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

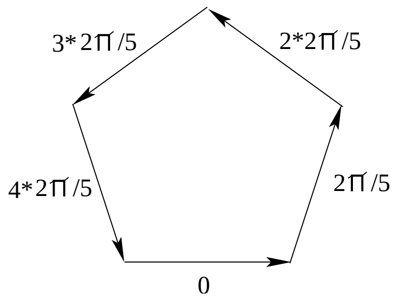
T. H. Merrett[*]
McGill University, Montreal, Canada

December 26, 2006

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 $F^5_{j\,k} = \frac{1}{\sqrt{5}} e^{ijk2\pi/5}$

$$\begin{array}{cc|cccccc}
jk & k= & 0 & 1 & 2 & 3 & 4 & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & F^5_{0\,k} \\
 & 1 & 0 & 1 & 2 & 3 & 4 & F^5_{1\,k} \\
j= & 2 & 0 & 2 & 4 & 1 & 3 & F^5_{2\,k} \\
 & 3 & 0 & 3 & 1 & 4 & 2 & F^5_{3\,k} \\
 & 4 & 0 & 4 & 3 & 2 & 1 & F^5_{4\,k} \\
 & & (F_0) & (F_1) & (F_2) & (F_3) & (F_4) & F_k
\end{array}$$

[*]Copyleft ©T. H. Merrett, 2006 Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation in a prominent place. Copyright for components of this work owned by others than T. H. Merrett must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to republish from: T. H. Merrett, School of Computer Science, McGill University, fax 514 398 3883. 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 (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

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

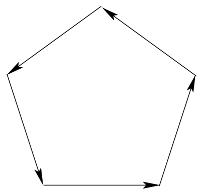
Note 2. $(F_k)_j = F^5_{j\,k} = \frac{1}{\sqrt{5}}e^{ijk\,2\pi/5}$, not just jk.

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

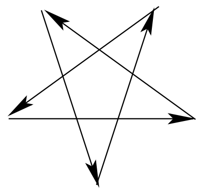
$$\begin{aligned}
F_0^*.F_0 &= \sum_{j=0}^{4}\frac{1}{\sqrt{5}}e^{-ij0\,2\pi/5}\frac{1}{\sqrt{5}}e^{ij0\,2\pi/5} = \frac{1}{5}\sum_{j=0}^{4}1 = 1\\
F_1^*.F_1 &= \sum_{j=0}^{4}\frac{1}{\sqrt{5}}e^{-ij1\,2\pi/5}\frac{1}{\sqrt{5}}e^{ij1\,2\pi/5} = \frac{1}{5}\sum_{j=0}^{4}e^{i(j-j)1\,2\pi/5} = \frac{1}{5}\sum_{j=0}^{4}1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

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:

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

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,

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

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

or shear,

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

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

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
```

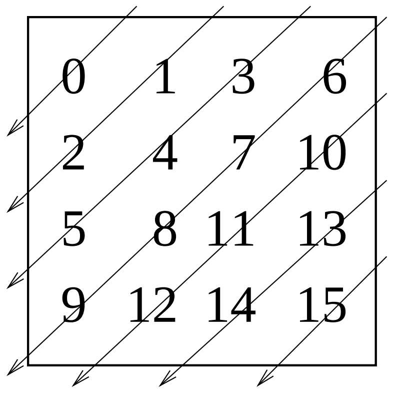
```
   0.3090 + 0.9510i
```

The real part is exact.

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

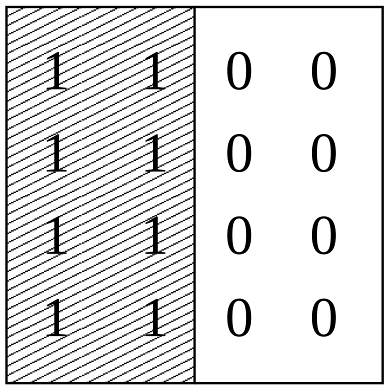
4. This is used by JPEG (Joint Photographic Experts Group) to compress images along something like the following lines.

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:

1	2	0
5	0.75	0.75

(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.

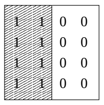
Still, we can do better.

$f'_k = P(\omega^k)$ 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 multiplicaton).

Let's explore this for `F16` and 

1	1	0	0
1	1	0	0
1	1	0	0
1	1	0	0

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

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.

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

So

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

2.

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

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

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.

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

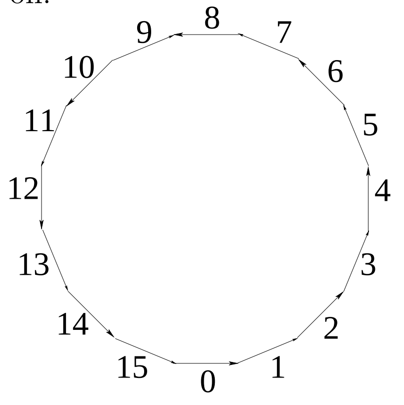
and

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

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:

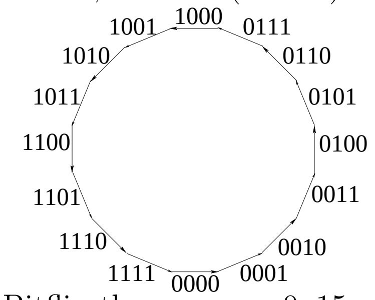


$$\omega^8=-\omega^0 \qquad \omega^0+\omega^8=0$$

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

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:

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

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

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.

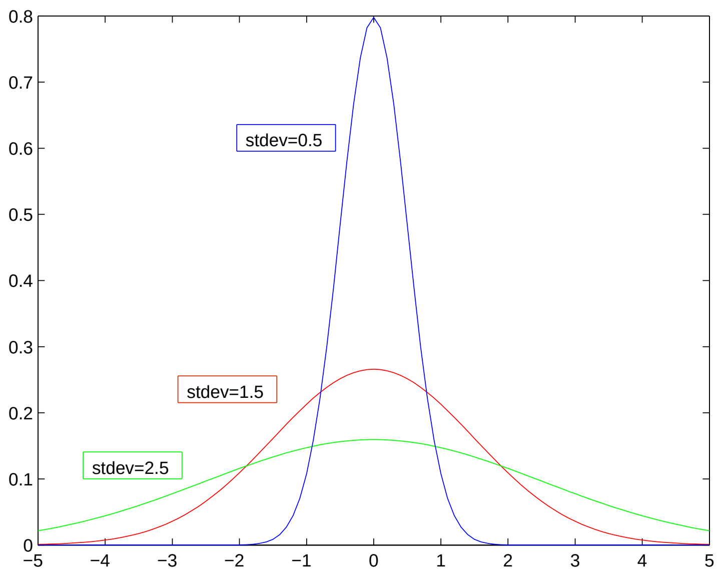
In general, $2n \lg n$ versus n^2.

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

7. The Uncertainty Principle

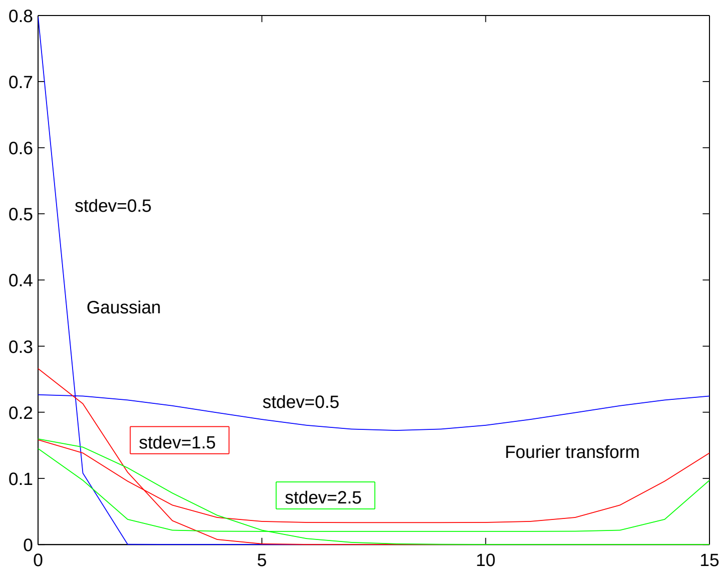
What function is the Fourier transform of itself?

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 interptet 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

10. **Excursions for Friday and beyond.**

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. 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?
8. 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
```

9. 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.)

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

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

12. 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.

13. 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?)

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

15. 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?
    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.)

16. 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?

17. 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?

18. 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?

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

20. Any part of the lecture 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.