Excursions in Computing Science: Week 9. Many Dimensions: Data Compression and Content

T. H. Merrett*
McGill University, Montreal, Canada

June 10, 2015

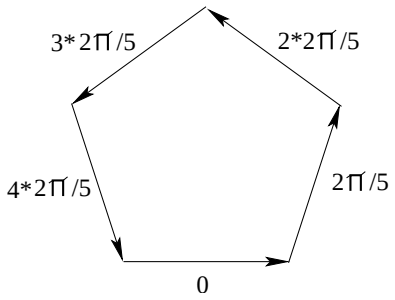
I. Prefatory Notes

1. With two-dimensional numbers, we can make a vector space of any number of dimensions.

Let's try 5 dimensions.

Here's a clue.

$$\sum_{k=0}^{4} e^{ik2\pi/5} = 0$$



Let's try vectors $F^5_{j\,k} = \frac{1}{\sqrt{5}} e^{ijk2\pi/5}$

jk	$k=$	0	1	2	3	4	
	0	0	0	0	0	0	$F^5_{0\,k}$
	1	0	1	2	3	4	$F^5_{1\,k}$
$j=$	2	0	2	4	1	3	$F^5_{2\,k}$
	3	0	3	1	4	2	$F^5_{3\,k}$
	4	0	4	3	2	1	$F^5_{4\,k}$
		(F_0)	(F_1)	(F_2)	(F_3)	(F_4)	F_k

 The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students and their funding agencies.

Note 1. It's jk mod 5, e.g., 3*2 mod 5 = 6 mod 5 = 1

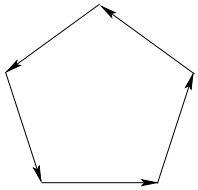
Note 2. $(F_k)_j = F^5_{j\,k} = \frac{1}{\sqrt{5}} e^{ijk\ 2\pi/5}$, not just jk.

If we use the conjugation operator, *: $i \to -i$, these vectors are normalized:

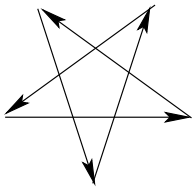
$$\begin{aligned}
F_0^*.F_0 &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ij0\ 2\pi/5} \frac{1}{\sqrt{5}} e^{ij0\ 2\pi/5} = \frac{1}{5}\sum_{j=0}^{4} 1 = 1 \\
F_1^*.F_1 &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ij1\ 2\pi/5} \frac{1}{\sqrt{5}} e^{ij1\ 2\pi/5} = \frac{1}{5}\sum_{j=0}^{4} e^{i(j-j)1\ 2\pi/5} = \frac{1}{5}\sum_{j=0}^{4} 1 = 1 \\
F_k^*.F_k &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ijk\ 2\pi/5} \frac{1}{\sqrt{5}} e^{ijk\ 2\pi/5} = \frac{1}{5}\sum_{j=0}^{4} e^{i(j-j)k2\pi/5} = \frac{1}{5}\sum_{j=0}^{4} 1 = 1
\end{aligned}$$

They are orthogonal:

$$\begin{aligned}
F_0^*.F_1 &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ij0\ 2\pi/5} \frac{1}{\sqrt{5}} e^{ij1\ 2\pi/5} \\
&= \frac{1}{5}\sum_{j=0}^{4} e^{ij(1-0)2\pi/5} = 0
\end{aligned}$$



$$\begin{aligned}
F_0^*.F_2 &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ij0\ 2\pi/5} \frac{1}{\sqrt{5}} e^{ij2\ 2\pi/5} \\
&= \frac{1}{5}\sum_{j=0}^{4} e^{ij(2-0)2\pi/5} = 0
\end{aligned}$$



Thus they are orthonormal:

$$\begin{aligned}
F_l^*.F_k &= \sum_{j=0}^{4} \frac{1}{\sqrt{5}} e^{-ijl2\pi/5} \frac{1}{\sqrt{5}} e^{ijk2\pi/5} \\
&= \frac{1}{5}\sum_{j=0}^{4} e^{ij(k-l)2\pi/5} \\
&= \delta_{l\,k} \stackrel{\text{def}}{=} \text{if } l{=}k \text{ then 1 else 0}
\end{aligned}$$

It's a 5-dimensional space.

And we can do this for any n: use a regular n-gon and its stars instead of the pentagon. What's important is that they are closed figures.

So we have n-dimensional space.

2. Is it good for anything?

It's actually a matrix (a row of column vectors): it's a transformation, just like rotation,

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix}:$$

$$(c,-s)\begin{pmatrix} c \\ -s \end{pmatrix}=1\,;\;\; (s,c)\begin{pmatrix} s \\ c \end{pmatrix}=1\,;\;\; (c,-s)\begin{pmatrix} s \\ c \end{pmatrix}=0\,;\;\; (s,c)\begin{pmatrix} c \\ -s \end{pmatrix}=0$$

or shear,

$$\gamma\begin{pmatrix} 1 & -v \\ -v & 1 \end{pmatrix}:$$

$$(1,-v)\begin{pmatrix} 1 \\ -v \end{pmatrix}=1/\gamma^2\,;\;\; (-v,1)\begin{pmatrix} -v \\ 1 \end{pmatrix}=1/\gamma^2\,;\;\; (1,-v)\begin{pmatrix} -v \\ 1 \end{pmatrix}=0\,;\;\; (-v,1)\begin{pmatrix} 1 \\ -v \end{pmatrix}=0$$

So we can transform vectors in 5-dimensional space.

$$\vec{f'}=F\vec{f}\,;\quad \vec{f}=F^{*T}\vec{f'}$$

It's called the Fourier transform (more precisely, DFT, the discrete Fourier transform).

What could these vectors be? Let's try functions.

E.g., $f(x)=x^2$

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}=\begin{pmatrix} 0 \\ 1 \\ 4 \\ 9 \\ 16 \end{pmatrix}$$

MATLAB:

```
E5 = [0,0,0,0,0;
      0,1,2,3,4;
      0,2,4,1,3;
      0,3,1,4,2;
      0,4,3,2,1];
F5 = exp((i*2*pi/5).*E5)./sqrt(5);
parabPlus = [0;1;4;9;16];
parabPlFT = F5*parabPlus

parabPlFT =

  13.4164
  -2.3541 - 7.6942i
  -4.3541 - 1.8164i
  -4.3541 + 1.8164i
  -2.3541 + 7.6942i
```

(Note that $13.4164/\sqrt{5}=6$, the average value of `parabPlus`. For this reason, the Fourier transform is usually defined as $F/\sqrt{5}$ and the inverse as $\sqrt{5}F^{*T}$. But we'll stick to our symmetric definition.)

What is this giving us?

It's not clear for x^2, but let's try $\cos(x)$

MATLAB:

```
function Cn = cosine(n,f)   % f is "frequency"
for j=1:n, Cn(j) = cos(2*pi*(j-1)*f/n); end;

F5*cosine(5,1)'

ans =

  -0.0000
   1.1180 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   1.1180 - 0.0000i

F5*cosine(5,2)'

ans =

  -0.0000
   0.0000 + 0.0000i
   1.1180 + 0.0000i
   1.1180 - 0.0000i
  -0.0000 + 0.0000i
```

When frequency is 1, component 1 is nonzero.
When frequency is 2, component 2 is nonzero.

The Fourier transform is picking out the *frequencies*.

3. So if we have a function in which some frequencies are much more important than others, we could take the Fourier transform, store or transmit only the important frequencies, then when we want the function back, take the inverse Fourier transform of these only, to get a (good?) approximation.

Let's try it for `cosine`

MATLAB:

```
cosine(5,1)

ans =

    1.0000    0.3090   -0.8090   -0.8090    0.3090

cosFTapprox = [0;2.236;0;0;0];
conj(F5)*cosFTapprox

ans =

   1.0000
   0.3090 - 0.9510i
  -0.8090 - 0.5878i
  -0.8090 + 0.5878i
```

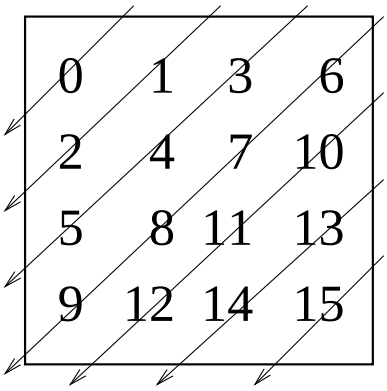
```
0.3090 + 0.9510i
```

The real part is exact.

In the case that only a few frequencies are important, F gives us *data compression.*

4. This is used by JPEG (Joint Photographic Experts Group) to compress images along something like the following lines.

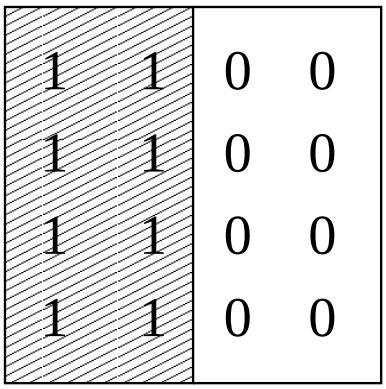
1. Scan the picture (or 8×8 subpictures of it) diagonally,
   e.g. (4×4):



2. Fourier transform (e.g., `F16`).

3. Set the unimportant frequencies to zero (i.e., keep only the important frequencies).

Let's try it on the 4×4 black and white pattern



MATLAB:

```
F16 = makeDFT(16);
leftBlack = [1,1,0,0;
             1,1,0,0;
             1,1,0,0;
             1,1,0,0];
```

```
diag = makeDiag(4,leftBlack)
    1   1   1   0   1   1   0   0   1   1   0   0   1   0   0   0
leftBlFT = F16*diag'           conj(leftBlFT).*leftBlFT
   1    2.0000                              4.0000
   2    0.0811 + 0.4077i                    0.1728
   3    0.1768 + 0.4268i                    0.2134
   4    0.0542 + 0.0811i                    0.0095
   5    0.7500 + 0.7500i                    1.1250
   6   -0.4077 - 0.2724i                    0.2405
   7   -0.1768 - 0.0732i                    0.0366
   8    0.2724 + 0.0542i                    0.0772
   9    0.5000 + 0.0000i                    0.2500
  10    0.2724 - 0.0542i                    0.0772
  11   -0.1768 + 0.0732i                    0.0366
  12   -0.4077 + 0.2724i                    0.2405
  13    0.7500 - 0.7500i                    1.1250
  14    0.0542 - 0.0811i                    0.0095
  15    0.1768 - 0.4268i                    0.2134
  16    0.0811 - 0.4077i                    0.1728
leftBlFTAx = zeros(size(leftBlFT));
leftBlFTAx(1) = leftBlFT(1);        % try only frequency component 1
diagAx = conj(F16)*leftBlFTAx;
leftBlAx1 = binarize(unmakeDiag(4,diagAx'));
      1     1     1     1
      1     1     1     1
      1     1     1     1
      1     1     1     1
leftBlFTAx(5) = leftBlFT(5);        % now add frequency component 5
diagAx = conj(F16)*leftBlFTAx;
leftBlAx5 = binarize(unmakeDiag(4,diagAx'));
      1     1     0     0
      0     1     0     0
      1     1     0     1
      1     1     0     0
```

To store this approximation, instead of 16 numbers, keep only 6:

$$\begin{array}{ccc} 1 & 2 & 0 \\ 5 & 0.75 & 0.75 \end{array}$$

(To get it perfect, we must also include components 9, 6 and 3.)

5. The fast Fourier transform

So far, we've been using at least n^2 operations to calculate F^n:

$$f'_k = \sum_{j=-0}^{n-1} F^n_{k\,j} f_j$$

But F^n has special properties which may speed this up.

$$F^n_{k\,j} = e^{\frac{2\pi i}{n} kj} = \omega^{kj} = (\omega^k)^j$$

So

$$f'_k = \sum_{j=-0}^{n-1} f_j * (\omega^k)^j = f_0 * (\omega^k)^0 + f_1 * (\omega^k)^1 + ... + f_{n-1} * (\omega^k)^{n-1}$$

It's a *polynomial* in ω^k.

And it can be evaluated by *Horner's rule*

$$f'_k = f_0 + \omega^k * (f_1 + \omega^k * (f_2 + ... + \omega^k * f_{n-1}...))$$

Now this is still n operations for each $k = 0, .., n-1$:
$n^2(+, *)$ to be precise.
But that's all the operations and we don't have to work out $e^{\frac{2\pi i}{n}}$ every time.

Still, we can do better.

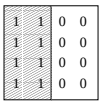
$f'_k = P(\omega^k)$ for a polynomial P.

And any $P(x)$ can be evaluated at $x = a$ by finding a *remainder*

$$P(a) = P(x) \text{mod} (x - a)$$

This is *polynomial arithmetic*: polynomials obey the same axioms as integers—polynomials form a *ring* (see Week 4)—and so we can do long division and get *quotients* and *remainders* (among other things, such as addition, subtraction and multiplication).

Let's explore this for `F16` and



$$\begin{aligned}
f'_k &= \sum_{j=-0}^{n-1} (\omega^k)^j f_j \\
(f_j) &= (\texttt{1};\texttt{1};\texttt{1};\texttt{0};\texttt{1};\texttt{1};\texttt{0};\texttt{0};\texttt{1};\texttt{1};\texttt{0};\texttt{0};\texttt{1};\texttt{0};\texttt{0};\texttt{0}) \\
f'_k &= f'(\omega^k) \text{ where} \\
4f'(x) &= x^0 + x^1 + x^2 + x^4 + x^5 + x^8 + x^9 + x^{12}
\end{aligned}$$

Let's practice polynomial long division with two examples before we try to reduce the n^2 operations for Fourier to fewer.

$$\begin{aligned}
4f'(x) &= q_1(x)d_1(x) + r_1(x) \\
4f'(x) &= q_2(x)d_2(x) + r_2(x)
\end{aligned}$$

with $d_1(x) = x^8 - \omega^0 = x^8 - 1$
and $d_2(x) = x^8 + \omega^0 = x^8 + 1$

1.

$$
\begin{array}{r}
x^4+x+1 \\
\hline
x^8-1\,)\,x^{12}+x^9+x^8+x^5+x^4+x^2+x+1 \\
\quad x^{12}\qquad\qquad\qquad -x^4 \\
\qquad\quad x^9+x^8+x^5+2x^4+x^2+x+1 \\
\qquad\quad x^9\qquad\qquad\qquad\quad -x \\
\qquad\qquad\quad x^8+x^5+2x^4+x^2+2x+1 \\
\qquad\qquad\quad x^8\qquad\qquad\qquad\quad -1 \\
\qquad\qquad\qquad\quad x^5+2x^4+x^2+2x+2
\end{array}
$$

So

$$
\begin{aligned}
q_1(x) &= x^4+x+1 \\
r_1(x) &= x^5+2x^4+x^2+2x+2
\end{aligned}
$$

2.

$$
\begin{array}{r}
x^4+x+1 \\
\hline
x^8+1\,)\,x^{12}+x^9+x^8+x^5+x^4+x^2+x+1 \\
\qquad\quad x^9+x^8+x^5\qquad +x^2+x+1 \\
\qquad\qquad\quad x^8+x^5\qquad +x^2\qquad +1 \\
\qquad\qquad\qquad\quad x^5\qquad +x^2
\end{array}
$$

$$
\begin{aligned}
q_1(x) &= x^4+x+1 \\
r_1(x) &= x^5+x^2
\end{aligned}
$$

So we know that we can find $f'(x_1) = r_1(x_1)$ for any x_1 for which $d_1(x_1) = 0$ and $f'(x_2) = r_2(x_2)$ for x_2 making $d_2(x_2) = 0$.

That should save some work for x_1 and x_2.
Can we use this? Can we do it for any ω^k? All ω^k?

Well, here's a trick.

$$
\begin{aligned}
d_1(x) &= (x-\omega^0)(x-\omega^8)(x-\omega^4)(x-\omega^{12})(x-\omega^2)(x-\omega^{10})(x-\omega^6)(x-\omega^{14}) \\
&= (x^2+\omega^8)\qquad (x^2+\omega^0)\qquad (x^2+\omega^{12})\qquad (x^2+\omega^4) \\
&= (x^4+\omega^8)\qquad\qquad\qquad (x^4+\omega^0) \\
&= x^8+\omega^8 \\
&= x^8-\omega^0 \\
&= x^8-1
\end{aligned}
$$

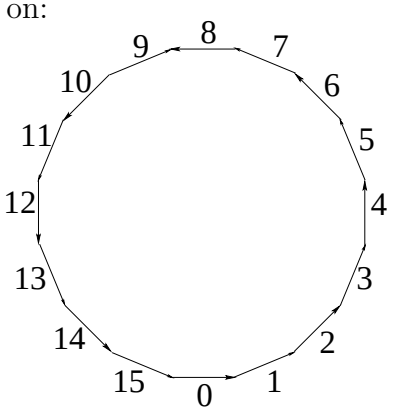
and

$$
\begin{aligned}
d_2(x) &= (x-\omega^1)(x-\omega^9)(x-\omega^5)(x-\omega^{13})(x-\omega^3)(x-\omega^{11})(x-\omega^7)(x-\omega^{15}) \\
&= (x^2+\omega^{10})\qquad (x^2+\omega^2)\qquad (x^2+\omega^{14})\qquad (x^2+\omega^6) \\
&= (x^4+\omega^{12})\qquad\qquad\qquad (x^4+\omega^4 \\
&= x^8+\omega^0 \\
&= x^8+1
\end{aligned}
$$

Why?

$$\begin{aligned}(x-\omega^0)(x-\omega^8) &= x^2-(\omega^0+\omega^8)x+\omega^8\\ (x-\omega^4)(x-\omega^{12}) &= x^2-(\omega^4+\omega^{12})x+\omega^{16}\\ (x^2-\omega^8)(x^2-\omega^0) &= x^4-(\omega^8+\omega^0)x^2+\omega^8\end{aligned}$$

and so on:

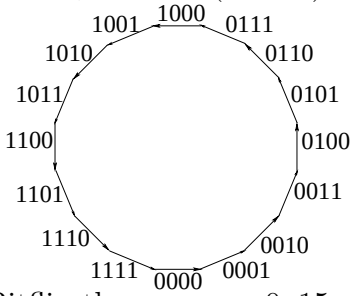


$\omega^8 = -\omega^0 \qquad \omega^0+\omega^8 = 0$

$\omega^{12} = -\omega^4 \qquad \omega^4+\omega^{12} = 0$

How did we pick 0, 8, 4, 12, 2, 10, 6, 14 for d_1? Pairs of opposites!
Similarly d_2: 1, 9, 5, 13, 3, 11, 7, 15

In fact, there's a(nother) trick:



0	8	4	12	2	10	6	14
0000	1000	0100	1100	0010	1010	0110	1110
0000	0001	0010	0011	0100	0101	0110	0111
0	1	2	3	4	5	6	7

1	9	5	13	3	11	7	15
0001	1001	0101	1101	0011	1011	0111	1111
1000	1001	1010	1011	1100	1101	1110	1111
8	9	10	11	12	13	14	15

Bitflip the sequence 0..15 and get pairs of opposites.

So we can find $f'(x_1)$ for $x_1 = \omega^0, \omega^8, \omega^4, \omega^{12}, \omega^2, \omega^{10}, \omega^6, \omega^{14}$ by finding $r_1(x_1)$
and we can find $f'(x_2)$ for $x_2 = \omega^1, \omega^9, \omega^5, \omega^{13}, \omega^3, \omega^{11}, \omega^7, \omega^{15}$ by finding $r_2(x_2)$

Is this less work than directly finding $f'(x_1)$ and $f'(x_2)$?

Each long division costs at most 8 subtractions of a 2-term polynomial: 8*2(+,*)
Total 2*8*2(+,*)

Then solving the remainder, each at most 8 terms, is another 8(+,*)

6. **Divide and Conquer**

But we can do the same thing with the remainders:

$$\begin{aligned}d_{11}(x) &= (x-\omega^0)(x-\omega^8)(x-\omega^4)(x-\omega^{12}) = x^4+\omega^8 = x^4-\omega^0 = x^4-1\\ d_{12}(x) &= (x-\omega^2)(x-\omega^{10})(x-\omega^6)(x-\omega^{14}) = x^4+\omega^0 = x^4+1\\ d_{21}(x) &= (x-\omega^1)(x-\omega^9)(x-\omega^5)(x-\omega^{13}) = x^4+\omega^{12} = x^4-\omega^4 = x^4-i\\ d_{22}(x) &= (x-\omega^3)(x-\omega^{11})(x-\omega^7)(x-\omega^{15}) = x^4+\omega^4 = x^4+i\end{aligned}$$

$$
\begin{array}{lll}
 & & x+2 \\
r_1(x) \div d_{11}(x) & = x^4+1) & \overline{x^5+2x^4+x^2+2x+2} \\
 & & \quad 2x^4+x^2+3x+2 \\
 & & \quad\quad x^2+3x+4 \quad = r_{11}(x) \\
 & & x+2 \\
r_1(x) \div d_{12}(x) & = x^4-1) & \overline{x^5+2x^4+x^2+2x+2} \\
 & & \quad 2x^4+x^2+x+2 \\
 & & \quad\quad x^2+x \quad = r_{12}(x) \\
 & & x \\
r_2(x) \div d_{21}(x) & = x^4-i) & \overline{x^5+x^2} \\
 & & \quad\quad x^2+ix \quad = r_{21}(x) \\
 & & x \\
r_2(x) \div d_{22}(x) & = x^4+i) & \overline{x^5+x^2} \\
 & & \quad\quad x^2-ix \quad = r_{22}(x)
\end{array}
$$

This has cost at most 4*4*2(+,*).

Doing it once more will give 8 linear remainders of the form $ax+b$,
and a final time will give 16 constant remainders, the final values of the $4f'(\omega^k)$s

That is, 4 iterations gives $4f'(\omega^k)$ for all $\omega^k, k = 0, .., 15$
Now there are no remainders left to evaluate. We need only the divisions.

Cost:
2*8*2(+,*) + 4*4*2(+,*) + 8*2*2(+,*) + 16*1*2(+,*) = 4*16*2(+,*)

versus cost for the Horner's rule evaluation 16 times:
16*16(+,*)

The divide-and-conquer takes half the cost.

If $n = 32$: 5*32*2 versus 32*32: one third the cost.

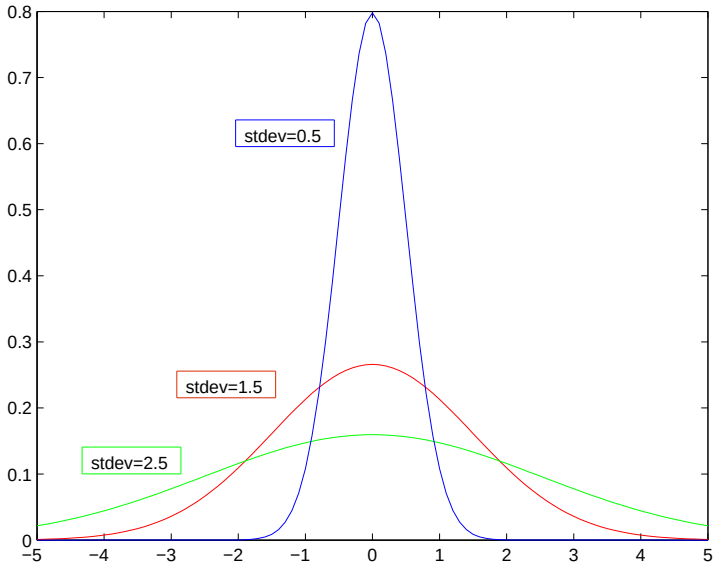
In general, $2n \lg n$ versus n^2.

Divide and conquer is frequently used to reduce n^2 algorithms to $n \log n$: for example, sorting, Voronoi diagrams.

7. The Uncertainty Principle

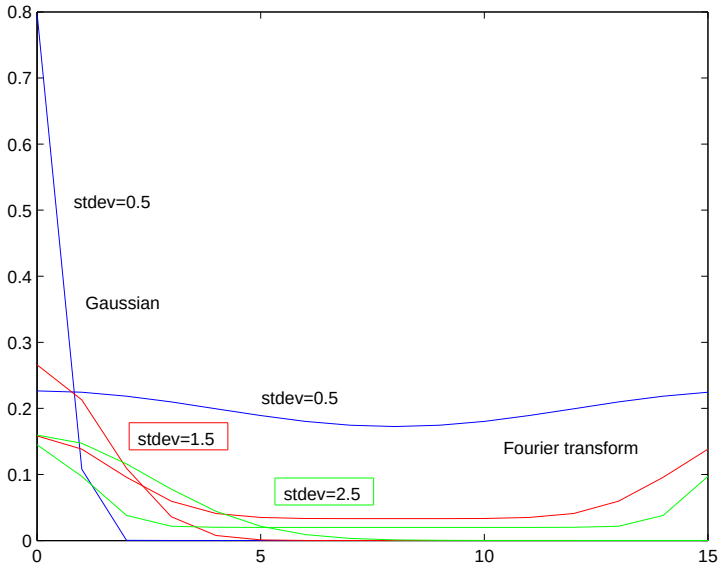
What function is the Fourier transform of itself?

Try Gauss' bell curve, the *gaussian*, $\frac{e^{-x^2/2\delta^2}}{\sqrt{2\pi\delta^2}}$,
centred at 0 with "width" or standard deviation δ.



MATLAB:

```
stdev = ..; plotcolour = '..';
g16 = gaussian(16,stdev);
plot(0:15,g16,plotcolour)
hold on
plot(0:15,F16*g16',plotcolour)          % will ignore imaginary parts
```



The Fourier transform of the gaussian is close to itself at `stdev`=1.5

More important, the fatter the gaussian, the thinner the Fourier transform, and vice-versa.

The Fourier transform gives "frequencies".
If we interpret the gaussian as the (uncertain) *position* of a quantum particle,
then the Fourier transform gives its (uncertain) frequency.
But frequency together with velocity gives *momentum*

The width of the bell shape is uncertainty:
a very narrow gaussian gives a very precise measure of position;
a very narrow Fourier transform gives a very precise measure of momentum.

But we can't have *both*: the QM "uncertainty principle".

8. **Compression and Content**

We've seen how we can compress data by throwing away the unimportant frequencies of the Fourier transform before transforming back again.

(Throwing away $\longrightarrow$ "lossy" compression.)

There is a sense in which compression is equivalent to knowing the content.
A theory can be seen as a compressed representation.
For instance, $\gamma \begin{pmatrix} 1 & -v \\ -v & 1 \end{pmatrix}$, the Lorentz transform, captures an extensive understanding of time and space, and that speed does not change the laws of physics.
Or $\mathbf{e}_1.\mathbf{e}_2 = -\mathbf{e}_2.\mathbf{e}_1$, the axiom of Clifford algebra, captures an understanding of surfaces in spaces.

These are all much more compact than all the particular facts that can be deduced from them.

So when we compress data by focussing on the most important part, in this case of the Fourier transform (but there are many other possibilities), we are making a theory about the content of the data.

The Fourier transform is used for content analysis of multimedia including pictures and time series (even stock prices).

One of the most common questions about content involve similarities: "how similar are these two pictures?", or "find stock price series that are behaving like this one here". If two things have the same frequency spectra, revealed by Fourier analysis, this one way in which they are similar.

9. **Summary**

(These notes show the trees. Try to see the forest!)

- Orthonormal basis "vectors" for abstract spaces of any number of dimensions.
- Functions are vectors.
- Fourier transforms extract "frequencies" from functions.
- Throwing away unimportant frequencies gives (lossy) data compression.

- Divide-and-conquer reduces, e.g., $O(n^2)$ algorithms to $O(n \log n)$:
  fast Fourier transform uses polynomial remainders and opposing sides of the regular 2^n-gon to accomplish this.

- Fourier transform of Gauss' bell curve gives back a bell curve, but fat $\longleftrightarrow$ thin:
  if the two bells are errors of measuring postions and momenta of particles, these measurements are "complementary" in that both cannot be made exactly.

- Data compression $\equiv$ theory about data $\equiv$ content analysis

II. The Excursions

You've seen lots of ideas. Now *do* something with them!

1. Check that the 5-dimensional basis vectors, F_k, are orthonormal:
   $F_l^*.F_k =$ if $l = k$ then 1 else 0, for $l, k = 0 : 4$
   Draw all the closed figures (pentagon, star, etc.) that show orthogonality.

2. Draw the closed figures that show that similar vectors, F_k, are orthogonal in 6 dimensions.

3. If we take only the "real" parts of the two-dimensional numbers used in F_k (that is, the projections on the one-dimensional number line, meaning use $\cos(x)$ instead of $\exp(ix)$) do we get the same orthonormal relationships? What if we take the "imaginary" parts ($\sin(x)$)?

4. Use MATLAB to Fourier-transform some functions other than x^2 and $\cos(x)$ from 0 to 4.

5. Explain the fourth component of the Fourier transform of cosine when the frequency is 1, and the third component when the frequency is 2.

6. The MATLAB code shown in the notes to invert the Fourier transform uses only `conj(F16)` and not the transpose, `conj(F16')`. Why is it still correct?

7. Diagonalize the approximation

   ```
   1100
   0100
   1101
   1100
   ```

   to the half-black image and show that it has the period expected from the frequency components used in the approximation.

8. Take the Fourier transform of the checkerboard image

   ```
   1100
   1100
   0011
   0011
   ```

approximate it by setting unimportant frequencies to zero, and transform back again to the approximate image. How many Fourier terms did you keep to get a reasonable approximation? An exact reproduction?

9. Show that the following vectors are orthonormal and so also provide a basis for functions in 5 dimensions. Why could they be called the "bar chart basis"? They give the the idea for what the computing community calls "wavelets" and the physics community calls "Green's functions" [FLS64, p. 25-4]: look these up!

```
1  0  0  0  0
0  1  0  0  0
0  0  1  0  0
0  0  0  1  0
0  0  0  0  1
```

10. Why is $p(a) = p(x) \bmod (x - a)$ for polynomial p? (Write p in terms of $x - a$ and its quotient and remainder after division.)

11. Do the two last steps of the divide-and-conquer not done in the notes, and check your result against `leftBlFT` from the notes.

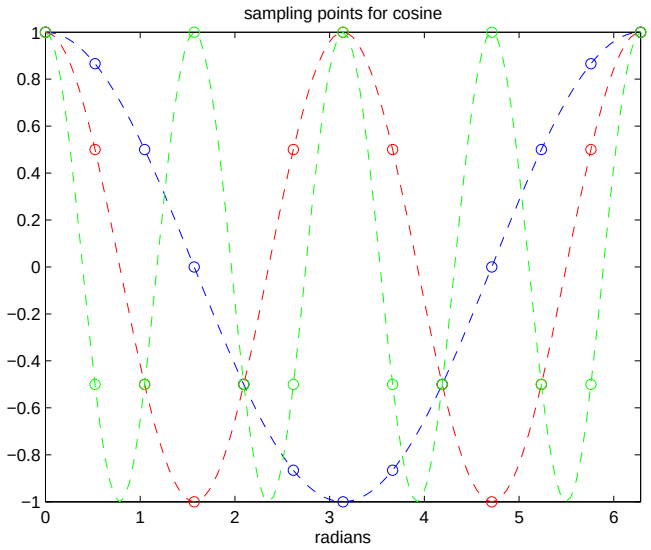
12. The notes give the worst-case costs for a 16-dimensional fast Fourier transform. How much did the half-black image example actually cost?

13. Use tic and toc in MATLAB to compare the $O(n^2)$ Fourier transform, coded at the beginning of the notes, with MATLAB's `fft()` function.

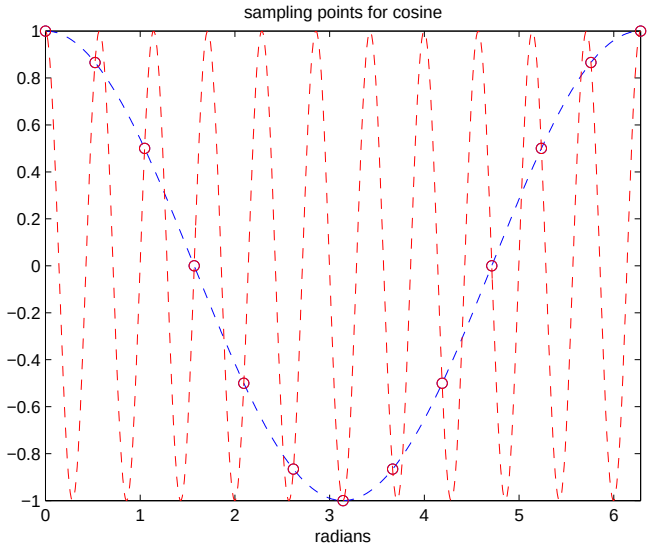
14. Outline the algorithms for "bubble sort" and for "merge sort", which are different ways of sorting sets of records (or numbers, or words). What are the costs of these algorithms, approximately, as functions of n, the number of records to be sorted? Comment on the divide-and-conquer aspects. Is there another sorting algorithm which illustrates divide-and-conquer in a different way? (What is the logarithm of the total number of permutations n things can form, and why is this related to the cost of sorting?)

15. Rewrite the Fourier transform code so that it symmetrically gives positive and "negative" frequencies. (E.g., frequencies 0,1,2 → 0,1,-1 → -1,0,1) Use this to give Fourier transforms, of the gaussian bell curve, which are themselves symmetrical bell curves.

16. **Artefacts of sampling.** a) Write a MATLAB program, `sampling(freqs,rate)` which plots cosine for harmonics of a fundamental frequency plotted from 0 to 2π radians and sampled `rate` times. Thus `sampling([1,2,4],12)` should yield the following plot.



Note that the smooth cosines are plotted and that the 12 samples less than 2π (13 including 2π) are highlighted with circles.
b) Run your program as `sampling([1,11],12)` to get



Explore this for different frequencies, including those greater than 12. Why must the sampling rate be at least twice the highest frequency one needs to detect?
c) Modify your program to interpret negative frequencies by their absolute values but with the phase altered by π. Explore the sampling now.
d) Discuss the behaviour of the discrete Fourier transform in the light of your explorations of sampling.

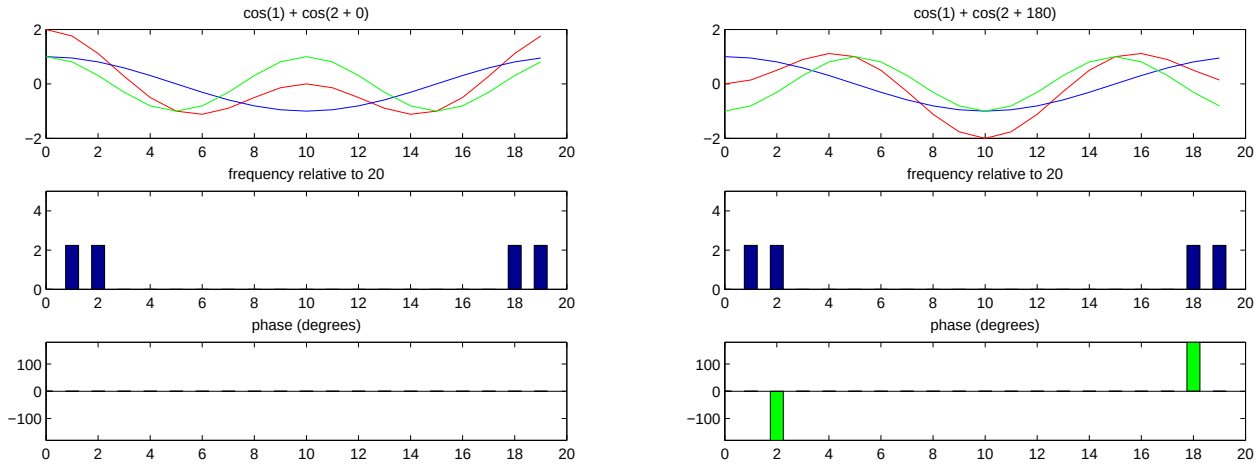
17. Why are the Fourier transforms symmetric in the way shown by the examples in the notes, if negative frequencies are not used? What do the last component of `F5*cosine(5,1)'` and the next-to-last component of `F5*cosine(5,2)'` mean? The

components near frequency 15 of the 16-point gaussians? If we remove these terms and Fourier transform back again, what is the result?

18. Two aspects of the Fourier transform should be a little puzzling. The first is that the results are repeated beyond the mid-way point, apart from the change of sign in the "imaginary" part. The second is the imaginary part itself—the results are 2-dimensional numbers.
We have addressed the repetition in the previous excursions.
Note that the total number of different values is thus the same for the Fourier transform as for the input.
We have used the *magnitudes* of the resulting 2-numbers but not their *angles*. These can be interpreted as the relative *phases* of the waves making up the original data.
For example, we could generate a signal of two waves of different frequency with one not in phase with the other.

```
%function Cn = cos2freqNphas(n,f1,f2,p12)                THM              110329
% generates n-point vector from sum of cos(f1) + cos(f2+p12)
% frequencies f1, f2 are relative to n: one period shown when f = 1
% relative phase p12 in degrees
function Cn = cos2freqNphas(n,f1,f2,p12)
phase = p12*pi/180;     % radians
for j=1:n
  C1(j) = cos(2*pi*(j-1)*f1/n);
  C2(j) = cos(2*pi*(j-1)*f2/n + phase);
end %for j%
Cn = C1 + C2;
```

Use `makeDFT(n)` to find the Fourier transform of this sum of waves and plot the wave, its components, and the magnitude (frequency) and angle (phase) given by the DFT. Here are results for a wave (red) with components of frequencies 1 (blue) and 2 (green) a) in phase and b) 180 degrees out of phase with each other.



19. **Two-dimensional discrete Fourier transform** The two-dimensional waves of Note 22 in Book 8c (part II) can be used in a two-dimensional Fourier transform

$$\mathcal{F}_2 = \frac{1}{3} e^{2\pi i(j_x k_x + j_y k_y)/3}$$

where (j_x, j_y) runs over the 3×3 points of the crystal lattice and (k_x, k_y) runs over the 3×3 points of the reciprocal lattice. $j_x k_x + j_y k_y$ takes on the following values (shown modulo 3)

$k_x k_y =$ $j_x j_y$	00	10	20	01	11	21	02	12	22
00	0	0	0	0	0	0	0	0	0
10	0	1	2	0	1	2	0	1	2
20	0	2	1	0	2	1	0	2	1
01	0	0	0	1	1	1	2	2	2
11	0	1	2	1	2	0	2	0	1
21	0	2	1	1	0	2	2	1	0
02	0	0	0	2	2	2	1	1	1
12	0	1	2	2	0	1	1	2	0
22	0	2	1	2	1	0	1	0	2

Confirm this and compare with the "one-dimensional" Fourier transform of Note 1. Show that the conjugate ($i \to -i$) of the transpose ($jk \to kj$) of $\mathcal{F}$ is its inverse.

20. **DSP filters.** The Fourier transform is heavily used in digital signal processing. Here is an excursion.
a) A sequence of values (a "signal") can be *filter*ed into new sequences (signals). A running average is an example: fix a "window" size, w, and find the average of the first w values. Then shift by one position and find the average of the next w values (overlapping with the first by $w-1$). Shift again, and so on. Such a filter can be used to "smooth" the sequence, effectively by reducing the highest frequencies.
To save arithmetic but cover the same ideas, let's take running *totals*. Here is a sequence and its running total for a window size 3.

2	3	2	3	3	1	2	2	3	1	0	2
2	5	7	8	8	7	6	5	7	6	4	3

Note that the total starts with the smaller window sizes 1 and 2 to get started until the first three elements have been processed.
The formula we could use to describe this is

$$y_n = x_n + x_{n-1} + x_{n-2}$$

or, more generally,

$$y_n = b_1 x_n + b_2 x_{n-1} + b_3 x_{n-2} + ..$$

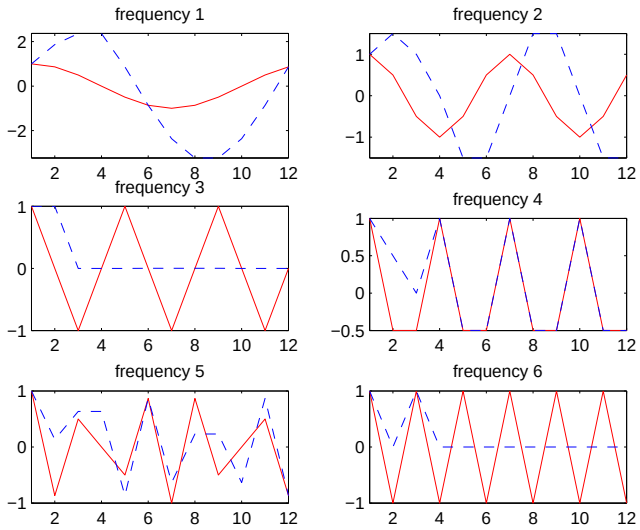
for coefficients b_j which for the running sum are $b_1 = b_2 = b_3 = 1$ and all other $b_j = 0$. A clear way to do this calculation by hand is to make a multiplication table for the elements of b and x and then sum the lower-left-to-upper-right diagonals as shown.

b\x	2	3	2	3	3	1	2	2	3	1	0	2
1	2	3	2	3	3	1	2	2	3	1	0	2
1	2	3	2	3	3	1	2	2	3	1	0	2
1	2	3	2	3	3	1	2	2	3	1	0	2
y	2	5	7	8	8	7	6	5	7	6	4	3

And this works for any b not just 1 1 1.
Try it for the 4-running total of the sequence.
Try it for x = three cycles of cosine sampled at twelve points between 0 and $11\pi/6$ inclusively, and b = 1,1,1,1. Show

cosine[12,3]	1	0	−1	0	1	0	−1	0	1	0	−1	0
total run 4	1	1	0	0	0	0	0	0	0	0	0	0

b) Write a MATLAB program `myFilter(B,A,data)` which applies coefficients `B` to the sequence stored in the array `data`. (For the moment, let `A` be 1 on invocation and ignored in `myFilter()`.)
Write a MATLAB program `cosine(sampNo,freq)` which generates `sampNo` samples of cosine over `freq` complete cycles.
Combine these two programs to study the effect of running totals on sampled cosines at various frequencies. For instance, I combined them in a program `filterSamp()` which I ran on 12 samples at frequencies 1, 2, 3, 4, 5 and 6, totalled with run length 4, so that `filterSamp([1,2,3,4,5,6],[1,1,1,1])` made the following plot.



See how frequency 3 reproduces your manual calculation from (a).
Play with `myFilter()` on other sequences (`data`) and coefficients `B`.
c) Now the Fourier transform, and a generalization of it. The above sequences are in the time domain, but filters are best analyzed in the frequency domain. In fact, filters are usually *specified* by what they do to different frequencies in the signal.
(To complete the time domain description, though, you may want to relate the "normalized", dimensionless frequencies I've been using (1, 3, etc.) to actual cycles per second. If the *sampling frequency* is, say, 20kHz (for so-so sound recording, for instance), i.e., the time between samples is 1/20 msec., show that frequency 1 is 20/12 kHz, frequency 2 is 40/12 kHz, and so on.)
With sampling, the *frequency* is periodic—see the above excursion "artefacts of sampling"—and so it is plausible to use a periodic function of frequency, $e^{i\omega}$, where ω is the normalized frequency in radians, i.e., 2π times the normalized frequency itself. We consider

(radian) frequencies between 0 and 2π, with 2π corresponding to the sampling (radian) frequency.
In fact, since only frequencies which are half the sampling frequency or less can be reproduced faithfully (see "artefacts of sampling", above), we actually consider normalized frequencies between $-\pi$ and π. And then we drop the negative frequencies. (But all filter action will be symmetric about zero, which can make the math easier.)
So the Fourier transform of the sequence x_n in terms of $e^{i\omega}$ is

$$X(e^{i\omega}) = \sum x_n e^{-i\omega n}$$

But we're going to extend this to the "z-transform", $\mathcal{Z}(x_n)$

$$X(z) = \mathcal{Z}(x_n) = \sum x_n z^{-n}$$

in which z is a 2-number, including, of course, the unit circle $e^{i\omega}$.
The reason we go to z is to have the whole 2-number space available to us, which turns out to be useful. Also it is simpler to write than $e^{i\omega}$, and enables us to see easily two important properties we'll need.
The first is *linearity*,

$$\begin{aligned}\mathcal{Z}(ax_n + by_n) &= \sum (ax_n + by_n)z^{-n} \\ &= a(\sum x_n z^{-n}) + b(\sum y_n z^{-n}) \\ &= a\mathcal{Z}(x_n) + b\mathcal{Z}(y_n)\end{aligned}$$

The second is the *delay property*,

$$\begin{aligned}\mathcal{Z}(x_{n-k}) &= \sum x_{n-k} z^{-n} \\ &= \sum x_m z^{-m-k} \\ &= z^{-k} \sum x_m z^{-m} \\ &= z^{-k} \sum x_n z^{-n} \\ &= z^{-k} \mathcal{Z}(x_n)\end{aligned}$$

where m was introduced as $n - k$ and where the $\sum$, although we didn't say so before, is from $-\infty$ to ∞ and so the shift by k does not change it. (That's the "bilateral z-transform. Alternatively the "unilateral" z-transform sums from 0 to ∞ but is better defined as $z_n = 0$ for $n < 0$: the change of variable from $n - k$ to m still makes no difference to the sum.) (By the way, the bilateral z-transform is "acausal", as we can now see, because it involves not only *lags* of non-negative n but also "leads"—lags backwards in time—of negative n.)
With these two properties, we can find the z-transform of the "difference equation" we've been playing with, $y_n = b_1 x_n + b_2 x_{n-1} + b_3 x_{n-2} + ..$:

$$\begin{aligned}\mathcal{Z}(y_n) &= b_1 \mathcal{Z}(x_n) + b_2 z^{-1} \mathcal{Z}(x_n) + b_2 z^{-2} \mathcal{Z}(x_n) + .. \\ &= (b_1 + b_2 z^{-1} + b_3 z^{-2}) \mathcal{Z}(x_n)\end{aligned}$$

And we can consider the behaviour of the filter defined by the coefficients $b_1, b_2, b_3, ..$ to be the ratio of the z-transform of the output, y to the z-transform of the input, x

$$H(z) = \frac{\mathcal{Z}(y_n)}{\mathcal{Z}(x_n)} = b_1 + b_2 z^{-1} + b_3 z^{-2}$$

For the running total of window size 4, show that the filter "system function" is

$$H(z) = 1 + z^{-1} + z^{-2} + z^{-3} = \frac{1 - z^{-4}}{1 - z^{-1}}$$

d) Part (c) has dealt with turning difference equations into polynomials using the z-transform. We can just as well go the other way and turn polynomials in z-space ("frequency space" except that we now cover more than just the unit circle giving the frequencies from $-\pi$ to π) into the difference equations (in the time domain) that actually build the filter.
The appearance at the end of (c) of a fraction of polynomials adds a twist. But we can take it in stride.
The difference equation corresponding to the polynomial fraction we just met can be worked out from

$$\frac{\mathcal{Z}(y_n)}{\mathcal{Z}(x_n)} = \frac{1 - z^{-4}}{1 - z^{-1}}$$

so

$$(1 - z^{-1})\mathcal{Z}(y_n) = (1 - z^{-4})\mathcal{Z}(x_n)$$

and turning this into a difference equation by inverting the z-transform

$$y_n - y_{n-1} = x_n - x_{n-4}$$

And in general,

$$a_1 y_n + a_2 y_{n-1} + a_3 y_{n-2} + .. = b_1 x_n + b_2 x_{n-1} + b_3 x_{n-2} + ..$$

Working this out is more complicated than the difference equations that involve no y other than y_n. But we can do it by a sort of dynamical multiplication table. Let's take the 4-running total of our original sequence.
We start with the b coefficients as before.

$b\backslash x$	2	3	2	3	3	1	2	2	3	1	0	2
1	2	3	2	3	3	1	2	2	3	1	0	2
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	−2	−3	−2	−3	−3	−1	−2	−2	−3	−1	−0	−2
	2	3	2	3	1	−2	0	−1	0	0	−2	0

Now we repeat the process for the ys but this time we must build the second row as we go, because y is changing.

	2	3	2	3	1	−2	0	−1	0	0	−2	0
1	2	5	7	10	11	9	9	8	8	8	6	6

Compare this with the 4-running total you got for this sequence in (a).
Let's do another example of a coefficients. This has two a coefficients, better to reveal the process. With the starting sequence

x	0	1	0	0	0	0	0	0

let's apply $y_n - y_{n-1} - y_{n-2} = x_n$.

−a\y	0	1	1	2	3	5	8	13
x	0	1	0	0	0	0	0	0
1	0	1	1	2	3	5	8	13
1	0	1	1	2	3	5	8	13

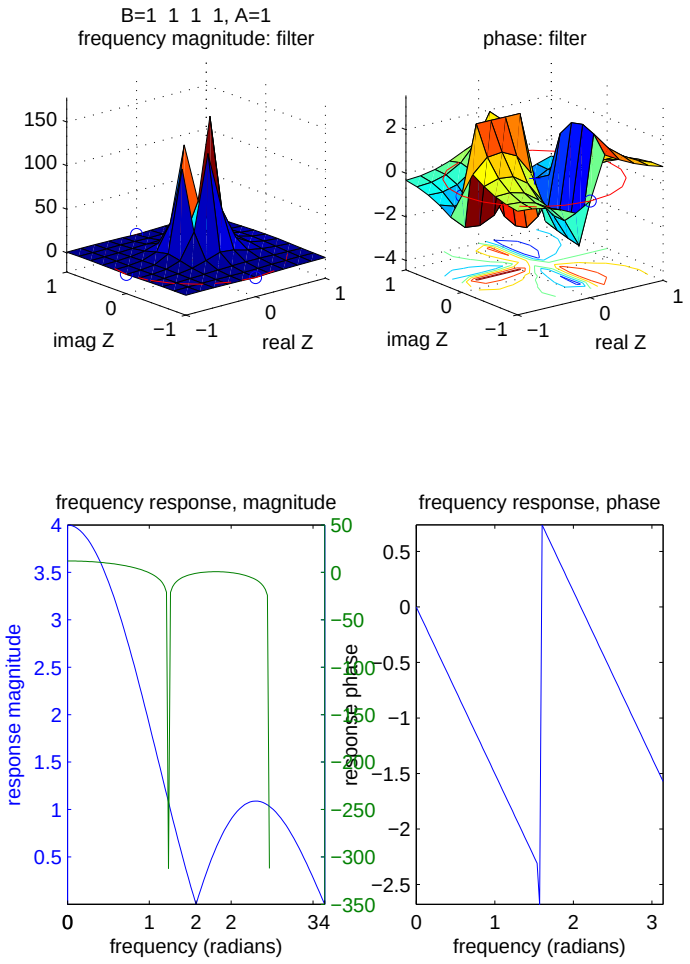
Note that x is included in the diagonals being added. And note that no column of the multiplication table can be filled in (except for x) until the corresponding y has been calculated.
What is the resulting sequence?
Modify your program from (b), `myFilter(B,A,data)`, so that it interprets the `A` array as the a coefficients. Check that it now does exactly what the MATLAB library program `filter()` does. Check it for the fractional version of the 4-running total and for the Fibonacci sequence.
e) Now we explore the filter system function $H(z)$ in z-space, i.e., the frequency domain. Write a MATLAB program `plotFilter(B,A)` which, given the coefficient sets `B` and `A`, i) plots $H(z)$ in z-space, ii) plots $H(e^{i\omega})$ as a function of ω from 0 to π, and iii) uses MATLAB's `solve` to find the zeros of the numerator polynomial (the *zeros* of $H(z)$) and the zeros of the denominator polynomial (the *poles* of $H(z)$).
For b = 1,1,1,1 and $a = 1$

$$H(z) = 1 + z^{-1} + z^{-2} + z^{-3} = \frac{z^3 + z^2 + z + 1}{z^3}$$

and the zeros are the fourth roots of 1 except for 1 itself, i.e., $-1, i, -i$. There is a pole at 0, even though $a = 1$. (MATLAB's `solve` will work better if you get rid of the negative powers in this way.)
For $b = 1, 0, 0, 0, -1$ and $a = 1, -1$

$$H(z) = \frac{1 - z^{-4}}{1 - z^{-1}} = \frac{z^4 - 1}{z^4 - z^3}$$

and the zeros are all the fourth roots of unity, i.e., $1, -1, i, -i$. But there is a pole at 1 as well as at 0 now, and this pole and the first zero cancel.
So `plotFilter([1,1,1,1],1)` will give the same result as
`plotFilter([1,0,0,0,−1],[1,−1])` except for the above apparent difference in reporting zeros and poles. Here is the result your program might give for both of these runs.



Note the three fourth roots of 1—you can see them most clearly in the phase plot in z-space. Note that the amplitude goes to 0 for frequencies $\pi/2$ and π—you can see this in the "frequency response, magnitude" plot: how does this tie up with what you found in (a) for the cosine with the frequency 1/4 of the sampling frequency, and with what you found in (b) for cosines with other frequencies up to half the sampling frequency? Use your `plotFilter()` program to explore other filters, especially 3-running totals, 5-running totals and so on. What happens to the zeros? What happens to the corresponding cosine plots as in (b) above?

(My `plotFilter()` program also converts the "frequency response, magnitude" to decibels, but the MATLAB `plotyy()` does not seem to align the horizontal axis quite the same for both plots.)

e) Looking at the "frequency response, magnitude" result for the 4-running total filter shows that it slightly attenuates frequencies above the zero at the halfway point. This allows it to serve as a (very poor) "lowpass" filter: frequencies below a certain cutoff are kept but frequencies above are removed, ideally.

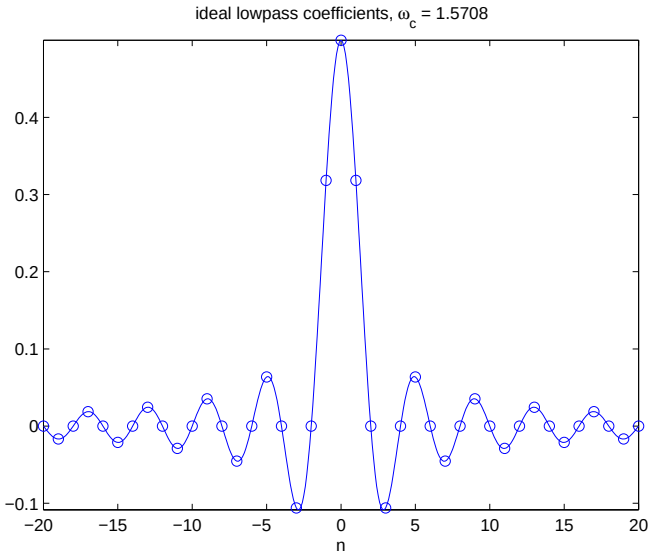
Using the fractional form of $H(z)$, try eliminating zeros other than at $z = 1$. For example, $b = 1, 0, 0, 0, -1$ and $a = 1, 1$ eliminates $z = -1$ as a zero, and gives a (very

poor) "highpass" filter: frequencies above a certain cutoff are kept but those below eliminated, ideally. Or, $b = 1, 0, 0, 0, -1$ and $a = 1, 0, 1$ eliminates $z = \pm i$ as zeros, and gives a (very poor) "bandpass" filter: frequencies below one cutoff or above another cutoff are eliminated but the ones in between are kept, ideally.
Check the math behind these assertions and figure out where all the zeros will lie.
f) Let's use one straightforward technique to design a proper lowpass filter. We'll take the easy route of leaving the denominator 1, i.e., $a = 1$. Then our task is to figure out what the bs should be.
Here is the ideal lowpass filter, drawn symmetrically about 0.

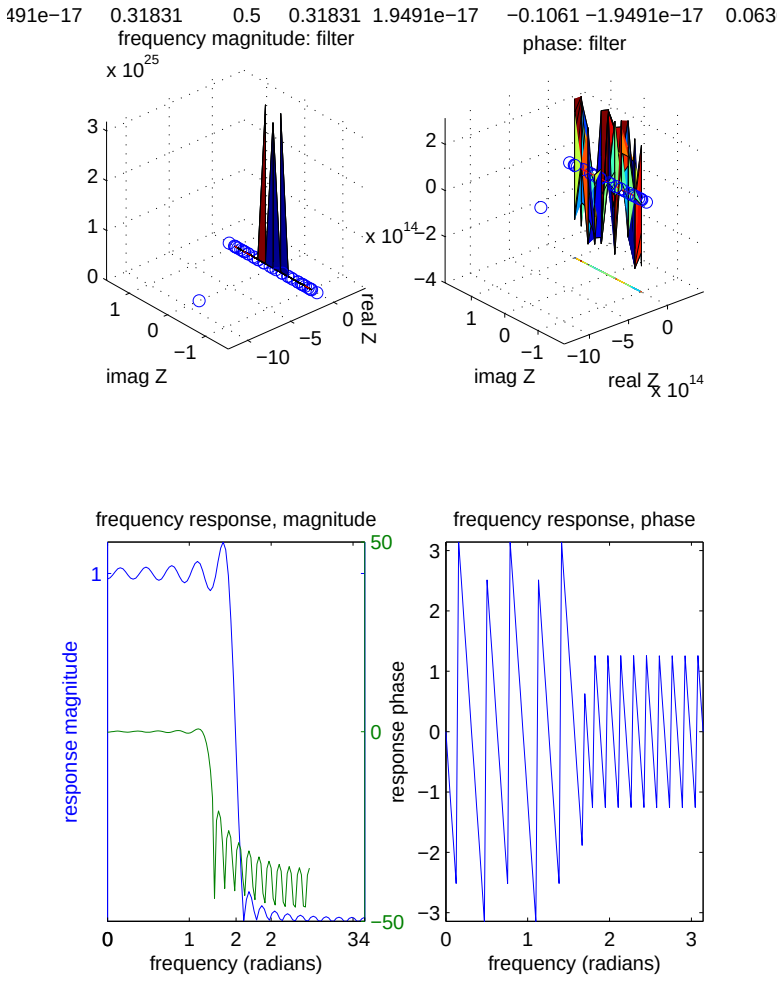
To find the filter coefficients in the time domain (i.e., for the difference equation) corresponding to this frequency-domain ideal, we must of course Fourier-transform back again. Only now we are going from a continuous set of frequencies to a discrete set of coefficients, so we must use a continuous inverse Fourier transform. That is, instead of the *sum* over all coefficients we used to Fourier-transform from sequence to frequency, we now need an *antislope*.

$$\begin{aligned}
h_n &= \frac{1}{2\pi}\text{antislope}_\omega H(\omega)e^{i\omega n} \,|_{-\pi}^{\pi} \\
&= \frac{1}{2\pi}\text{antislope}_\omega(\text{if } -\omega_c \le \omega \le \omega_c \text{ then 1 else 0})e^{i\omega n} \,|_{-\pi}^{\pi} \\
&= \frac{1}{2\pi}\frac{1}{in}e^{i\omega n} \,|_{-\omega_c}^{\omega_c} \\
&= \frac{1}{2i\pi n}(e^{i\omega_c n} - e^{-i\omega_c n}) \\
&= \frac{2i\sin\omega_c n}{2i\pi n} \\
&= \frac{\sin\omega_c n}{\pi n}
\end{aligned}$$

which, by the way, is ω_c/π when $n = 0$.
If you have learned enough calculus from other Weeks, you can check this derivation. Otherwise you'll have to accept the result that the coefficients are $\sin(\omega_c n)/(\pi n)$, for $n = -\infty$ to ∞. Here is a plot, with the specific coefficients marked for the case $\omega_c = \pi/2$.



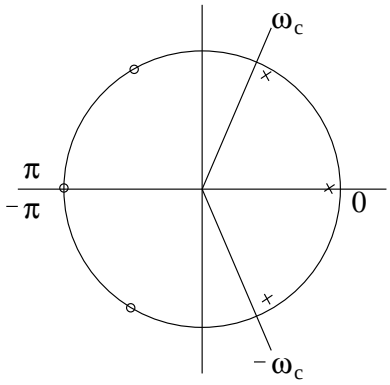
This set of coefficients goes on forever in both directions, so we must truncate to be practical. We must also shift it so that all the ns are non-negative, otherwise we run afoul of acausality. Doing this and running `plotFilter(lowpassIdeal(pi/2,20),1)` (where `lowpassIdeal()` is my program to calculate $\sin(\omega_c n)/(\pi n)$ for $\omega_c = \pi/2$ and truncated to 20 points each side of 0: you can write it for yourself) gives the following results.



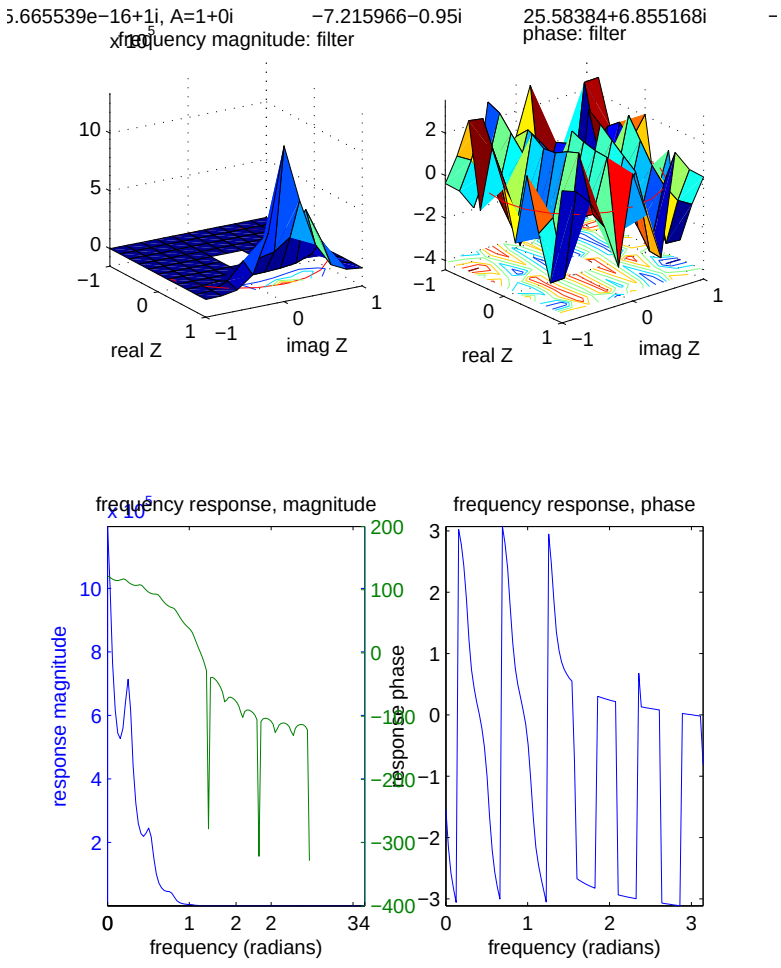
You can see how the frequency response plummets at $\pi/2$, halfway along the frequency range: it drops, say, 25 decibels, which means that the power drops a factor of $10^{(25/10)} \approx 300$ (a decibel is 10 $\log_{10}$(power ratio)). Getting rid of the wiggles top and bottom is what you'll learn in an engineering course—except that it can never completely be done because of that truncating we did.

Run this lowpass filter on the various frequencies of cosine from (b) above.

g) An informative but less satisfactory way to design a lowpass filter is the *ad hoc* method of just placing poles and zeros at suitable points in the z-plane. An idea is to space zeros and poles at equal angles around the unit circle, with zeros on the circle outside the frequencies of interest and poles just inside the circle within the range of frequencies to be accepted. Here is a picture for six equally-spaced zeros and poles.



Write a program `[zeroes poles] = lowpassAdhoc(wC,wNo,zeroRad,poleRad)` so that `lowpassAdhoc(1.22,6,1,.95)` corresponds to the picture.
Running `plotFilter(zeroes,poles)` after `[zeroes poles] = lowpassAdhoc(pi/2,24,1,.95)` will give the following, not so good, result.



In all these discussions, an engineer must remember that there are severe constraints on the number of coefficients, due to space in the DSP chip and because of running time for the filter. So the 41 coefficients in (f) and the 24 in (g) are quite impractical.

Filters with poles other than at 0 (i.e., with polynomials in z^{-1} on the denominator) are called "infinite impulse response" (IIR)—as opposed to "finite impulse response" (FIR) filters—and are often designed by transforming from continuous-time designs, which are beyond the scope of this excursion.

21. In a physical system, frequencies, which are energies, cannot be negative. Feynman mentions [FW87] a theorem which says that a function which can be Fourier decomposed into only positive frequencies must itself be nonzero essentially everywhere, including outside the lightcone if it is a function of timespace, such as the amplitude function for the location of a particle.
    Thus any quantum particle has an amplitude, perhaps very small, for moving faster than light. We'll call such states "tachyons". We saw in Weeks 3 and 7 that such a state can be seen by some relativistic observers as moving backwards in time, and so we get antimatter. We also get the exclusion principle for fermions, leading to the structure of matter, and the opposite behaviour of bosons, leading to important quantum applications such as lasers and superconductivity.
    ("Tachy" is the root of tachometer, as in a high performance sports car. It is a Greek word. The Greek word for post office is tachydromio. I'm not sure about the "dromio"—I get an image of some kind of camel—but the "tachy" means fast—very fast. In physics a tachyon is a hypothetical particle that always travels faster than light.)

22. Look up Jean Baptiste *Joseph* Fourier (1768–1830). What problem was he working on when he came up with the idea of representing functions by series?

23. Look up James W. Cooley and John W. Tukey's 1965 paper, "An Algorithm for the Machine Calculation of Complex Fourier Series". Did anybody come up with this method before them?

24. Look up chapter 32 in Kee Dewney's *The New Turing Omnibus* [Dew93] (chapter 29 of the 1989 original book). Where does his description of FFT go wrong?

25. Find other applications of the Fourier transform and use MATLAB to perform the calculations for simple examples.

26. Any part of the Preliminary Notes that needs working through.

References

[Dew93] A. K. Dewdney. *The New Turing Omnibus: 66 Excursions in Computer Science.* Computer Science Press, Rockville, MD, 1993.

[FLS64] R. P. Feynman, R. B. Leighton, and M. Sands. *The Feynman Lectures on Physics, Volume I.* Addison-Wesley, 1964.

[FW87] Richard P. Feynman and Steven Weinberg. *Elementary Particles and the Laws of Physics: The 1986 Dirac Memorial Lectures.* Cambridge University Press, Cambridge, 1987.