melting DNA…

salts (Na+ and K+)

higher salt concentrations increase melting temperature

**Fig. 1.** Temperature dependence of the absorbancy of *Ps. aeruginosa* DNA at salt concentration between 0.01 and 0.6M. The per cent increase is relative to the absorbancy at 25°C.

Schildkraut and Lifson (1965)
...melting DNA...

composition (%GC)

higher %GC increase melting temperature

Fig. 2. Thermal denaturation curves of DNA isolated from various bacteria. All samples at 20 μg/ml in SSC. Relative absorbance (corrected for thermal expansion) measured at the elevated temperatures. The midpoint of each transition ($T_m$) is indicated at the arrow. DNA isolated from *Proteus vulgaris* (open circles); *Bacillus licheniformis* (filled circles); *Klebsiella pneumoniae* (inverted triangles); *Pseudomonas fluorescens* (filled squares); *Sarcina lutea* (open squares).

Mandel and Marmur (1968)
melting DNA...

polyols (e.g. betaine)

higher betaine concentration decreases melting temperature

**FIGURE 2**: Variation of $T_m$ with betaine concentration for DNAs of varying base composition. Melting was monitored by the UV absorbance at 260 nm. Buffers and heating rates were as in Figure 1A. (Open circles) *M. lysodeikticus* DNA (72% GC); (open squares) *E. coli* DNA (50% GC); (open diamonds) calf thymus DNA (42% GC); (open triangles) *Cl. perfringens* DNA (26% GC); (filled triangles) poly(dA-dT); (filled circles) poly(dG-dC).

Rees et al. (1993)
empirical formulae

empirical formulae (usually) use some combination of:
  
salt concentration
%GC
DNA length
DNA concentration
  
formamide concentration
  
e.g. Howell et al. (1979), Wetmur (1991)

von Ahsen et al. (2001) revised the most common formulae using regression on 475 oligos
vertical stacking

vertical stacking is, by far, the greatest contributor to helix stability

nearest-neighbor interactions between bases predict helix stability

ratio of $\Delta H/\Delta S$ (Borer et al. 1974) $E$ (Khandelwal and Bhyravabhotla 2010)

can be used to estimate $T_m$ (with some ‘corrections’).
Table 1. Comparison of published NN free energy parameters at 37°C

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Parameter, kcal/mol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Go( ref. 7)</td>
</tr>
<tr>
<td>AA/TT</td>
<td>-0.43</td>
</tr>
<tr>
<td>AT/TA</td>
<td>-0.27</td>
</tr>
<tr>
<td>TA/AT</td>
<td>-0.22</td>
</tr>
<tr>
<td>CA/GT</td>
<td>-0.97</td>
</tr>
<tr>
<td>GT/CA</td>
<td>-0.98</td>
</tr>
<tr>
<td>CT/GA</td>
<td>-0.83</td>
</tr>
<tr>
<td>GA/CT</td>
<td>-0.93</td>
</tr>
<tr>
<td>CG/GC</td>
<td>-1.70</td>
</tr>
<tr>
<td>GC/CG</td>
<td>-1.64</td>
</tr>
<tr>
<td>GG/CC</td>
<td>-1.22</td>
</tr>
<tr>
<td>Average</td>
<td>-0.92</td>
</tr>
<tr>
<td>Init. w/term. G·C*</td>
<td>NA</td>
</tr>
<tr>
<td>Init. w/term. A·T*</td>
<td>NA</td>
</tr>
<tr>
<td>Sodium concentration, M</td>
<td>0.0195</td>
</tr>
<tr>
<td>Rank of stacking matrix</td>
<td>8</td>
</tr>
</tbody>
</table>

ΔH/ΔS

SantaLucia (1998)
nearest–neighbor algorithms

Borer et al. (1974), Rychlik et al. (1990), Khandelwal and Bhyravabhotla (2010)

start at the 5' end; for each base:

  sum the look up values for the base and its 3' neighbor

correct the sum:

  ratio of $\Delta H/\Delta S$ or oligo length

  salt concentration

  DNA concentration
variable scopes (simplified)

local: {} (let)
the variable is declared and exists within {}
everything within the same {} can access the variable

global
the variable exists everywhere in the script
everything can access the variable
if another variable of the same name is declared, the local version has precedence in its sphere of influence
What is it?

determine what kind of object a variable is
output to console.log
typeof() returns a string describing the object
undefined (was not declared), boolean, number, string, symbol, object (null or none of the above)
Object.prototype.toString() returns a string describing the object
more reliable
the value that is not there

undefined: was not declared or set
  an error in the code and/or asynchronous execution
null: was declared, but not set
  an error in the code and/or asynchronous execution
NaN (not a number): could not be coerced into a number
  an error in the code (and/or asynchronous execution)
Infinity: an error in the code, usually divide by zero

if(typeof(x) !== 'undefined'); if(x !== null);
if(isNaN(x) === false); if(isFinite(x) === false);
functions

can be declared anywhere (even within other functions)

```javascript
function x(){ /* a function x */ };
let x = function(){}; /* reference to a function */
```
can be anonymous (nameless)

often passed to another function as an argument

‘callback’ or ‘closure’ functions

```javascript
object.method(function(input){})
```
return values are optional
function variables

variables in the environment are passed into functions
arguments (objects) are passed as named variables

function(argument){};

if there is an argument and variable in the environment
with the same name, the argument’s value is used

variables declared within a function are destroyed when
the function returns

functions are often evaluated before they are called

use if() statements to check the input/prevent crashes
objects...

really just custom data structures
  group similar variables together under a single name
  keep related data synchronized

create one instance using ‘let’

create many instances using ‘function’ and ‘new’

```javascript
function Sequence(s){
  this.seq = s.toUpperCase().replace(/[^ACGTNVDBHWMRKSY]/g, '');
  this.bp = this.seq.length;
}

let x = new Sequence('CGTACAGTACTTTTGTGTTTACGAG');
```
...objects

may include methods (functions)

can compute from and/or change an object’s data

```javascript
function Sequence(s)
{
    this.seq = s.toUpperCase().replace(/[^ACGTNVDBHWMRKSY]/g, '');
    this.bp = this.seq.length;
    this.update = function(s)
    {
        this.seq = s.toUpperCase().replace(/[^ACGTNVDBHWMRKSY]/g, '');
        this.bp = this.seq.length;
    }
}

x.update('ACCCAGTCCATCTGGAAATCTTGGTTC');
```
for loops: the C way

for(let k = array.length - 1; k >= 0; k--){
    console.log(array[k]);
}

for loops: the JavaScript way

array.forEach(function(element){
  console.log(element);
  return(false);
});
+ concatenates strings
possible to chain multiple operations together

let x = y.toUpperCase().trim();
the original string is unchanged

capture in a variable or pass it to a function
regular expressions

relatively perl compatible

a RegExp object must be created (for most uses)

```javascript
let x = new RegExp('^a+rd');
if(x.test('aardvark') === true) {};
```

may be used directly in some instances

```javascript
let y = x.replace(/a/g, 'A');
```
async/await

allow slow tasks run in the background and avoid problems with asynchronous execution

forces JavaScript to wait for something to finish

```javascript
let x = await slowFunction();
async function slowFunction(){
    /* do something */
}
```
promises

allow slow tasks run in the background

avoid problems with asynchronous execution

forces JavaScript to wait for something to finish

let x = new Promise((resolve, reject) => {
    setTimeout(function(){
        resolve('done');
    }, 250);
});

x.then((m) => {
    console.log(m);
});
frameworks/toolkits...

Angular (Google; TypeScript)

MEAN (MongoDB, Express, Angular, and Node)

binds data sources (server) to client displays

designed to avoid DOM interactions

Dojo (TypeScript)

controllers/communications (dojo); database and data
binding (dstore); and display widgets (dijit, dojox, dgrid)

event framework for controlling user interaction

browser compatibility layer for DOM manipulation
...frameworks/toolkits...

Ember
controllers (routes); database and data binding (ember data); and display templates (HTMLBars)

jQuery
the most common web framework (but not trendy)
event framework for user interaction

browser compatibility layer for DOM manipulation
...frameworks/toolkits

React (Facebook)
  binds data sources (server) to client displays
  browser compatibility layer for (v)DOM manipulation
  JSX is used to create widgets

Vue (Google)
  a ‘reformed’ version of AngularJS
  binds data sources (server) to client displays
  designed to avoid DOM interactions
minification

reduces the size of JavaScript code (better performance)
removal of syntactic sugar
syntax checking; variable references and types
renaming of variables
removal of unneeded or unused code
closure
uglify.js (version 1, 2, or 3)
typescript (tsc)
linting

find programming errors (sometimes), style errors, and unusual constructs

Stephen Johnson (AT&T Bell Labs; 1978)

for C

good for large projects with multiple contributors

jshint or tslint
how to think like a programmer...

(1) determine what problem you are trying to solve
    think specific problem, general solution

(2) break it down into (very) small tasks

(3) write out the steps needed to accomplish each task
    think about writing instructions for a (very simpleminded and literal) person

(4) modify to match builtin functions and data structures

(5) convert steps to computer code

(6) be persistent
make pseudocode

think about what differentiates things

e.g. What makes GenBank accessions different from regular text? Position? Composition? Syntax?

make a minimal model

think: What does each step do?

think: Why it needs to be done?

think: What is the simplest way to do it?
example pseudocode...

convert DNA FASTA file to its reverse complement

(1) read FASTA file
(2) make reverse complement
(3) output new FASTA file
...example pseudocode...

(1) read FASTA file
    (a) get file name from user
    (b) open file
    (c) read line by line
    (d) differentiate between labels and sequence
    (e) store labels and sequence in RAM
...example pseudocode...

(1) read FASTA file
   (a) get file from user
       `<input type="file" id="input">`
       `document.getElementById('input').files[0]`
   (b) open file
       `FileReader.readAsText()`
   (c) read line by line
       `split(/
|\n|\n/); for()`
...example pseudocode...

(e) differentiate between labels and sequence
   > as first character versus
   ACGTNVDBHWMRKSYacgtnvdbhwmrksy
(i) accumulate multiple lines of sequence
   add to a temporary string
(ii) clean data
   replace()
...example pseudocode...

(f) store sequence in RAM

make reverse complement of complete temporary string and then output name and sequence (saves memory)
...example pseudocode

(2) make reverse complement
   .split('').reverse().join('')
   replace()
   (a) store in buffer
      buffer +=

(3) output new FASTA file
   domConstruct.create()
const antiDNA = new RegExp(/[^ACGTNVDBHWMRKSY]/g);
let d = 1;
let kb = false;
let l = '';
let m = 50;
let o = '';
let r = '';
let t = '';
```javascript
for(let k = process.argv.length-1; k >= 0; k--){
    if(process.argv[k] === '-d'){
        if(/^[0-5]{0,1}\.[0-9]{0,2}$/test(process.argv[k+1]) === true){
            d = process.argv[k+1];
        }
    } else if(process.argv[k] === '-k'){
        kb = true;
    } else if(process.argv[k] === '-l'){
        if(typeof(process.argv[k+1]) === 'string'){
            l = process.argv[k+1].toUpperCase().replace(antiDNA, '');
        }
    } else if(process.argv[k] === '-m'){
        if(/^[1-9]|1-9][0-9]|1-9][0-9]{2,2}|1-4][0-9]{3,3}|5000$/test(process.argv[k+1]) === true){
            m = process.argv[k+1];
        }
    } else if(process.argv[k] === '-r'){
        if(typeof(process.argv[k+1]) === 'string'){
            r = process.argv[k+1].toUpperCase().replace(antiDNA, '');
        }
    } else if(process.argv[k] === '-t'){
        if(typeof(process.argv[k+1]) === 'string'){
            t = process.argv[k+1].toUpperCase().replace(antiDNA, '');
        }
    }
}
```
if((l.length > 10) && (r.length > 10) && (t.length > 50)){
    o = '
A JavaScript for computing optimal PCR annealing temperatures using either
the algorithm of Rychlik et al. (1990) with Osborne’s (1992) corrections and
additional modifications for polymorphic sequences (thermodynamic constants
from Breslauer et al. [1986]) or the algorithm of Khandelwal
and Bhyravabhotla (2010) modified for polymorphic sequences using the Rychlik
and et al. (1990) combining formula [without Osborne’s (1992) corrections].

Usage:
-d\tOligo DNA concentration (pM; default = ' + d + 'pM).
-k\tUse the Khandelwal and Bhyravabhotla (2010) algorithm (default = ' + kb + ').
-l\tLeft oligo sequence (required).
-m\tMonovalent cation concentration (mM; required; default = ' + m + 'mM).
-r\tRight oligo sequence (required).
-t\tTemplate sequence (required).

References:
tduplex stability from the base sequence. Proceedings of the National\nAcademy of Sciences of the United States of America 83: 3746–3750.
Khandelwal, G. and J. Bhyravabhotla. 2010. A phenomenological model for
tpredicting melting temperatures of DNA sequences. PLOS ONE 5: e12433.
Research 18: 6409–6412.
}
process.stderr.write(o, 'UTF8');
function format(x, d){
    return (Math.round(x*d)/d);
}
function oligoScore(oligo, matrix){ /* modified to work with polymorphisms */
    function base2number(base){
        return base.replace(/R/,'02').replace(/Y/,'13').replace(/M/,'01').replace(/K/,'23').replace(/S/,'12').replace(/W/,'03').replace(/H/,'013').replace(/B/,'123').replace(/V/,'012').replace(/D/,'023').replace(/N/,'0123').split('');
    }
    const bases = oligo.replace(/A/g,'0').replace(/C/g,'1').replace(/G/g,'2').replace(/T/g,'3').split('');
    let score = 0;
    const num = new RegExp(/^0|1|2|3$/);
    for(let k = bases.length-1; k > 0; k--){
        if((num.test(bases[k-1]) === true) && (num.test(bases[k]) === true)){
            score += matrix[bases[k-1]][bases[k]];
        } else {
            const x = base2number(bases[k-1]);
            const y = base2number(bases[k]);
            let sum = 0;
            for(j = x.length-1; j >= 0; j--){
                for(i = y.length-1; i >= 0; i--){
                    sum += matrix[x[j]][y[i]];
                }
            }
            score += sum/(x.length*y.length);
        }
    }
}
return(score);
function KhandelwalBhyravabhotla2010(oligo) {
    return (7.35 * (oligoScore(oligo, [
        [5, 10, 8, 7], /* AA, AC, AG, AT */
        [7, 11, 10, 8], /* CA, CC, CG, CT */
        [8, 13, 11, 10], /* GA, GC, GG, GT */
        [4, 8, 7, 5] /* TA, TC, TG, TT */
    ]) / oligo.length) + 17.34 * Math.log(oligo.length) + 4.96 * Math.log(m / 1000) + 0.89 * Math.log(d / 1000000000000) - 25.42;
}
function RychlikEtAl1990(oligo){
    const dH = oligoScore(oligo, [
        [9.1,6.5,7.8,8.6], /* AA,AC,AG,AT */
        [5.8,11.0,11.9,7.8], /* CA,CC,CG,CT */
        [5.6,11.1,11.0,6.5], /* GA,Gc,GG,GT */
        [6.0,5.6,5.8,9.1] / * TA,TC,TG,TT */
    ]);)

    const dS = oligoScore(oligo, [
        [24.0,17.3,20.8,23.9], /* AA,AC,AG,AT */
        [12.9,26.6,27.8,20.8], /* CA,CC,CG,CT */
        [13.5,26.7,26.6,17.3], /* GA,Gc,GG,GT */
        [16.9,13.5,12.9,24.0] / * TA,TC,TG,TT */
    ]);)

    return((-1000*dH)/(-1*dS - 57.48) - 273.15 + 16.6*Math.log10(m/1000));
}
let lTm = RychlikEtAl1990(l);
if (kb === true) {
  lTm = KhandelwalBhyravabhotla2010(l);
}
o += 'left oligo Tm = ' + format(lTm, 10) + °C ' + l + 'n';

let rTm = RychlikEtAl1990(r);
if (kb === true) {
  rTm = KhandelwalBhyravabhotla2010(r);
}
o += 'right oligo Tm = ' + format(rTm, 10) + °C ' + r + 'n';
let tGC = (t.match(/G/g) || []).length
  + (t.match(/C/g) || []).length
  + (t.match(/S/g) || []).length
  + (1/2)*
    (t.match(/K/g) || []).length
  + (t.match(/M/g) || []).length
  + (t.match(/N/g) || []).length
  + (t.match(/R/g) || []).length
  + (t.match(/Y/g) || []).length
)
+ (1/3)*
  (t.match(/D/g) || []).length
+ (t.match(/H/g) || []).length
)
+ (2/3)*
  (t.match(/B/g) || []).length
+ (t.match(/V/g) || []).length
);

tGC /= t.length;
let tTm = 41*tGC + 16.6*Math.log(m) - 675/t.length; /* log base e contra RychlikEtAl1990 */
o += 'template Tm = ' + format(tTm, 10) + '°C
';
let Tm = lTm;
if (lTm > rTm) {
    Tm = rTm;
}
o += 'optimal combined Tm = ';
if (kb === true) {
o += format((0.3*(Tm) + 0.7*tTm - 14.9), 10);
} else {
o += format((0.3*(Tm+14) + 0.7*tTm - 22.9), 10);
}
o += '°C

process.stdout.write(o, 'UTF8');