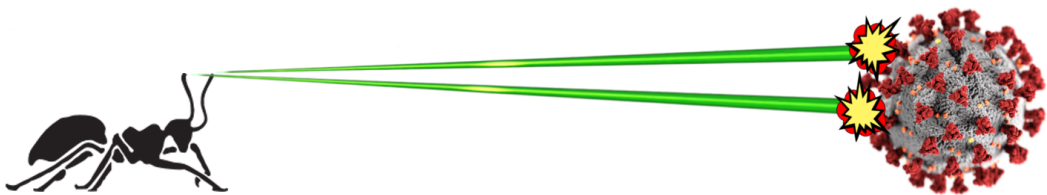
Recent results on fast multiplication

David Harvey

ANTS XIV, University of Auckland (or thereabouts), 29th/30th June 2020

University of New South Wales, Sydney
Joint work with Joris van der Hoeven (École Polytechnique, Palaiseau)



Claim

New Zealand is the “best country in the world”

— Steven Galbraith, personal communication, April 2020

What does the data show?

Claim

New Zealand is the “best country in the world”

— Steven Galbraith, personal communication, April 2020

What does the data show?

	Australia	New Zealand
Sheep (per capita)	2.7	5.3
Fields medallists (per 100,000,000)	7.8	20.0
ANTS conferences (per 100,000,000)	3.9	20.0
Reported COVID-19 cases (per 100,000)	30.3	30.5

Integer multiplication

$$\mathsf{M}(n) := \text{cost of multiplying integers with at most } n \text{ digits}$$

- "digits" means in some fixed base (e.g. binary or decimal).
- "cost" means "bit complexity"
  (e.g. # steps on multi-tape Turing machine, or # gates in Boolean circuit).

Integer multiplication

$$M(n) := \text{cost of multiplying integers with at most } n \text{ digits}$$

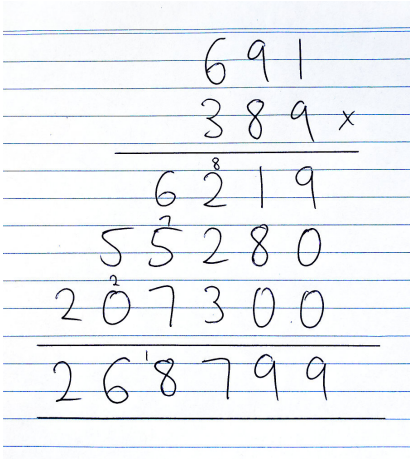
- "digits" means in some fixed base (e.g. binary or decimal).
- "cost" means "bit complexity"
  (e.g. # steps on multi-tape Turing machine, or # gates in Boolean circuit).

Polynomial multiplication over finite fields

$$M_q(n) := \text{cost of multiplying polynomials in } \mathbb{F}_q[x] \text{ of degree at most } n$$

- $\mathbb{F}_q$ = field with q elements, q a fixed prime power.
- "cost" means bit complexity, or # ring operations in $\mathbb{F}_q$.

Long multiplication



Goes back at least to ancient Egypt — probably much older.

Complexity:

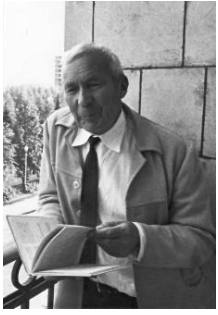
$$\mathrm{M}(n) = O(n^2).$$

Same algorithm for polynomials:

$$\mathrm{M}_q(n) = O(n^2).$$

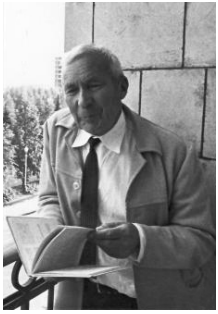
Conjecture (Kolmogorov, around 1956)

$$M(n) = \Theta(n^2).$$



Conjecture (Kolmogorov, around 1956)

$$M(n) = \Theta(n^2).$$



According to Karatsuba (1995),

> *"Probably, [the conjecture's] appearance is based on the fact that throughout the history of mankind people have been using [the algorithm] whose complexity is $O(n^2)$, and if a more economical method existed, it would have already been found."*

Brief history of bounds for $\mathsf{M}(n)$:

1962	Karatsuba	$n^{\log 3/\log 2}$ $(\approx n^{1.58})$
1969	Knuth	$n\, 2^{\sqrt{2\log n/\log 2}} \log n$
1971	Schönhage–Strassen	$n \log n \log\log n$
2007	Fürer	$n \log n\, K^{\log^* n}$ for some $K > 1$
2019	H.–van der Hoeven*	$n \log n$

*not yet published

Brief history of bounds for $\mathsf{M}(n)$:

1962	Karatsuba	$n^{\log 3/\log 2}$ $(\approx n^{1.58})$
1969	Knuth	$n\,2^{\sqrt{2\log n/\log 2}}\log n$
1971	Schönhage–Strassen	$n\log n\log\log n$
2007	Fürer	$n\log n\,K^{\log^* n}$ for some $K>1$
2019	H.–van der Hoeven*	$n\log n$

Conjecture (Schönhage–Strassen, 1971)

$$\mathsf{M}(n)=\Theta(n\log n).$$

*not yet published

Brief history of bounds for $\mathsf{M}_q(n)$:

1977	Schönhage	$n \log n \log\log n$
2017	H.–van der Hoeven–Lecerf	$n \log n \, 8^{\log^* n}$
2019	H.–van der Hoeven	$n \log n \, 4^{\log^* n}$
2019	H.–van der Hoeven*	$n \log n$ (conditional on unproved number-theoretic hypothesis)

*not yet published

Brief history of bounds for $\mathsf{M}_q(n)$:

1977	Schönhage	$n \log n \log\log n$
2017	H.–van der Hoeven–Lecerf	$n \log n \, 8^{\log^* n}$
2019	H.–van der Hoeven	$n \log n \, 4^{\log^* n}$
2019	H.–van der Hoeven*	$n \log n$ (conditional on unproved number-theoretic hypothesis)

Unsolved problem

Can we get $\mathsf{M}_q(n) = O(n \log n)$ unconditionally?

*not yet published

Outline of rest of talk

1. Complex DFTs and FFTs
2. Further FFT techniques (Rader & Nussbaumer)
3. Conditional $O(n \log n)$ multiplication for integers and polynomials
4. Unconditional $O(n \log n)$ integer multiplication

Complex DFTs and FFTs

Let $n \geqslant 1$ and $\zeta := e^{2\pi i/n} \in \mathbb{C}$. The roots of $x^n - 1$ are $1, \zeta, \ldots, \zeta^{n-1}$.

The **DFT of length n** over $\mathbb{C}$ is the linear map (in fact ring isomorphism)

$$\mathbb{C}[x]/(x^n - 1) \longrightarrow \mathbb{C}^n, \qquad F \longmapsto (F(1), F(\zeta), \ldots, F(\zeta^{n-1})).$$

Let $n \geqslant 1$ and $\zeta := e^{2\pi i/n} \in \mathbb{C}$. The roots of $x^n - 1$ are $1, \zeta, \ldots, \zeta^{n-1}$.

The **DFT of length n** over $\mathbb{C}$ is the linear map (in fact ring isomorphism)

$$\mathbb{C}[x]/(x^n - 1) \longrightarrow \mathbb{C}^n, \qquad F \longmapsto (F(1), F(\zeta), \ldots, F(\zeta^{n-1})).$$

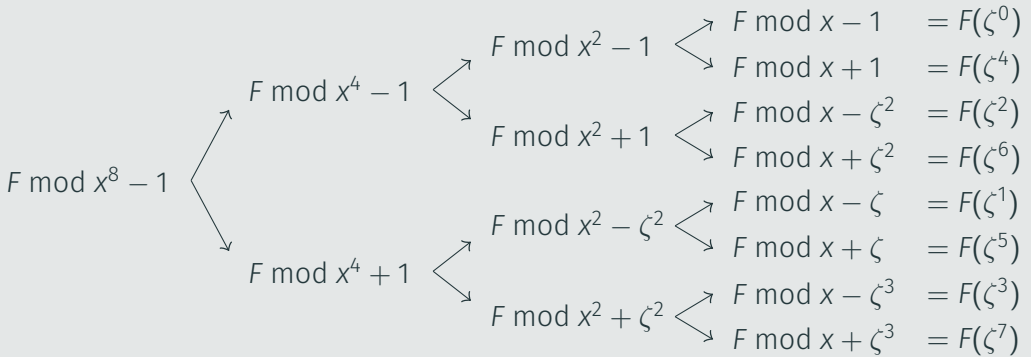
Example: DFT of length 4

$$a + bx + cx^2 + dx^3 \pmod{x^4 - 1} \quad \longmapsto \quad \begin{pmatrix} a+b+c+d \\ a+ib-c-id \\ a-b+c-d \\ a-ib-c+id \end{pmatrix} \in \mathbb{C}^4.$$

The naive algorithm to evaluate the DFT requires $O(n^2)$ operations in $\mathbb{C}$.

The simplest case of the **Cooley–Tukey FFT** (1965) reduces the complexity of the DFT from $O(n^2)$ to $O(n \log n)$ operations in the case $n = 2^k$.

Example: FFT of length 8



DFTs can be used to compute **cyclic convolutions**, i.e. multiply in $\mathbb{C}[x]/(x^n - 1)$.

Given as input $F, G \in \mathbb{C}[x]/(x^n - 1)$:

1. use DFT to compute $a_j := F(\zeta^j)$ and $b_j := G(\zeta^j)$ for $j = 0, \ldots, n-1$
2. compute pointwise products $c_j := a_j \cdot b_j$
3. use inverse DFT to find $H \in \mathbb{C}[x]/(x^n - 1)$ such that $H(\zeta^j) = c_j$ for all j

Output is $H = FG \pmod{x^n - 1}$.

Therefore, FFTs lead to fast convolution algorithms.

Suppose we want to multiply n-bit integers u and v.

Step 1. Rewrite u and v in base 2^b where $b \approx \log n$.
Encode as polynomials $F, G \in \mathbb{Z}[x]$, coefficient size b bits, degree $\approx n/\log n$.

Baby example (in decimal). Suppose that

$$u = 314159265, \qquad v = 271828182.$$

Rewrite in base 10^3, i.e. put $u = F(10^3)$ and $v = G(10^3)$ where

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182.$$

Step 2. Multiply in $\mathbb{C}[x]$ using complex FFTs, with working precision $O(\log n)$ bits. Round result to get correct product in $\mathbb{Z}[x]$.

Baby example continued. Compute the polynomial product

$$\begin{aligned} H(x) &= (314x^2 + 159x + 265) \times (271x^2 + 828x + 182) \\ &= 85094x^4 + 303081x^3 + 260615x^2 + 248358x + 48230. \end{aligned}$$

Step 2. Multiply in $\mathbb{C}[x]$ using complex FFTs, with working precision $O(\log n)$ bits. Round result to get correct product in $\mathbb{Z}[x]$.

Baby example continued. Compute the polynomial product

$$\begin{aligned} H(x) &= (314x^2 + 159x + 265) \times (271x^2 + 828x + 182) \\ &= 85094x^4 + 303081x^3 + 260615x^2 + 248358x + 48230. \end{aligned}$$

Step 3. Substitute $x = 2^b$ to get product in $\mathbb{Z}$.

Baby example continued. Substitute $x = 10^3$ to get

$$uv = H(10^3) = 85397341863406230.$$

Sketch of complexity analysis:

- Overall complexity dominated by the FFTs of length $O(n/\log n)$.
- Coefficient multiplications in $\mathbb{C}$ are handled recursively:
  cost per multiplication is $O(\mathsf{M}(\log n))$.

Sketch of complexity analysis:

- Overall complexity dominated by the FFTs of length $O(n/\log n)$.
- Coefficient multiplications in $\mathbb{C}$ are handled recursively: cost per multiplication is $O(\mathsf{M}(\log n))$.

So we get

$$\begin{aligned}\mathsf{M}(n) &= O\big(\tfrac{n}{\log n}\log\big(\tfrac{n}{\log n}\big)\,\mathsf{M}(\log n)\big)\\ &= O(n\,\mathsf{M}(\log n))\\ &= O(n\log n\,\mathsf{M}(\log\log n))\\ &= O(n\log n\log\log n\,\mathsf{M}(\log\log\log n))\\ &= \cdots\end{aligned}$$

Further FFT techniques (Rader & Nussbaumer)

Rader's algorithm (1968)

A DFT of prime length p may be reduced to a cyclic convolution of length $p-1$, together with $O(p)$ additions in $\mathbb{C}$.

Example: let $\zeta = e^{2\pi i/5}$, and suppose we want to compute the DFT

$$
\begin{aligned}
&u_0 + u_1 \quad + u_2 \quad + u_3 \quad + u_4 \\
&u_0 + u_1\zeta^1 + u_2\zeta^2 + u_3\zeta^3 + u_4\zeta^4 \\
&u_0 + u_1\zeta^2 + u_2\zeta^4 + u_3\zeta^1 + u_4\zeta^3 \\
&u_0 + u_1\zeta^3 + u_2\zeta^1 + u_3\zeta^4 + u_4\zeta^2 \\
&u_0 + u_1\zeta^4 + u_2\zeta^3 + u_3\zeta^2 + u_4\zeta^1.
\end{aligned}
$$

This reduces to computing the cyclic convolution of

$$(u_1, u_2, u_4, u_3) \quad \text{and} \quad (\zeta^3, \zeta^4, \zeta^2, \zeta^1),$$

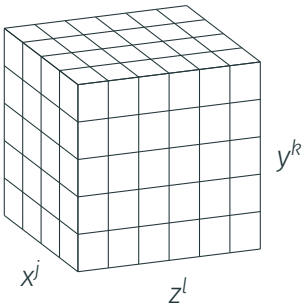
plus a few additions.

The ordering is given by the powers of the multiplicative generator 2 modulo 5.

DFTs may be generalised to polynomials in several variables.

Example: let $F \in \mathbb{C}[x, y, z]/(x^4 - 1, y^5 - 1, z^6 - 1)$, say

$$F = \sum_{j=0}^{3} \sum_{k=0}^{4} \sum_{l=0}^{5} F_{j,k,l} x^j y^k z^l.$$



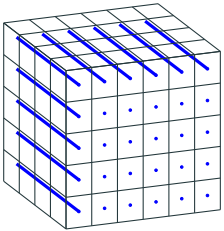
Let $\zeta_n = e^{2\pi i/n}$, and assume we want to evaluate

$$F(\zeta_4^j, \zeta_5^k, \zeta_6^l)$$

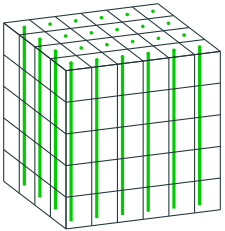
for all $j \in \{0, 1, 2, 3\}$, $k \in \{0, \ldots, 4\}$, $l \in \{0, \ldots, 5\}$.

This is a 3-dimensional DFT of size $4 \times 5 \times 6$.

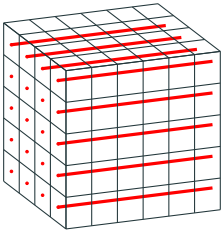
The "standard" method for evaluating a multidimensional DFT is to perform 1-dimensional DFTs in each dimension successively:



30 DFTs of length 4
with respect to x



24 DFTs of length 5
with respect to y



20 DFTs of length 6
with respect to z

Nussbaumer proposed a better algorithm (late 1970s) for the special case that **all the dimensions are powers of two**, say

$$F \in \mathbb{C}[x_1, \ldots, x_d]/(x_1^{2^{k_1}} - 1, \ldots, x_d^{2^{k_d}} - 1), \qquad k_1 \leqslant \cdots \leqslant k_d.$$

Step 1. Treating the last variable x_d as a "synthetic" root of unity of order 2^{k_d}, evaluate the first $d-1$ variables at suitable powers of x_d.*

This uses the Cooley–Tukey FFT, but with no multiplications in $\mathbb{C}$: multiplying by a "synthetic" root of unity only involves data rearrangement.

Step 2. Evaluate the last variable at the usual complex roots of unity, using any convenient method.

*Actually, this doesn't quite work; one needs to fudge things to work with $x^{2^k} + 1$ instead of $x^{2^k} - 1$.

Let $n = 2^{k_1} \times \cdots \times 2^{k_d}$ be the total data size.

Assuming all the dimensions 2^{k_j} are roughly equal:

	additions in $\mathbb{C}$	multiplications in $\mathbb{C}$
standard algorithm	$O(n \log n)$	$O(n \log n)$
Nussbaumer's algorithm	$O(n \log n)$	$O\left(\frac{n \log n}{d}\right)$

The bottom line

Nussbaumer reduces the number of multiplications in $\mathbb{C}$ by a factor of $O(d)$.

Conditional $O(n \log n)$ multiplication for integers and polynomials

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

Step 1. Cut integers into chunks of 46 ($\approx \log_2 10^{14}$) bits.

Encode into polynomials in $\mathbb{Z}[t]$, with 46-bit coefficients, and degree less than

$$\lceil n/46 \rceil = 2\,173\,913\,043\,479.$$

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

Step 1. Cut integers into chunks of 46 ($\approx \log_2 10^{14}$) bits.

Encode into polynomials in $\mathbb{Z}[t]$, with 46-bit coefficients, and degree less than

$$\lceil n/46 \rceil = 2\,173\,913\,043\,479.$$

It suffices to multiply the polynomials in the ring $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = 5\,509\,236\,183\,041 = p_1 p_2 p_3, \qquad p_1 = 15361,\ p_2 = 18433,\ p_3 = 19457.$$

This let us recover the product in $\mathbb{Z}[t]$ because $N > 2 \times 2\,173\,913\,043\,479$.

The primes are chosen very carefully; I will return to this in a few slides.

Step 2. Using the Chinese remainder theorem, there is an isomorphism (Agarwal–Cooley 1977):

$$\mathbb{Z}[t]/(t^{5\,509\,236\,183\,041} - 1) \cong \mathbb{Z}[x, y, z]/(x^{15361} - 1, y^{18433} - 1, z^{19457} - 1),$$
$$t \longmapsto xyz.$$

Can be computed efficiently in either direction (just rearrange coefficients).

So we have reduced to a 3-dimensional cyclic convolution of size $p_1 \times p_2 \times p_3$.

Step 3. Use the same strategy as the Schönhage–Strassen algorithm to reduce multiplication in

$$\mathbb{Z}[x, y, z]/(x^{15361} - 1, y^{18433} - 1, z^{19457} - 1)$$

to several complex DFTs of size $p_1 \times p_2 \times p_3$:

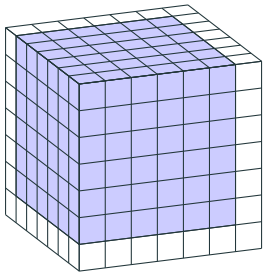
1. Compute (multidimensional) DFTs of both polynomials over $\mathbb{C}$.
2. Multiply pointwise in $\mathbb{C}$.
3. Perform inverse DFT to get approximate product in

$$\mathbb{C}[x, y, z]/(x^{15361} - 1, y^{18433} - 1, z^{19457} - 1).$$

4. Round resulting coefficients to nearest integer.
   (Working precision throughout is a small multiple of 46 bits.)

Step 4. Use multidimensional version of Rader's trick to reduce complex DFT of size $15361 \times 18433 \times 19457$ to a multiplication in

$$\mathbb{C}[x, y, z]/(x^{15360} - 1, y^{18432} - 1, z^{19456} - 1).$$



Key observation

Convolution lengths are reduced from p_i to $p_i - 1$.

I picked the primes very carefully; notice that

$$15360 = 15 \times 2^{10}, \qquad 18432 = 18 \times 2^{10}, \qquad 19456 = 19 \times 2^{10}.$$

Step 5. Use standard techniques to reduce multiplication in

$$\mathbb{C}[x, y, z]/(x^{15360} - 1, y^{18432} - 1, z^{19456} - 1)$$

to a collection of smaller 3-dimensional DFTs:

1. "Nice" DFTs of size $2^{10} \times 2^{10} \times 2^{10}$.

   These may be evaluated via Nussbaumer's trick.
   (This is the decisive point where we win over previous algorithms.)

2. "Annoying" DFTs of size $15 \times 18 \times 19$.

   These can be computed using any convenient method… (??!!)

In the general case, we need to take d somewhat bigger than 3. It turns out that $d = \Theta(1)$ is good enough (i.e., independent of n).

The sticking point in the complexity analysis turns out to be the cost of the "annoying" DFTs.

To make these DFTs cheap enough, we need to prove existence of small primes in arithmetic progressions of the type $p = 1 \pmod{2^k}$.

In the general case, we need to take d somewhat bigger than 3. It turns out that $d = \Theta(1)$ is good enough (i.e., independent of n).

The sticking point in the complexity analysis turns out to be the cost of the "annoying" DFTs.

To make these DFTs cheap enough, we need to prove existence of small primes in arithmetic progressions of the type $p = 1 \pmod{2^k}$.

Linnik's theorem (1944)

There exists a constant $L > 1$ such that for any relatively prime integers a and $m \gg 0$, there exists a prime $p = a \pmod{m}$ with $p < m^L$.

A **Linnik constant** is a value of L for which the above statement holds.

Best published Linnik constant is currently $L = 5.18$ (Xylouris, 2011).

Linnik's theorem is **embarrassingly weak**!

Example: consider $p = 1 \pmod{2^{10}}$. The first few primes are

$$p = 12289,\ 13313,\ 15361,\ 18433,\ 19457,\ 25601,\ 37889,\ 39937, \ldots.$$

But Linnik's theorem (with the best known L) only guarantees that

$$p < (2^{10})^{5.18} \approx 4 \times 10^{15}.$$

Linnik's theorem is **embarrassingly weak**!

Example: consider $p = 1 \pmod{2^{10}}$. The first few primes are

$$p = 12289,\ 13313,\ 15361,\ 18433,\ 19457,\ 25601,\ 37889,\ 39937, \ldots.$$

But Linnik's theorem (with the best known L) only guarantees that

$$p < (2^{10})^{5.18} \approx 4 \times 10^{15}.$$

Under GRH, can prove that any $L > 2$ is a Linnik constant (Heath-Brown 1992).

This is still hopeless: we get

$$p < (2^{10})^2 \approx 10^6.$$

Widely-believed conjecture

Any $L > 1$ is a Linnik constant.

Widely-believed conjecture

Any $L > 1$ is a Linnik constant.

Theorem (H.–van der Hoeven 2019)

If there exists a Linnik constant $L < 1 + \frac{1}{303}$, then the cost of the "annoying" DFTs can be controlled, and the algorithm sketched in this talk achieves

$$\mathsf{M}(n) = O(n \log n).$$

We can probably weaken the bound for L a bit, but we have no idea how to get anywhere near $L = 2$.

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)
4. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1} - 1, \ldots, x_d^{p_d} - 1)$
5. Reduce to DFTs of size $p_1 \times \cdots \times p_d$ over $\mathbb{F}_{q^s}$
6. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1-1} - 1, \ldots, x_d^{p_d-1} - 1)$ (Rader)
7. Use Nussbaumer to do synthetic FFTs in $d - 1$ dimensions, etc etc.

Theorem (H.–van der Hoeven 2019)

If there exists a Linnik constant $L < 1 + 2^{-1162}$, then

$$\mathsf{M}_q(n) = O(n \log n).$$

Can probably improve 2^{-1162}, but we don't know by how much.

Unconditional $O(n \log n)$ integer multiplication

Again take $n = 10^{14}$ bits.

As before, reduce to multiplying polynomials of degree 2 173 913 043 479 with 46-bit coefficients.

Again take $n = 10^{14}$ bits.

As before, reduce to multiplying polynomials of degree 2 173 913 043 479 with 46-bit coefficients.

This time we choose primes

$$p_1 = 16381, \qquad p_2 = 16369, \qquad p_3 = 16363.$$

Notice they are all **just below $2^{14} = 16384$**.

(Easy to find such primes. No arithmetic progressions involved.)

It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

As before, reduce to complex DFTs of size $16381 \times 16369 \times 16363$.

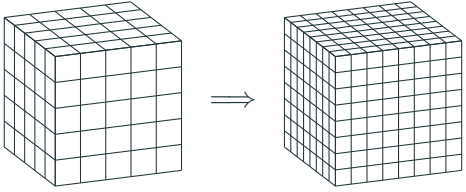
It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

As before, reduce to complex DFTs of size $16381 \times 16369 \times 16363$.

But instead of using Rader's algorithm, we use a new technique called **Gaussian resampling** to directly reduce to a DFT of size $2^{14} \times 2^{14} \times 2^{14}$.

Then we win by using Nussbaumer's method to evaluate this last DFT.



DFT of size $p_1 \times p_2 \times p_3$

DFT of size $2^{14} \times 2^{14} \times 2^{14}$

Example: given input $u \in \mathbb{C}^{13}$, suppose we want to compute DFT $\hat{u} \in \mathbb{C}^{13}$.

Suppose however that we only know how to compute DFTs of length 16.

Example: given input $u \in \mathbb{C}^{13}$, suppose we want to compute DFT $\hat{u} \in \mathbb{C}^{13}$.

Suppose however that we only know how to compute DFTs of length 16.

We will convert length 13 to length 16 via a certain **resampling map**

$$S\colon \mathbb{C}^{13} \to \mathbb{C}^{16}.$$

I will show how to construct S over the next few slides.

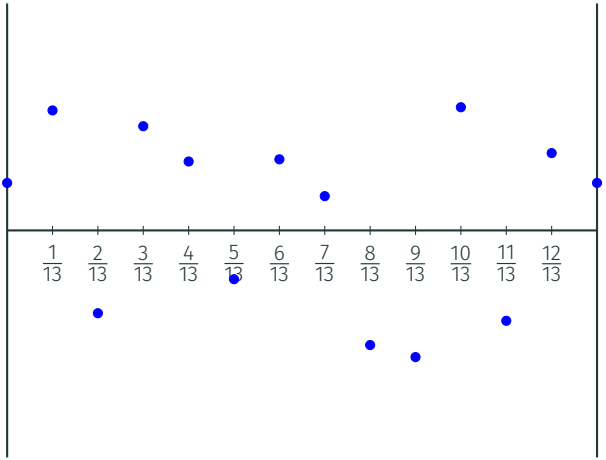
The resampling map

The diagram shows a typical input vector $u \in \mathbb{C}^{13}$.

For simplicity we assume $u_i \in \mathbb{R}$.

The blue points are $(\frac{i}{13}, u_i)$ for $i = 0, \ldots, 12$.

Notice the x-axis wraps around from left to right (i.e., the x-values live in $\mathbb{R}/\mathbb{Z}$).



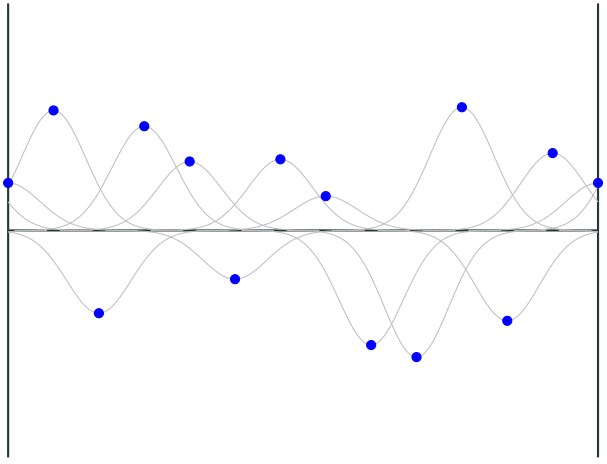
The resampling map

Draw a Gaussian curve centred around each data point.

The equation for the i-th point is

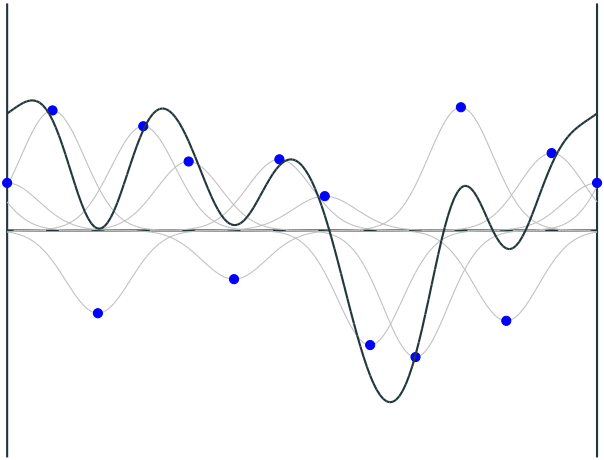
$$y = u_i e^{-13^2(x-\frac{i}{13})^2}.$$

The "height" of the curve is u_i and the "width" is $\frac{1}{13}$.



Add up all the Gaussians to get a nice smooth 1-periodic curve:

$$f(x) = \sum_{i=0}^{12} u_i e^{-13^2(x-\frac{i}{13})^2}.$$



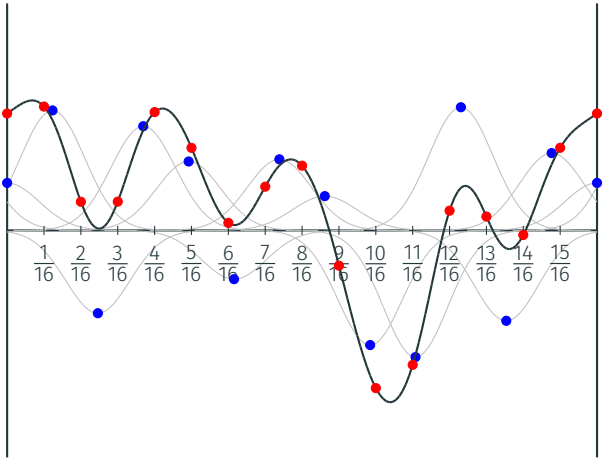
The resampling map

Add up all the Gaussians to get a nice smooth 1-periodic curve:

$$f(x) = \sum_{i=0}^{12} u_i e^{-13^2(x-\frac{i}{13})^2}.$$

The resampled vector $v = S(u)$ is defined by evaluating $f(x)$ at 16 equally-spaced points:

$$v_j = f(\tfrac{j}{16}), \qquad j = 0, \ldots, 15.$$



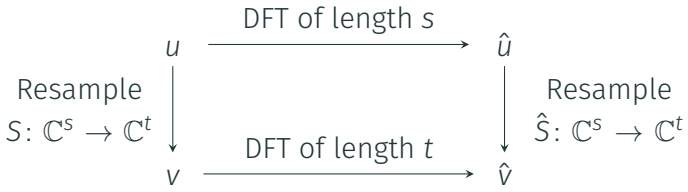
The resampling map

$$
\begin{bmatrix}
1.000 & 0.368 & 0.018 & & & & & & & & 0.018 & 0.368 \\
0.517 & 0.965 & 0.244 & 0.008 & & & & & & & & 0.037 \\
0.071 & 0.677 & 0.869 & 0.151 & 0.004 & & & & & & & 0.001 \\
0.003 & 0.127 & 0.826 & 0.729 & 0.087 & 0.001 & & & & & & \\
 & 0.006 & 0.210 & 0.939 & 0.570 & 0.047 & 0.001 & & & & & \\
 & & 0.014 & 0.323 & 0.996 & 0.415 & 0.023 & & & & & \\
 & & & 0.030 & 0.465 & 0.984 & 0.282 & 0.011 & & & & \\
 & & & 0.001 & 0.058 & 0.623 & 0.907 & 0.179 & 0.005 & & & \\
 & & & & 0.002 & 0.105 & 0.779 & 0.779 & 0.105 & 0.002 & & \\
 & & & & & 0.005 & 0.179 & 0.907 & 0.623 & 0.058 & 0.001 & \\
 & & & & & & 0.011 & 0.282 & 0.984 & 0.465 & 0.030 & \\
 & & & & & & & 0.023 & 0.415 & 0.996 & 0.323 & 0.014 \\
 & & & & & & & 0.001 & 0.047 & 0.570 & 0.939 & 0.210 & 0.006 \\
0.003 & & & & & & & & 0.001 & 0.087 & 0.729 & 0.826 & 0.127 \\
0.071 & 0.001 & & & & & & & & 0.004 & 0.151 & 0.869 & 0.677 \\
0.517 & 0.037 & & & & & & & & & 0.008 & 0.244 & 0.965
\end{bmatrix}
$$

Matrix of resampling map $S\colon \mathbb{C}^{13} \to \mathbb{C}^{16}$.

Each “output” coordinate depends mainly on the nearby “input” coordinates.

Fun fact #1. The Fourier transform of a Gaussian is again a Gaussian. This leads to a commutative diagram



In our example, $s = 13$ and $t = 16$.

The map $\hat{S}$ is defined almost exactly the same way as S; it differs by some straightforward scaling factors and data reindexing.

Fun fact #2. Due to the rapid decay of the Gaussians, the resampling map can be evaluated efficiently.

If the target transform length is t, the cost is

$$O(t\sqrt{\log t})$$

operations in $\mathbb{C}$ (assuming working precision $O(\log t)$ bits).

This is **asymptotically negligble** compared to $O(t \log t)$ cost of the FFT.

Fun fact #3. The resampling map is injective.

This follows more or less from the "diagonal" structure of the matrix of S.

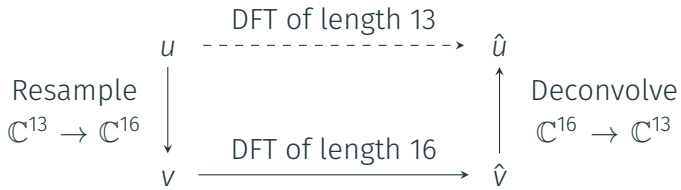
Moreover, there is a **deconvolution algorithm** that recovers u from $v = S(u)$ using

$$O(t\sqrt{\log t})$$

operations in $\mathbb{C}$.

(Note: we do not actually prove this in the paper. For technical reasons we do something a bit different.)

Conclusion: we can compute the map $u \mapsto \hat{u}$ (a DFT of length 13) by traversing the diagram as follows:



The cost of the vertical arrows is asymptotically negligible.

Combining a multidimensional version of Gaussian resampling with everything else from before, we get:

Theorem (H.–van der Hoeven 2019)

$$\mathsf{M}(n) = O(n \log n).$$

Combining a multidimensional version of Gaussian resampling with everything else from before, we get:

Theorem (H.–van der Hoeven 2019)

$$\mathsf{M}(n) = O(n \log n).$$

Unsolved problem

Can we get $\mathsf{M}_q(n) = O(n \log n)$ unconditionally?

Unfortunately, Gaussian resampling does not seem to work over $\mathbb{F}_q$.

Is there some other way of "changing the transform length" over $\mathbb{F}_q$???

Thank you!