeltingTemp.Tm_GC(*seq, check=True, strict=True, valueset=7, userset=None, Na=50, K=0, Tris=0, Mg=0, dNTPs=0, saltcorr=0, mismatch=True*)

Return the Tm using empirical formulas based on GC content.

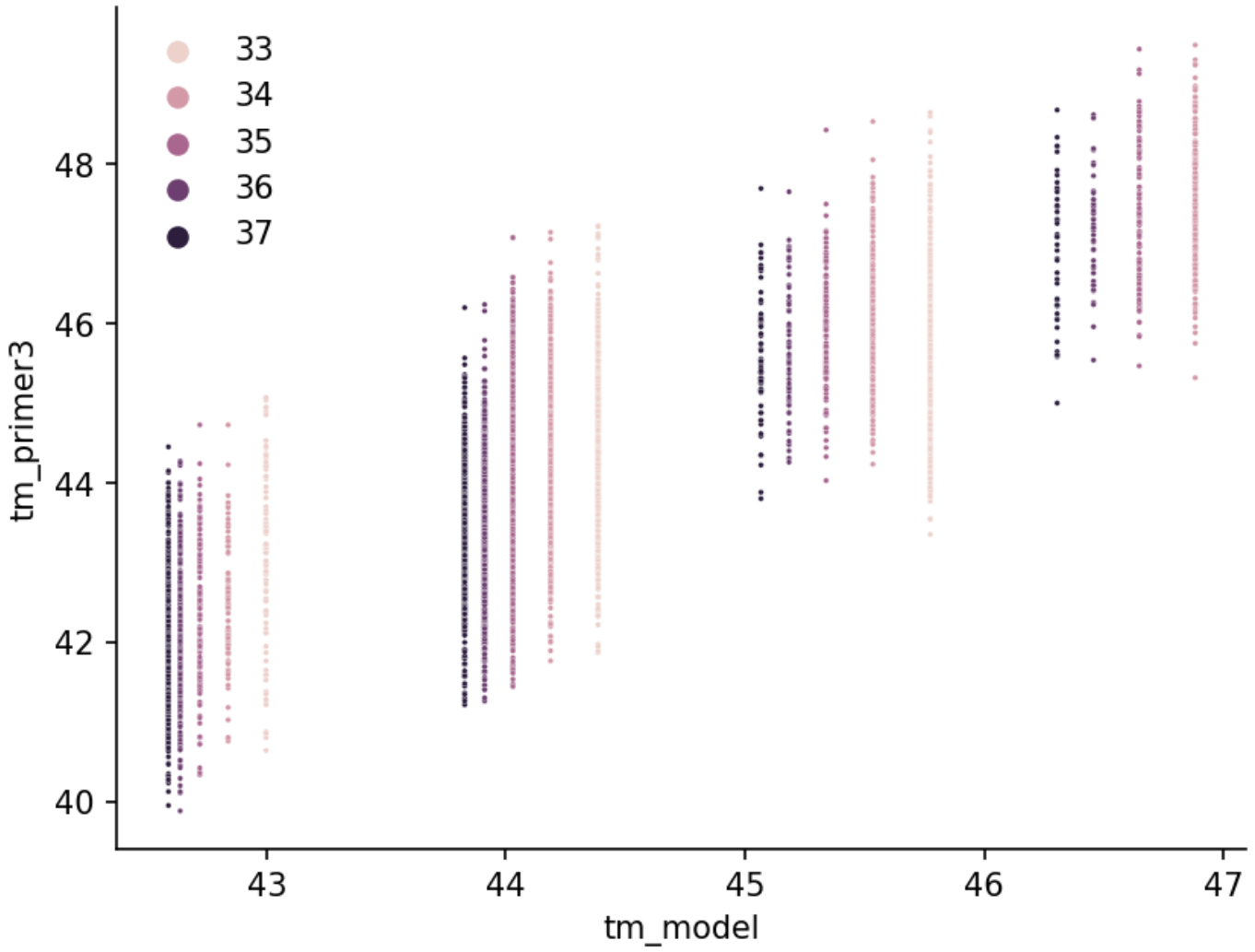
General format: Tm = A + B(%GC) - C/N + salt correction - D(%mismatch)

A, B, C, D: empirical constants, N: primer length D (amount of decrease in Tm per % mismatch) is often 1, but sometimes other values have been used (0.6-1.5). Use 'X' to indicate the mismatch position in the sequence. Note that this mismatch correction is a rough estimate.

```
>>> from Bio.SeqUtils import MeltingTemp as mt
>>> print("%0.2f" % mt.Tm_GC('CTGCTGATXGCACGAGGTTATGG', valueset=2))
69.20
```

Arguments:

- valueset: A few often cited variants are included:
  1. Tm = 69.3 + 0.41(%GC) - 650/N (Marmur & Doty 1962, J Mol Biol 5: 109-118; Chester & Marshak 1993), Anal Biochem 209: 284-290)
  2. Tm = 81.5 + 0.41(%GC) - 675/N - %mismatch 'QuikChange' formula. Recommended (by the manufacturer) for the design of primers for QuikChange mutagenesis.
  3. Tm = 81.5 + 0.41(%GC) - 675/N + 16.6 x log[Na+] (Marmur & Doty 1962, J Mol Biol 5: 109-118; Schildkraut & Lifson 1965, Biopolymers 3: 195-208)
  4. Tm = 81.5 + 0.41(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). This is the standard formula in approximative mode of MELTING 4.3.
  5. Tm = 78 + 0.7(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). For RNA.
  6. Tm = 67 + 0.8(%GC) - 500/N + 16.6 x log([Na+]/(1.0 + 0.7 x [Na+])) - %mismatch (Wetmur 1991, Crit Rev Biochem Mol Biol 126: 227-259). For RNA/DNA hybrids.
  7. Tm = 81.5 + 0.41(%GC) - 600/N + 16.6 x log[Na+] Used by Primer3Plus to calculate the product Tm. Default set.
  8. Tm = 77.1 + 0.41(%GC) - 528/N + 11.7 x log[Na+] (von Ahsen et al. 2001, Clin Chem 47: 1956-1961). Recommended 'as a tradeoff between accuracy and ease of use'.



Nearest-neighbor Tm Calculation

WIKIPEDIA
The Free Encyclopedia

Article | Talk | Read | Edit | View history | Search Wikipedia

Nucleic acid thermodynamics

From Wikipedia, the free encyclopedia

Nucleic acid thermodynamics is the study of how temperature affects the nucleic acid structure of double-stranded DNA (dsDNA). The melting temperature (T_m) is

$AB \leftrightarrow A + B$

The equilibrium constant for this reaction is $K = \frac{[A][B]}{[AB]}$. According to

ΔG, and K is $\Delta G° = -RT\ln K$, where R is the ideal gas law constant, and

$$\Delta G^\circ = -RT \ln \frac{[A][B]}{[AB]}.$$

The melting temperature, T_m, occurs when half of the double-stranded n
will be equal, and equal to half the initial concentration of double-strande
duplex of

$$T_m = -\frac{\Delta G^\circ}{R \ln \frac{[AB]_{initial}}{2}}.$$

Because $\Delta G° = \Delta H° - T\Delta S°$, T_m is also given by

$$T_m = \frac{\Delta H^\circ}{\Delta S^\circ - R \ln \frac{[AB]_{initial}}{2}}.$$

The terms $\Delta H°$ and $\Delta S°$ are usually given for the association and not the
turns into:[13]

$$T_m = \frac{\Delta H^\circ}{\Delta S^\circ + R \ln([A]_{total} - [B]_{total}/2)}$$, where $[B]_{total} \leq [A]_{total}$.

Nearest-neighbor method [edit]

The interaction between bases on different s
of treating a DNA helix as a string of interac
DNA helix as a string of interactions betwee
shown below has nearest-neighbor interacti

```
    ↓ ↓ ↓ ↓ ↓
5' C-G-T-T-G-A 3'
3' G-C-A-A-C-T 5'
```

Table 1. Nearest-neighbor parameters for DNA/DNA duplexes in 1 M NaCl.[13]

Nearest-neighbor sequence (5'-3'/3'-5')	$\Delta H°$ kJ/mol	$\Delta S°$ J/(mol·K)	$\Delta G°_{37}$ kJ/mol
AA/TT	−33.1	−92.9	−4.26
AT/TA	−30.1	−85.4	−3.67
TA/AT	−30.1	−89.1	−2.50
CA/GT	−35.6	−95.0	−6.12
GT/CA	−35.1	−93.7	−6.09
CT/GA	−32.6	−87.9	−5.40
GA/CT	−34.3	−92.9	−5.51
CG/GC	−44.4	−113.8	−9.07
GC/CG	−41.0	−102.1	−9.36
GG/CC	−33.5	−83.3	−7.66
Terminal A/T base pair	9.6	17.2	4.31
Terminal G/C base pair	0.4	−11.7	4.05

Nearest-neighbor Tm Calculation

Bio.SeqUtils.MeltingTemp.Tm_NN(*seq, check=True, strict=True, c_seq=None, shift=0, nn_table=None, tmm_table=None, imm_table=None, de_table=None, dnac1=25, dnac2=25, selfcomp=False, Na=50, K=0, Tris=0, Mg=0, dNTPs=0, saltcorr=5*)

Return the Tm using nearest neighbor thermodynamics.

Arguments:

- seq: The primer/probe sequence as string or Biopython sequence object. For RNA/DNA hybridizations seq must be the RNA sequence.

```
# Allawi and SantaLucia (1997), Biochemistry 36: 10581-10594
DNA_NN3 = {
    "init": (0, 0), "init_A/T": (2.3, 4.1), "init_G/C": (0.1, -2.8),
    "init_oneG/C": (0, 0), "init_allA/T": (0, 0), "init_5T/A": (0, 0),
    "sym": (0, -1.4),
    "AA/TT": (-7.9, -22.2), "AT/TA": (-7.2, -20.4), "TA/AT": (-7.2, -21.3),
    "CA/GT": (-8.5, -22.7), "GT/CA": (-8.4, -22.4), "CT/GA": (-7.8, -21.0),
    "GA/CT": (-8.2, -22.2), "CG/GC": (-10.6, -27.2), "GC/CG": (-9.8, -24.4),
    "GG/CC": (-8.0, -19.9)}
```

Nearest-neighbor Tm Calculation

Bio.SeqUtils.MeltingTemp.salt_correction(*Na=0, K=0, Tris=0, Mg=0, dNTPs=0, method=1, seq=None*)

Calculate a term to correct Tm for salt ions.

Depending on the Tm calculation, the term will correct Tm or entropy. To calculate corrected Tm values, different operations need to be applied:

- methods 1-4: Tm(new) = Tm(old) + corr
- method 5: deltaS(new) = deltaS(old) + corr
- methods 6+7: Tm(new) = 1/(1/Tm(old) + corr)

Parameters:

- Na, K, Tris, Mg, dNTPS: Millimolar concentration of respective ion. To have a simple 'salt correction', just pass Na. If any of K, Tris, Mg and dNTPS is non-zero, a 'sodium-equivalent' concentration is calculated according to von Ahsen et al. (2001, Clin Chem 47: 1956-1961): [Na_eq] = [Na+] + [K+] + [Tris]/2 + 120*([Mg2+] - [dNTPs])^0.5 If [dNTPs] >= [Mg2+]: [Na_eq] = [Na+] + [K+] + [Tris]/2
- method: Which method to be applied. Methods 1-4 correct Tm, method 5 corrects deltaS, methods 6 and 7 correct 1/Tm. The methods are:
  1. 16.6 x log[Na+] (Schildkraut & Lifson (1965), Biopolymers 3: 195-208)
  2. 16.6 x log([Na+]/(1.0 + 0.7*[Na+])) (Wetmur (1991), Crit Rev Biochem Mol Biol 126: 227-259)
  3. 12.5 x log(Na+] (SantaLucia et al. (1996), Biochemistry 35: 3555-3562
  4. 11.7 x log[Na+] (SantaLucia (1998), Proc Natl Acad Sci USA 95: 1460-1465
  5. Correction for deltaS: 0.368 x (N-1) x ln[Na+] (SantaLucia (1998), Proc Natl Acad Sci USA 95: 1460-1465)
  6. (4.29(%GC)-3.95)x1e-5 x ln[Na+] + 9.40e-6 x ln[Na+]^2 (Owczarzy et al. (2004), Biochemistry 43: 3537-3554)
  7. Complex formula with decision tree and 7 empirical constants. Mg2+ is corrected for dNTPs binding (if present) (Owczarzy et al. (2008), Biochemistry 47: 5336-5353)

Integer seq encoding for NN thermo calculations

Encode seq as int array

```
AAAAAAAA --> [0 0 0 0 0 0 0 0]

ACGTACGT --> [0 1 2 3 0 1 2 3]

CGCGCGCG --> [1 2 1 2 1 2 1 2]

ATATATAT --> [0 3 0 3 0 3 0 3]
```

Convert to dinucleotide array

```
AAAAAAAA --> [0 0 0 0 0 0 0 0]

[[0 0][0 0][0 0][0 0][0 0][0 0][0 0]]

ACGTACGT --> [0 1 2 3 0 1 2 3]

[[0 1][1 2][2 3][3 0][0 1][1 2][2 3]]

CGCGCGCG --> [1 2 1 2 1 2 1 2]

[[1 2][2 1][1 2][2 1][1 2][2 1][1 2]]

ATATATAT --> [0 3 0 3 0 3 0 3]

[[0 3][3 0][0 3][3 0][0 3][3 0][0 3]]
```

Batch process sequences

```
[
  'AAAAAAAA',
  'ACGTACGT',
  'CGCGCGCG',
  'ATATATAT'
]

[
  [[0 0][0 0][0 0][0 0][0 0][0 0][0 0]],
  [[0 1][1 2][2 3][3 0][0 1][1 2][2 3]],
  [[1 2][2 1][1 2][2 1][1 2][2 1][1 2]],
  [[0 3][3 0][0 3][3 0][0 3][3 0][0 3]],
]
```

Storing dH and dS NN params

```
#
# base dH values for the 16 dinucleotides
#

# lookups are done via DINUC_DH_LUT[first_base][second_base] using int encoded sequences
DINUC_DH_LUT = np.array([
    [ -7.9,  -8.4,  -7.8,  -7.2], # AA, AC, AG, AT
    [ -8.5,  -8.0, -10.6,  -7.8], # CA, CC, CG, CT
    [ -8.2,  -9.8,  -8.0,  -8.4], # GA, GC, GG, GT
    [ -7.2,  -8.2,  -8.5,  -7.9], # TA, TC, TG, TT
], dtype=NN_DTYPE)



#
# base dS values for the 16 dinucleotides
#

# lookups are done via DINUC_DH_LUT[first_base][second_base] using int encoded sequences
DINUC_DS_LUT = np.array([
    [-22.2, -22.4, -21.0, -20.4], # AA, AC, AG, AT
    [-22.7, -19.9, -27.2, -21.0], # CA, CC, CG, CT
    [-22.2, -24.4, -19.9, -22.4], # GA, GC, GG, GT
    [-21.3, -22.2, -22.7, -22.2], # TA, TC, TG, TT
], dtype=NN_DTYPE)
```

```
DINUC_DS_LUT
[[-22.2 -22.4 -21.  -20.4]
 [-22.7 -19.9 -27.2 -21. ]
 [-22.2 -24.4 -19.9 -22.4]
 [-21.3 -22.2 -22.7 -22.2]]

DINUC_DH_LUT
[[ -7.9  -8.4  -7.8  -7.2]
 [ -8.5  -8.  -10.6  -7.8]
 [ -8.2  -9.8  -8.   -8.4]
 [ -7.2  -8.2  -8.5  -7.9]]
```

Storing dH and dS NN params

```
#
# terminal nucleotide dH adjustments
#

# table with respect to 5' base in the dinucleotide
TERMINAL_5_DH_LUT = np.array([
    [2.3, 2.3, 2.3, 2.3], #  AA,  AC,  AG,  AT
    [0.1, 0.1, 0.1, 0.1], # *CA, *CC, *CG, *CT
    [0.1, 0.1, 0.1, 0.1], # *GA, *GC, *GG, *GT
    [2.3, 2.3, 2.3, 2.3], #  TA,  TC,  TG,  TT
], dtype=NN_DTYPE)

# table with respect to 3' base in the dinucleotide
#  AA,  AC*, AG*, AT
#  CA,  CC*, CG*, CT
#  GA,  GC*, GG*, GT
#  TA,  TC*, TG*, TT
TERMINAL_3_DH_LUT = TERMINAL_5_DH_LUT.T
```

```
#
# terminal nucleotide dS adjustments
#

# table with respect to 5' base in the dinucleotide
TERMINAL_5_DS_LUT = np.array([
    [ 4.1,  4.1,  4.1,  4.1], #  AA,  AC,  AG,  AT
    [-2.8, -2.8, -2.8, -2.8], # *CA, *CC, *CG, *CT
    [-2.8, -2.8, -2.8, -2.8], # *GA, *GC, *GG, *GT
    [ 4.1,  4.1,  4.1,  4.1], #  TA,  TC,  TG,  TT
], dtype=NN_DTYPE)

# table with respect to 3' base in the dinucleotide
#  AA,  AC*, AG*, AT
#  CA,  CC*, CG*, CT
#  GA,  GC*, GG*, GT
#  TA,  TC*, TG*, TT
TERMINAL_3_DS_LUT = TERMINAL_5_DS_LUT.T
```

Batch process sequences

```
[
  'AAAAAAAA',
  'ACGTACGT',
  'CGCGCGCG',
  'ATATATAT'
]

[
  [[0 0][0 0][0 0][0 0][0 0][0 0][0 0][0 0]],
  [[0 1][1 2][2 3][3 0][0 1][1 2][2 3]],
  [[1 2][2 1][1 2][2 1][1 2][2 1][1 2]],
  [[0 3][3 0][0 3][3 0][0 3][3 0][0 3]],
]
```

Vectorized NN Tm Calculation

Batch process sequences

```
[
  'AAAAAAAA',
  'ACGTACGT',
  'CGCGCGCG',
  'ATATATAT'
]

[
  [[0 0][0 0][0 0][0 0][0 0][0 0][0 0]],
  [[0 1][1 2][2 3][3 0][0 1][1 2][2 3]],
  [[1 2][2 1][1 2][2 1][1 2][2 1][1 2]],
  [[0 3][3 0][0 3][3 0][0 3][3 0][0 3]],
]
```

Batch lookup

```
delta_H = DINUC_DH_LUT[dinuc_grid[:,:,0],dinuc_grid[:,:,1]]

delta_S = DINUC_DS_LUT[dinuc_grid[:,:,0],dinuc_grid[:,:,1]]
```

ΔH

```
[[ -7.9  -7.9  -7.9  -7.9  -7.9  -7.9  -7.9]
 [ -8.4 -10.6  -8.4  -7.2  -8.4 -10.6  -8.4]
 [-10.6  -9.8 -10.6  -9.8 -10.6  -9.8 -10.6]
 [ -7.2  -7.2  -7.2  -7.2  -7.2  -7.2  -7.2]]
```

ΔS

```
[[-22.2 -22.2 -22.2 -22.2 -22.2 -22.2 -22.2]
 [-22.4 -27.2 -22.4 -21.3 -22.4 -27.2 -22.4]
 [-27.2 -24.4 -27.2 -24.4 -27.2 -24.4 -27.2]
 [-20.4 -21.3 -20.4 -21.3 -20.4 -21.3 -20.4]]
```

Vectorized NN Tm Calculation

1. Apply 5’ terminal base adjustments to dH and dS

2. Take rolling cumulative sum across rows

ΔH

```
[[ -7.9  -7.9  -7.9  -7.9  -7.9  -7.9  -7.9]
 [ -8.4 -10.6  -8.4  -7.2  -8.4 -10.6  -8.4]
 [-10.6  -9.8 -10.6  -9.8 -10.6  -9.8 -10.6]
 [ -7.2  -7.2  -7.2  -7.2  -7.2  -7.2  -7.2]]
```

ΔS

```
[[-22.2 -22.2 -22.2 -22.2 -22.2 -22.2 -22.2]
 [-22.4 -27.2 -22.4 -21.3 -22.4 -27.2 -22.4]
 [-27.2 -24.4 -27.2 -24.4 -27.2 -24.4 -27.2]
 [-20.4 -21.3 -20.4 -21.3 -20.4 -21.3 -20.4]]
```

Cumulative sum across rows

```
[[ -7.9 -15.8 -23.7 -31.6 -39.5 -47.4 -55.3]
 [ -8.4 -19.  -27.4 -34.6 -43.  -53.6 -62. ]
 [-10.6 -20.4 -31.  -40.8 -51.4 -61.2 -71.8]
 [ -7.2 -14.4 -21.6 -28.8 -36.  -43.2 -50.4]]

[[ -22.2  -44.4  -66.6  -88.8 -111.  -133.2 -155.4]
 [ -22.4  -49.6  -72.   -93.3 -115.7 -142.9 -165.3]
 [ -27.2  -51.6  -78.8 -103.2 -130.4 -154.8 -182. ]
 [ -20.4  -41.7  -62.1  -83.4 -103.8 -125.1 -145.5]]
```

3. Apply 3’ terminal base adjustment to each column

Now each position in each matrix is a sequence’s dH or dS value

4. Apply length-dependent salt correction to each dS matrix column

5. Calculate Tm using formula, using entire dH and dS matrices

Initial Benchmark Results

CHM13 whole-genome run:

- `qsub` each chromosome in parallel
  - wait 1 sec between qsubs
- provide 4 cores per chromosome
- provide 4G RAM per core

3,117,291,170 probe start sites in CHM13
X
8 probe lengths evaluated [30, ..., 37]

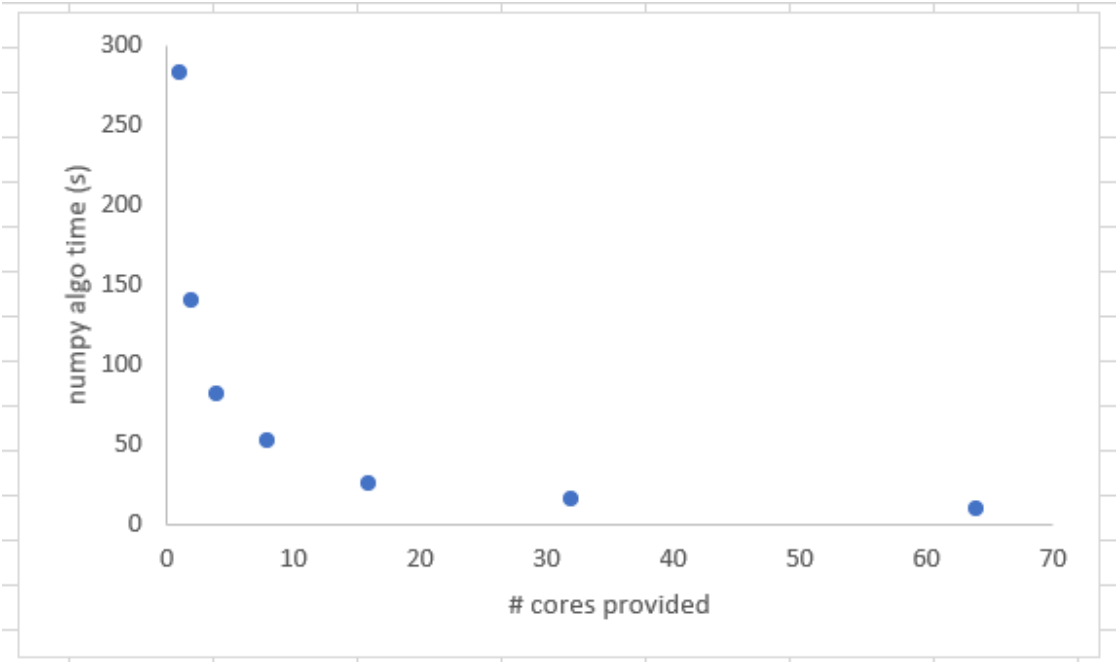
24,938,329,360 nearest neighbor Tms computed

Wallclock results:

5 min with overlapping probes

2m52s without overlapping probes

With multiprocessing



OligoMiner2 command-line interface (CLI)

OligoMiner provides a rapid, flexible environment for the design of genome-scale oligonucleotide in situ hybridization probes

Brian J. Beliveau[a,b,1], Jocelyn Y. Kishi[a,b], Guy Nir[c], Hiroshi M. Sasaki[a,b], Sinem K. Saka[a,b], Son C. Nguyen[c,2], Chao-ting Wu[c], and Peng Yin[a,b,1]

OligoMiner2 (coming soon):

- Toolkit for multiplexed DNA/RNA FISH probe design
- Python package

```
import oligominer
```

- Command-line interface

```
$ oligominer mine_probes
```

```
$ oligominer
Usage: oligominer [OPTIONS] COMMAND [ARGS]...

  Version:   0.0.1
  Docs:      https://om2-docs.s3.us-west-2.amazonaws.com/v0.0.1/index.html
  Code:      https://github.com/beliveau-lab/OligoMiner2

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  align_probes        Align mined candidate probes to a reference genome.
  build_bowtie        Build a bowtie2 index from a reference genome.
  build_jellyfish     Build a jellyfish k-mer index from a reference genome.
  create_rna_targets  Create an RNA target file from a GTF annotations file.
  merge_fastas        Merge multiple fasta files into a single file.
  mine_probes         Mine candidate probes from an input fasta sequence.
  preprocess_fasta    Filter and/or split a reference genome fasta file.
  preprocess_gtf      Filter a GTF annotations file.
  print_chrom_sizes   Print chrom.sizes information for a reference genome.
  verify_install      Verify that dependencies have been installed.
$
```

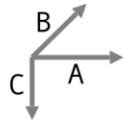
Outline

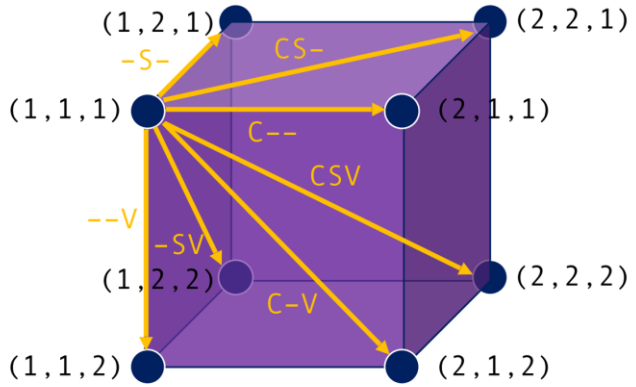
- Homework 6 overview

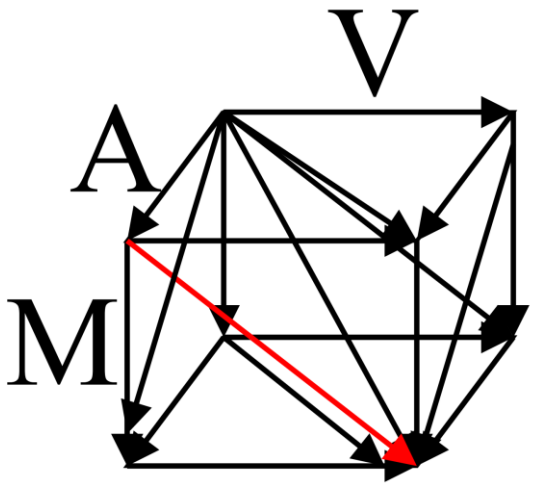
- Related topics:
  - DNA melting temp (Tm) for FISH probe design
  - GC-based linear model for DNA Tm
  - 8-bit integer encoding DNA sequences
  - Vectorized nearest-neighbor DNA Tm calculation

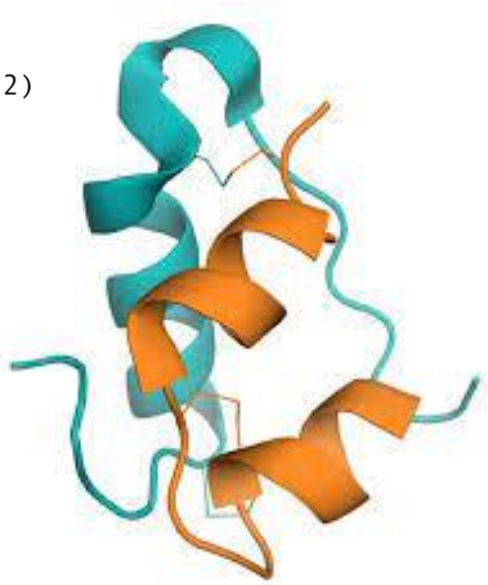
- Homework 5 questions

Homework 5 Questions ?



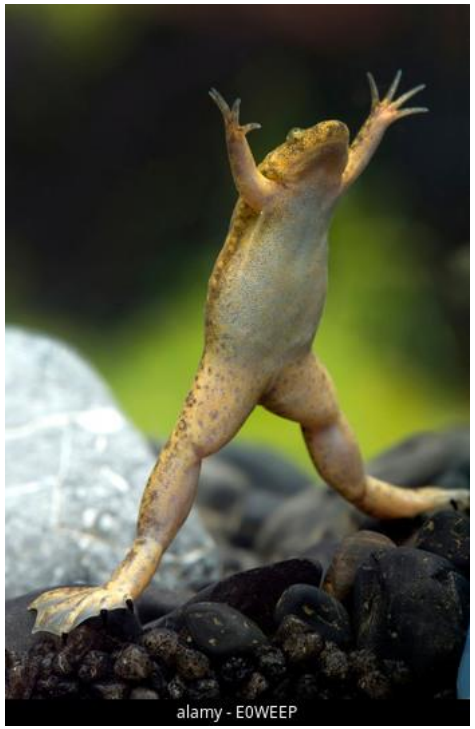






Insulin

Local alignment:
KKK
DLK
YWY
G--
LFL
KVN
REH
IPI
QRQ
KPN
SNS
AVV
FFF
MVE
GSG
SGE
LVI
KKS
KDE
HSP



Reminders

- Homework 5 due this Sunday Feb. 12, 11:59 pm

- Homework 6 due next Sunday Feb. 19, 11:59 pm

. . .