When an algorithm in dimension one is extended to dimension d, in nearly every case its computational cost is taken to the power d. This fundamental difficulty is the single greatest impediment to solving many important problems, and has been dubbed the "Curse of Dimensionality". We have developed a way to bypass the curse for many numerical analysis applications, by using a generalization of separation of variables that allows us to control accuracy. Basic linear algebra operations can be performed in this representation using one-dimensional operations, thus avoiding the exponential scaling with respect to the dimension. These methods not only speed up computations in dimensions 2, 3, and 4 , but also allow computations in much higher dimensions that would otherwise be impossible.

Algorithms for Numerical Analysis in High Dimensions. Part I: Definitions, linear algebra and basic algorithms

Martin J. Mohlenkamp

OHIO
UNIVERSITY

In collaboration with Gregory Beylkin

Numerical operator calculus in higher dimensions.
Proc. Natl. Acad. Sci. USA, 99(16):10246–10251, August 2002.

Algorithms for numerical analysis in high dimensions.
SIAM J. Sci. Comput. to appear.

Outline

- The Curse of Dimensionality
- The Separated Representation
- Example: $\sin(x_1 + x_2 + \cdots + x_d)$
- Computational paradigm
  - Linear Algebra
  - Algorithms
  - Numerical Example in 2D: sign
  - Separation-Rank Reduction
- Nature cooperates: the Laplacian
- Solving a linear system
  - Nature cooperates: the Inverse Laplacian

The Curse of Dimensionality

When an algorithm in dimension 1 is extended to dimension d, its computational cost is taken to the **power** d.

Dense matrices:	1D	2D	d D
matrix representation	M^2	$(M^2)^2$	$(M^2)^d$
matrix-matrix multiply	M^3	$(M^3)^2$	$(M^3)^d$

Even "fast" algorithms are cursed.

Banded matrices with band b**:**	1D	2D	d D
matrix representation	bM	$(bM)^2$	$(bM)^d$
matrix-matrix multiply	b^2M	$(b^2M)^2$	$(b^2M)^d$

The Curse of Dimensionality

- In dimensions 2–4 the curse of dimensionality is painful. Matrix-matrix multiplication for $M = 100$ and $d = 3$ takes

$$(M^3)^d = 10^{18}$$

operations, which is 12 days on a 1 Teraflop computer.

- When d is truly large, computations are unthinkable.

The N-particle Schrödinger wave function for Calcium has $d = 3N = 60$ variables. With $M = 100$, simply sampling takes

$$M^d = 10^{120}$$

operations.

Ultimate Massive Parallel Super Computer
(Strömberg, ICM, 1998)

- Let the number of processors be the number of protons in the universe, $\approx 10^{80}$.

- Let the clock speed be the travel time for light to go one nuclear radius, $\approx 10^{19}$ Hz.

- Let the running time be the lifetime of the universe. (The age is $\approx 10^{17}$ s ; guess 10^4 ages.)

Then the total number of computations is

$$\approx 10^{120}$$

Separation of variables:

$$f(x_1, \ldots, x_d) \approx \phi_1(x_1) \cdots \phi_d(x_d)$$

avoids the curse and can give a useful approximate solution, but there is no way to improve the accuracy.

The natural extension:

$$f(x_1, \ldots, x_d) \approx \sum_{l=1}^{r} s_l \phi_1^l(x_1) \cdots \phi_d^l(x_d).$$

We neither fix $\{\phi_i^l(x_i)\}$, nor try to solve for appropriate $\{\phi_i^l(x_i)\}$. *We let the computation itself control the details of the representation.*

The Separated Representation

For a given ϵ, we represent a matrix in dimension d

$$\begin{aligned}\mathbb{A} &= A(j_1, j_1'; j_2, j_2'; \ldots; j_d, j_d') \\ &\approx \sum_{l=1}^{r} s_l V_1^l(j_1, j_1') V_2^l(j_2, j_2') \cdots V_d^l(j_d, j_d')\end{aligned}$$

$s_1 \geq \cdots \geq s_r > 0$, $||\mathbb{V}_i^l|| = 1$. We require:

$$\|\mathbb{A} - \sum_{l=1}^{r} s_l \mathbb{V}_1^l \otimes \mathbb{V}_2^l \otimes \cdots \otimes \mathbb{V}_d^l\| < \epsilon.$$

We call r the *separation rank*.

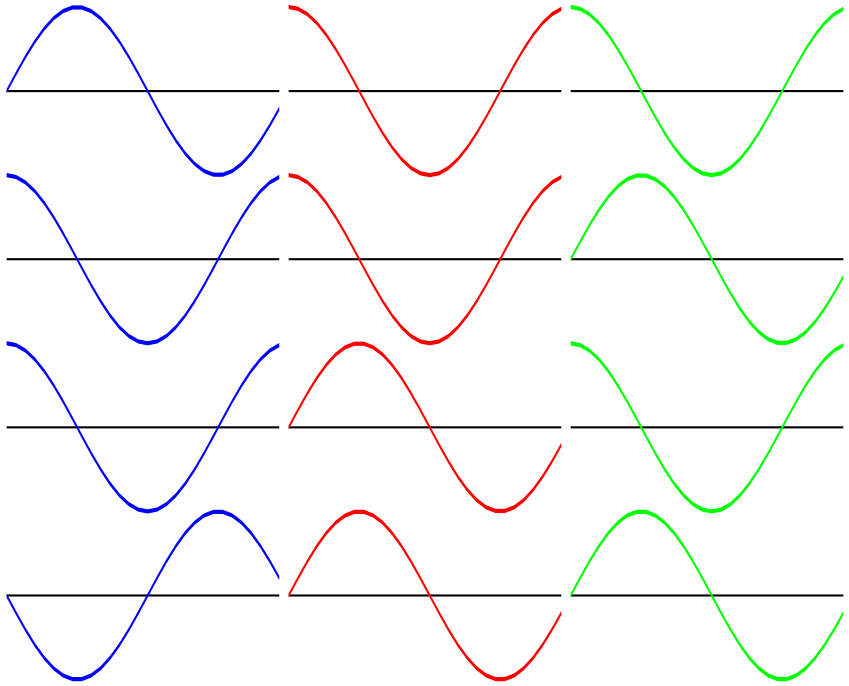
Similarly for a vector $\mathbf{F} \approx \sum_{l=1}^{r} s_l \mathbf{F}_1^l \otimes \mathbf{F}_2^l \otimes \cdots \otimes \mathbf{F}_d^l$.

Sine of the sum of several variables

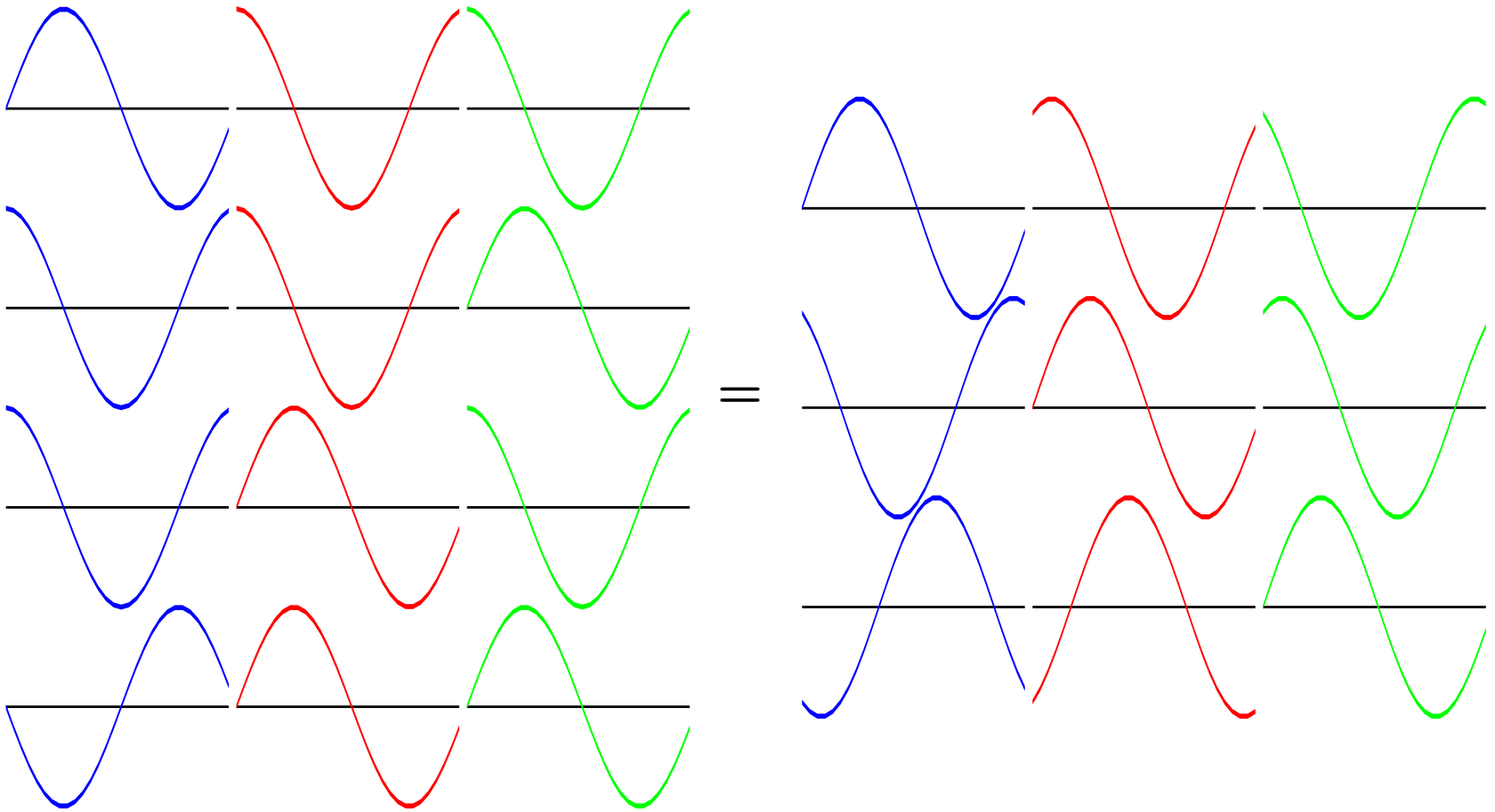
The standard trigonometric identities say $\sin(x_1 + \cdots + x_d)$ needs $r = 2^{d-1}$. For example, $d = 3 \Rightarrow r = 4$, and we have $\sin(\mathbf{x} + \mathbf{y} + \mathbf{z}) =$

$$\sin(\mathbf{x})\cos(\mathbf{y})\cos(\mathbf{z})$$
$$+ \cos(\mathbf{x})\cos(\mathbf{y})\sin(\mathbf{z})$$
$$+ \cos(\mathbf{x})\sin(\mathbf{y})\cos(\mathbf{z})$$
$$- \sin(\mathbf{x})\sin(\mathbf{y})\sin(\mathbf{z})$$

$=$



Would you have guessed?



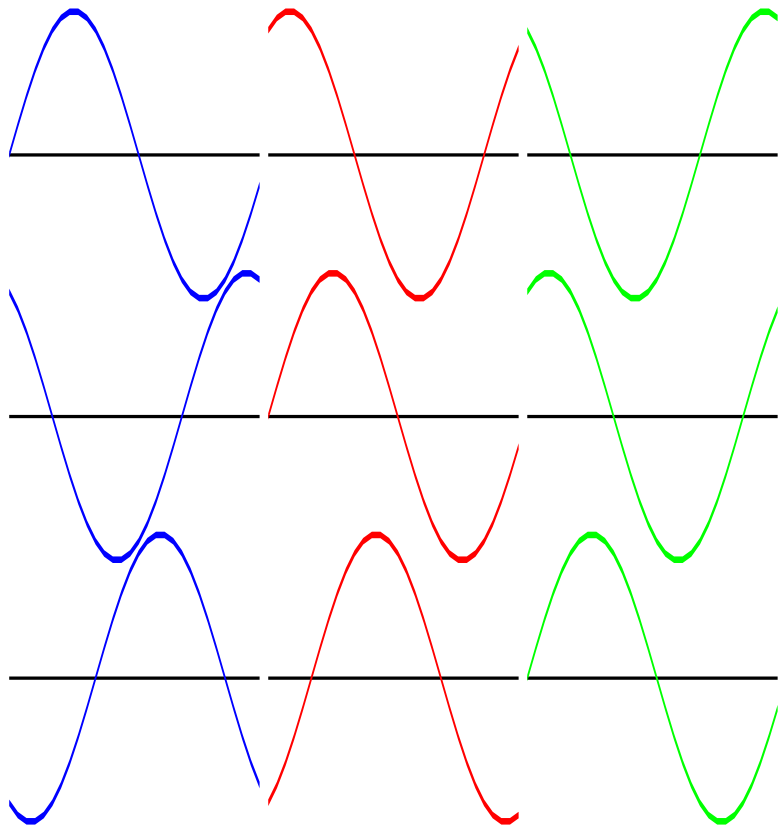
We discovered (numerically) a separation rank 3 representation: for arbitrary α, β, and γ, we have $\sin(\mathbf{x}+\mathbf{y}+\mathbf{z}) =$

$$\frac{\sin(\mathbf{x})\sin(\mathbf{y}+\beta-\alpha)\sin(\mathbf{z}+\gamma-\alpha)}{\sin(\beta-\alpha)\sin(\gamma-\alpha)}$$

$$+\frac{\sin(\mathbf{x}+\alpha-\beta)\sin(\mathbf{y})\sin(\mathbf{z}+\gamma-\beta)}{\sin(\alpha-\beta)\sin(\gamma-\beta)}$$

$$+\frac{\sin(\mathbf{x}+\alpha-\gamma)\sin(\mathbf{y}+\beta-\gamma)\sin(\mathbf{z})}{\sin(\alpha-\gamma)\sin(\beta-\gamma)}$$

$=$



Sine of the sum of several variables

Instead of $r = 2^{d-1}$ terms in dimension d,
we need only $r = d$ terms.

Theorem As long as $s(\alpha_k - \alpha_j) \neq 0$ for all $j \neq k$, the functions

$$s(x) = a \exp(bx) x$$

and

$$s(x) = a \exp(bx) \sin(cx)$$

for any complex $a \neq 0$, b, and $c \neq 0$, satisfy

$$s\left(\sum_{j=1}^{d} x_j\right) = \sum_{j=1}^{d} s(x_j) \prod_{k=1, k\neq j}^{d} \frac{s(x_k + \alpha_k - \alpha_j)}{s(\alpha_k - \alpha_j)}$$

(Joint work with Lucas Monzón.)

We have just learned

- There is more going on here than meets the eye.
- The "obvious" (analytic) separated representation may be woefully inefficient.
- There may be entire families of separated representations.
- Some separated representations may have large separation values, leading to poor conditioning.

Computational Paradigm

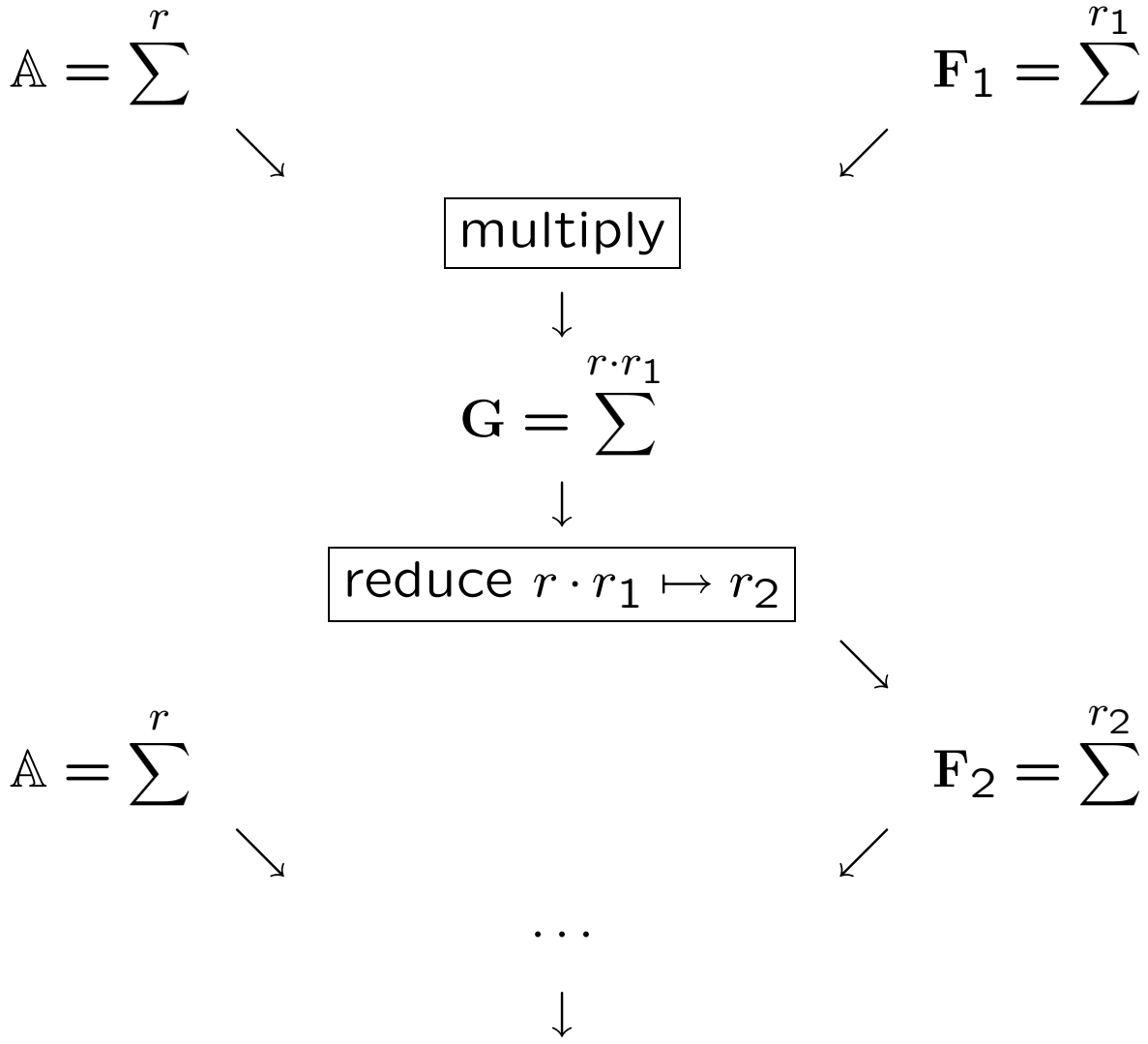
1. Start with matrices and vectors that can be represented within ϵ with low separation rank.

$$\mathbb{A} \approx \sum_{l=1}^{r} s_l \mathbb{V}_1^l \otimes \mathbb{V}_2^l \otimes \cdots \otimes \mathbb{V}_d^l$$

2. Do linear algebra operations with them. The computational cost is *linear* in d.

3. Adaptively re-minimize the separation rank of the output of each operation, while maintaining the required accuracy.

Computational Paradigm (Power Method)



Basic Linear Algebra

We can multiply a matrix times a vector in this form using only 1D operations:

$$\mathbf{G} = \mathbb{A}\mathbf{F} = \sum_{l=1}^{r} \sum_{m=1}^{r_1} s_l s_m^1 (\mathbb{A}_1^l \mathbf{F}_1^m) \otimes \cdots \otimes (\mathbb{A}_d^l \mathbf{F}_d^m).$$

The computational cost is

$$d \cdot r \cdot r_1 \cdot M^2,$$

which is **linear** in d rather than exponential.

The same applies to matrix-matrix multiplication, Frobenius norm, inner products, etc.

Compatible Algorithms

1. Power method ($\mathbf{F}_{k+1} = \mathbb{A}\mathbf{F}_k$).

2. Schulz iteration ($\mathbb{B}_{k+1} = 2\mathbb{B}_k - \mathbb{B}_k\mathbb{A}\mathbb{B}_k$) to construct $\mathbb{A}^{-1}$.

3. Sign iteration ($\mathbb{A}_{k+1} = (3\mathbb{A}_k - \mathbb{A}_k^3)/2$) to construct sign($\mathbb{A}$).

4. Scaling and squaring ($(\exp(\mathbb{A}/2^n))^{2^n}$) to construct exp($\mathbb{A}$).

5. . . .

No More Curse...

Separation rank r, dense 1D matrices:	2D	d D
matrix representation	$2rM^2$	drM^2
matrix-matrix multiply	$2r^2M^3$	dr^2M^3

Separation rank r, banded 1D matrices:	2D	d D
matrix representation	$2rbM$	$drbM$
matrix-matrix multiply	$2r^2b^2M$	dr^2b^2M

- Computations have precision ϵ.

Philosophy: The separation rank r, like the band b, has the potential to be small, and so bypasses the curse.

Computation of a Spectral Projector

A spectral projector can be computed using the sign of a matrix.

See: *Fast Spectral Projection Algorithms for Density-Matrix Computations*, J. Comput. Phys., 152(1), 1999

Compute sign($\mathbb{A}$) using

$$\mathbb{A}_0 = \mathbb{A}/\|\mathbb{A}\|_2$$
$$\mathbb{A}_{k+1} = (3\mathbb{A}_k - \mathbb{A}_k^3)/2, \quad k = 0, 1, \ldots$$

We compute sign($\mathcal{H} - 50\mathcal{I}$) for the Hamiltonian

$$\mathcal{H} = -\Delta + 20\cos(\|x\|^2),$$

on a periodic domain.

A Spectral Projector in dimension $d = 2$.

We perform 30 iterations at truncation 10^{-5} relative precision, capturing 5 eigenvalues.

	Ordinary		Separated	
M	dense	sparse	dense	sparse
8	1.4e+0	1.4e+0	2.2e+0	4.4e+0
16	9.8e+1	7.9e+1	1.3e+1	2.3e+1
32	2.9e+4	3.1e+2	9.2e+1	7.7e+1
64	(est.)e+6	2.2e+3	7.3e+2	2.3e+2
128	(est.)e+8	(est.)e+4	4.9e+3	7.8e+2
256	(est.)e+10	(est.)e+5	(est.)e+4	3.0e+3
	seconds	seconds	seconds	seconds
	Observed order			
	M^6	M^3	M^3	$M^{1.5}$

Reducing the Separation Rank

- Linear algebra operations raise the separation rank. We must reduce it, or the speed advantage will be lost.

We need to convert a sub-optimal separated representation

$$\mathbf{G} = \sum_{l=1}^{r_g} s_l^g \mathbf{G}_1^l \otimes \mathbf{G}_2^l \otimes \cdots \otimes \mathbf{G}_d^l.$$

to a (nearly) optimal one.

- When $d > 2$ there is no satisfactory theoretical foundation or known way to compute an optimal representation.
- The imperfect algorithm works well enough in practice.

Alternating Least Squares

Iteratively refine an initial approximation

$$\mathbf{F} = \sum_{l=1}^{r} s_l \mathbf{F}_1^l \otimes \mathbf{F}_2^l \otimes \cdots \otimes \mathbf{F}_d^l.$$

Loop through the directions $k = 1, \ldots, d$.

- Fix $\{\mathbf{F}_i^l\}$ for $i \neq k$, and solve a **linear** least squares problem for new $\mathbf{F}_k^l$ and s_l.

If $\|\mathbf{F} - \mathbf{G}\|$ stabilizes but is too large, then increase r and repeat.

One full alternating least squares iteration costs

$$\mathcal{O}(d \cdot r(r^2 + r_g \cdot M)).$$

Form the matrix $\mathbb{B}$ with entries

$$B(\hat{l},\tilde{l}) = \prod_{i\neq k} \langle \mathbf{F}_i^{\tilde{l}}, \mathbf{F}_i^{\hat{l}} \rangle.$$

Then form the vector $\mathbf{b}_j$ with entries

$$b_j(\hat{l}) = \sum_{l=1}^{r_{\mathbf{G}}} s_l^{\mathbf{G}} G_k^l(j) \prod_{i\neq k} \langle \mathbf{G}_i^l, \mathbf{F}_i^{\hat{l}} \rangle.$$

The normal equations for the direction k and coordinate j become

$$\mathbb{B}\mathbf{c}_j = \mathbf{b}_j,$$

which we solve for $\mathbf{c}_j = c_j(\tilde{l})$ as a vector in $\tilde{l}$. After computing $c_j(\tilde{l})$ for all j, we let $s_{\tilde{l}}^{\mathbf{F}} = \|c_j(\tilde{l})\|$ and $F_k^{\tilde{l}}(j) = c_j(\tilde{l})/s_{\tilde{l}}^{\mathbf{F}}$, where the norm is taken with respect to the coordinate j.

Controlling the Condition Number

The condition number is the ratio

$$\kappa = \frac{(\sum_{l=1}^{r} s_l^2)^{1/2}}{\|\mathbf{F}\|}\,.$$

In order to maintain significant digits we need

$$\kappa\mu\|\mathbf{F}\| \leq \epsilon\,,$$

where μ is the machine roundoff (e.g. 10^{-16}).

We can assure this by adding a penalty and minimizing $\|\mathbf{F}-\mathbf{G}\|^2 + 2\epsilon(\kappa\|\mathbf{F}\|)^2$ instead. To minimize this all we have to do is add $2\epsilon\mathbb{I}$ to $\mathbb{B}$ in the alternating least-squares algorithm.

The Laplacian

The d-dimensional Laplacian apparently has separation rank d. When $d = 4$ we have $\Delta =$

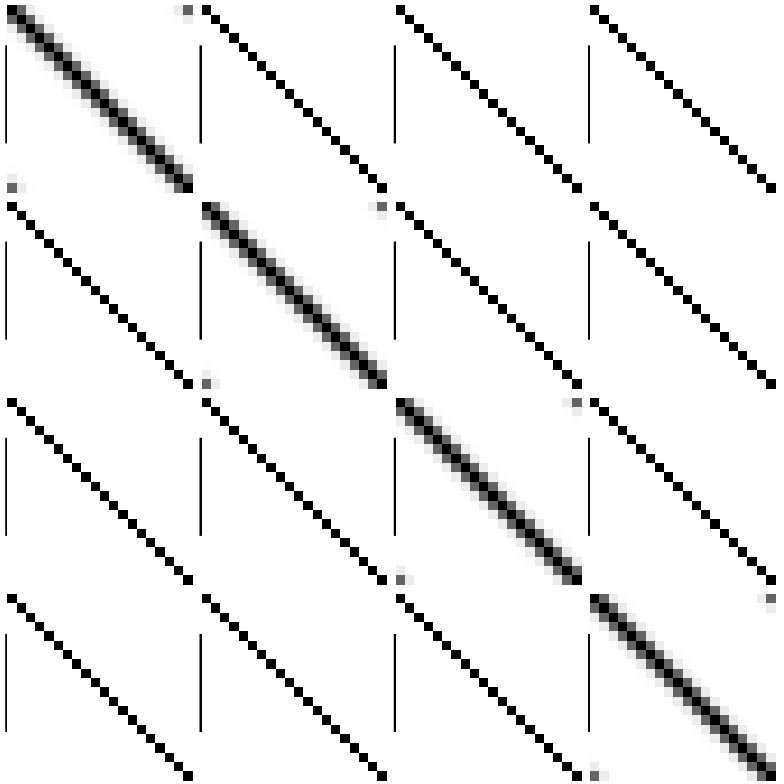
$$\frac{\partial^2}{\partial x_1^2} \otimes \mathcal{I}_2 \otimes \mathcal{I}_3 \otimes \mathcal{I}_4$$

$$+\mathcal{I}_1 \otimes \frac{\partial^2}{\partial x_2^2} \otimes \mathcal{I}_3 \otimes \mathcal{I}_4$$

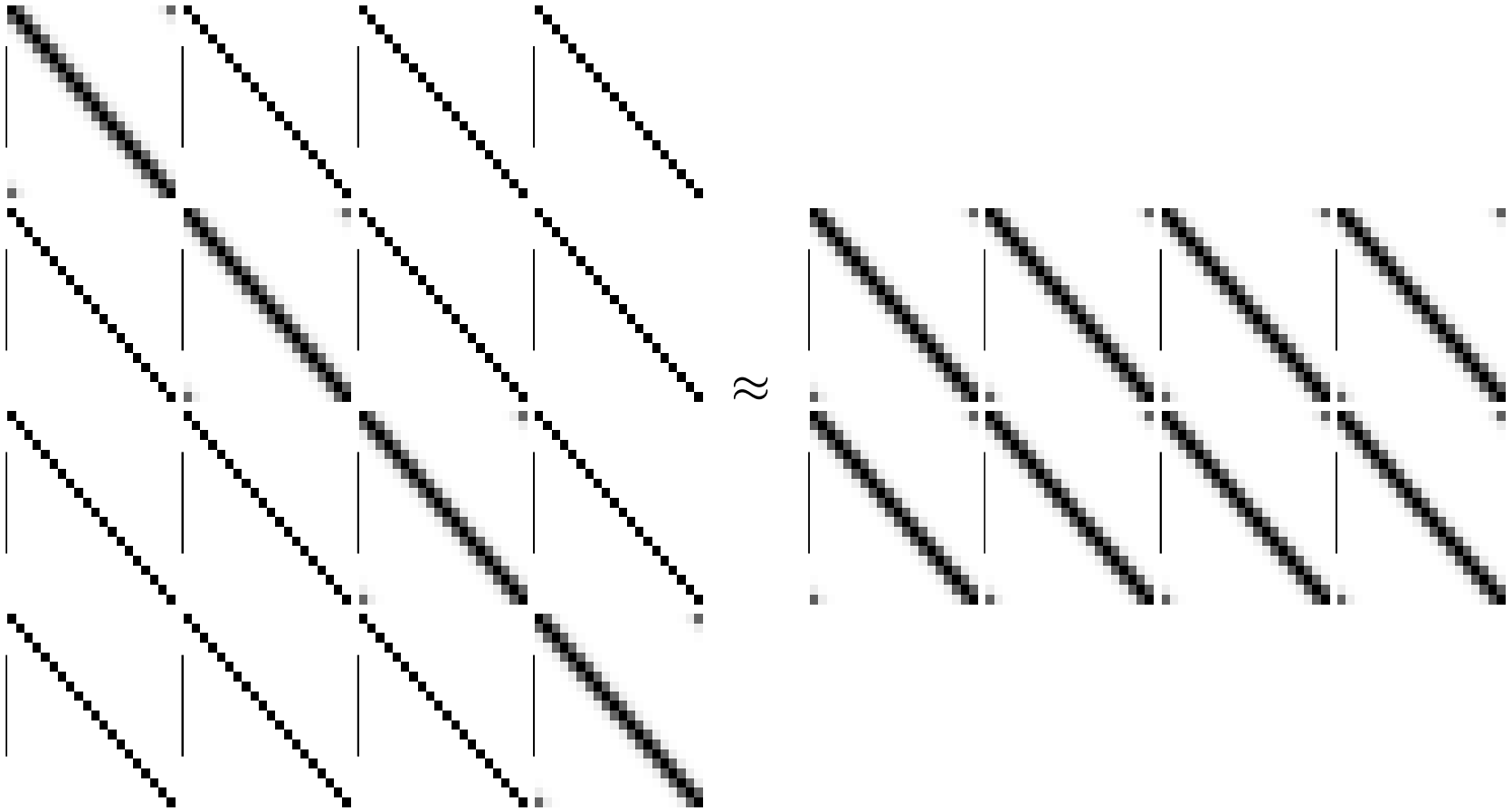
$$+\mathcal{I}_1 \otimes \mathcal{I}_2 \otimes \frac{\partial^2}{\partial x_3^2} \otimes \mathcal{I}_4$$

$$+\mathcal{I}_1 \otimes \mathcal{I}_2 \otimes \mathcal{I}_3 \otimes \frac{\partial^2}{\partial x_4^2}$$

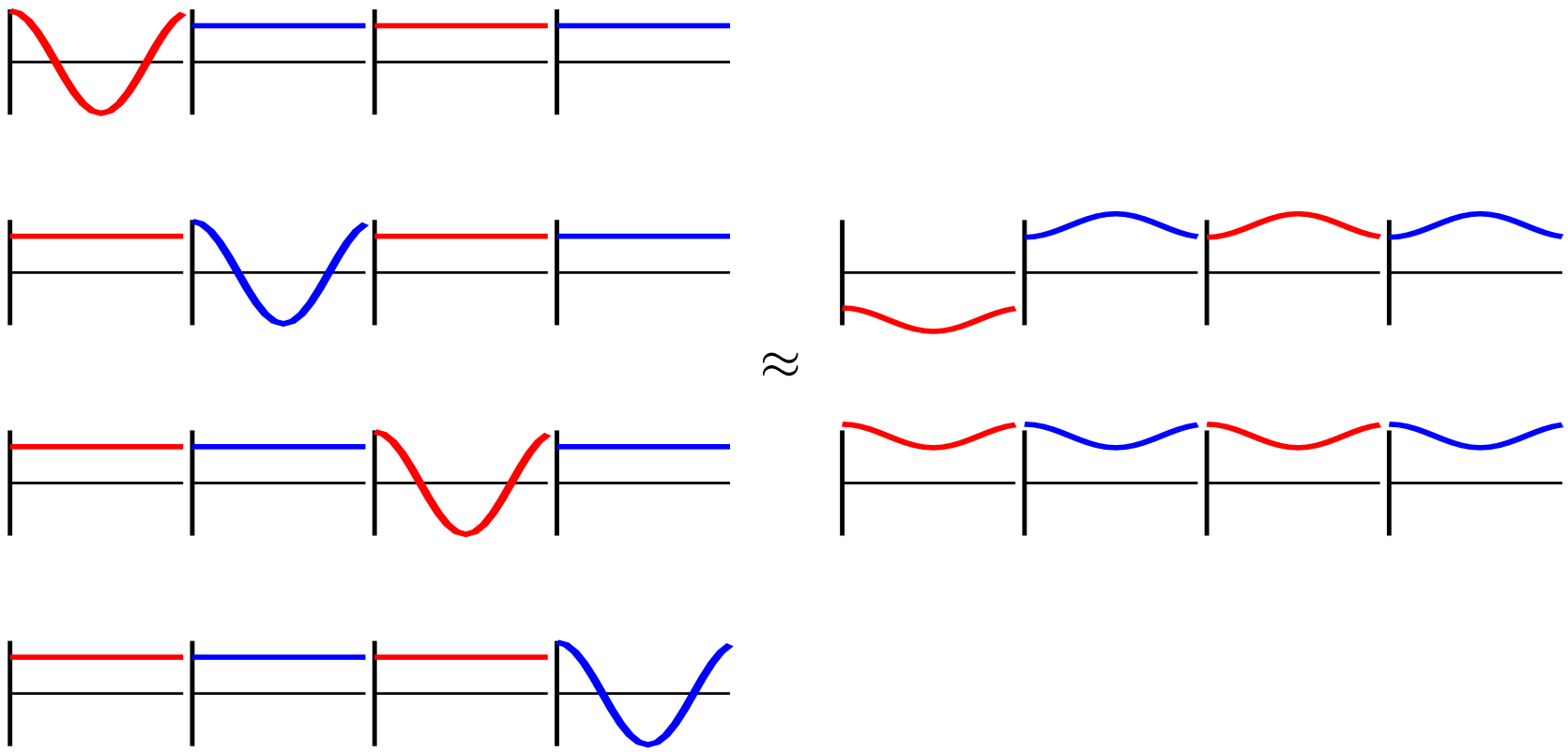
$=$



Would you have guessed?



Would you have guessed?



The Laplacian

Let

$$\mathcal{G}(t) = \bigotimes_{i=1}^{d} \left(\mathcal{I}_i + t \frac{\partial^2}{\partial x_i^2} \right)$$

and note that

$$\mathcal{G}'(t) = \Delta + \mathcal{O}(t)$$

so $\mathcal{G}'(0) = \Delta$. Using an appropriate finite difference formula of order r, we approximate

$$\mathcal{G}'(0) \approx \sum_{j=1}^{r} \alpha_j \mathcal{G}(t_j) \equiv \sum_{j=1}^{r} \alpha_j \bigotimes_{i=1}^{d} \left(\mathcal{I}_i + t_j \frac{\partial^2}{\partial x_i^2} \right),$$

thus providing a separation rank r approximation for Δ.

The Laplacian

Theorem Let $\mathcal{A}_i$ be a fixed, bounded operator $\mathcal{A}$ acting in the direction i, $\epsilon > 0$ be the error bound, and $0 < \mu \ll 1$ be the machine unit roundoff. Assuming $\mu d \|\mathcal{A}\| < \epsilon$, we can represent $\sum_i^d \mathcal{A}_i$ to within ϵ in the operator norm with separation rank

$$r = \mathcal{O}(\frac{\log(d\|\mathcal{A}\|/\epsilon)}{\log(1/\mu) - \log(d\|\mathcal{A}\|/\epsilon)}).$$

- We must first make the Laplacian bounded (by e.g. discretizing).

- The main behavior is $r \approx \log(d)$.

A linear system in dimension d

Given a matrix

$$\mathbb{A} = \sum_{l=1}^{r_{\mathbb{A}}} s_l^{\mathbb{A}} \mathbb{A}_1^l \otimes \mathbb{A}_2^l \otimes \cdots \otimes \mathbb{A}_d^l,$$

and a right-hand-side vector

$$\mathbf{G} = \sum_{l=1}^{r_{\mathbf{G}}} s_l^{\mathbf{G}} \mathbf{G}_1^l \otimes \mathbf{G}_2^l \otimes \cdots \otimes \mathbf{G}_d^l,$$

we attempt to solve the linear system $\mathbb{A}\mathbf{F} = \mathbf{G}$ for $\mathbf{F}$ of the form

$$\mathbf{F} = \sum_{l=1}^{r_{\mathbf{F}}} s_l^{\mathbf{F}} \mathbf{F}_1^l \otimes \mathbf{F}_2^l \otimes \cdots \otimes \mathbf{F}_d^l,$$

by iteratively refining an approximate solution.

Alternating Least Squares

Recast $\mathbb{A}\mathbf{F} = \mathbf{G}$ as the minimization of the error $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|$.

Start with random $\mathbf{F}$ with $r_{\mathbf{F}} = 1$.

- Loop while $\|\mathbb{A}\mathbf{F} - \mathbf{G}\| > \epsilon$:
  - Loop $k = 1, \ldots, d$:
    - Fix $\{\mathbf{F}_i^l\}_{i \neq k}$ and solve a linear least-squares problem for $\mathbf{F}_k^l$ (and $s_l^{\mathbf{F}}$) to minimize $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|$.
  - If the relative decrease in $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|$ is too small, then increase $r_{\mathbf{F}}$.

Form the matrix $\mathbb{B}$ with entries

$$B((\hat{j},\hat{l}),(j,l)) = \sum_{l_1}^{r_{\mathbb{A}}} \sum_{l_2}^{r_{\mathbb{A}}} s_{l_1}^{\mathbb{A}} s_{l_2}^{\mathbb{A}} \left(\sum_{j'}^{M} A_k^{l_1}(j',\hat{j}) A_k^{l_2}(j',j) \right) \prod_{i \neq k} \langle \mathbb{A}_i^{l_1} \mathbf{F}_i^{l}, \mathbb{A}_i^{l_2} \mathbf{F}_i^{\hat{l}} \rangle,$$

and the vector $\mathbf{b}$ with entries

$$b((\hat{j},\hat{l})) = \sum_{l_1}^{r_{\mathbb{A}}} s_{l_1}^{\mathbb{A}} \sum_{l_3}^{r_{\mathbf{G}}} s_{l_3}^{\mathbf{G}} \left(\sum_{j'}^{M} A_k^{l_1}(j',\hat{j}) g_k^{l_3}(j') \right) \prod_{i \neq k} \langle \mathbf{G}_i^{l_3}, \mathbb{A}_i^{l_1} \mathbf{F}_i^{\hat{l}} \rangle.$$

The normal equations for the direction k become

$$\mathbb{B}\mathbf{c} = \mathbf{b},$$

which we solve for $\mathbf{c} = c((j,l))$. Once we find $\mathbf{c}$, we compute $s_l^{\mathbf{F}} = \|c(\cdot,l)\|$ and set $F_k^l(j) = c(j,l)/s_l^{\mathbf{F}}$.

Assuming that M is the largest parameter, the dominant computational cost is $\mathcal{O}(dr_{\mathbf{F}}^3 M^3)$.

A Random Numerical Example

We chose dimension $d = 20$, resolution $M = 30$, and precision $\epsilon = 10^{-6}$.

We generated a random $\mathbf{F}_0$ with $r_{\mathbf{F}_0} = 2$ and a random $\mathbb{A}$ with $r_{\mathbb{A}} = 6$, and formed $\mathbf{G} = \mathbb{A}\mathbf{F}_0$.

In a typical run, the algorithm performed 50 iterations at $r_{\mathbf{F}} = 1$, 'decided' that it was stuck, performed 18 iterations at $r_{\mathbf{F}} = 2$, and achieved $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|/\|\mathbf{G}\| = 9.96 \cdot 10^{-7}$.

We also measured the error $\|\mathbf{F} - \mathbf{F}_0\|/\|\mathbf{F}_0\| = 1.08 \cdot 10^{-6}$.

Separation rank $r_{\mathbf{F}} = 2$ or 3 is achieved routinely in under 30 seconds on a laptop with a 1.7GHz processor.

A Numerical Example with the Laplacian

With everything else unchanged, we chose $\mathbb{A}$ to be a discretization of the Laplacian Δ, which has $r_{\mathbb{A}} = d = 20$ using the form

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_d^2}.$$

We estimated $\|\mathbb{A}\| \approx 1.17 \cdot 10^5$.

The algorithm performed only 5 iterations at $r_{\mathbf{F}} = 1$, 'decided' that it was stuck with error $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|/\|\mathbf{G}\| = 0.231$, performed 5 iterations at $r_{\mathbf{F}} = 2$, and achieved $\|\mathbb{A}\mathbf{F} - \mathbf{G}\|/\|\mathbf{G}\| = 3.94 \cdot 10^{-8}$. The total time used was less than one minute.

A Numerical Example with the Laplacian: Conditioning

The error $\|\mathbf{F} - \mathbf{F}_0\|/\|\mathbf{F}_0\| = 2.99 \cdot 10^{-8}$ is much **better** than expected since the linear system is singular.

Two effects save us:

1. The constraint on $r_{\mathbf{F}}$ discourages extra terms, such as a constant term that is in the nullspace of Δ.

2. A random vector like $\mathbf{F}_0$ is expected to have a small ($\sim M^{-d/2} \approx 10^{-15}$) projection on the constant vector.

To disable and so verify these effects, we had to choose $r_{\mathbf{F}} = 3$, $d = 2$, and $M = 10$.

A Test of the Separation Rank of the Inverse Laplacian

We kept $\mathbb{A}$ as a discretization of Δ, constructed $\mathbf{G}$ randomly with separation rank $r_{\mathbf{G}} = 1$, and attempted to find $\mathbf{F}$.
The separation rank of $\mathbf{F}$ cannot exceed that of the Δ^{-1}.

Using 2 iterations at each value of $r_{\mathbf{F}}$, a selection of errors is:

$r_{\mathbf{F}}$	$\|\mathbb{A}\mathbf{F} - \mathbf{G}\|/\|\mathbf{G}\|$	seconds
1	$2.5 \cdot 10^{-2}$	13
3	$3.8 \cdot 10^{-3}$	84
5	$6.8 \cdot 10^{-4}$	213
9	$8.0 \cdot 10^{-5}$	789
13	$8.4 \cdot 10^{-6}$	2048
19	$9.5 \cdot 10^{-7}$	6121

The Inverse Laplacian

Lemma: For any $0 < \delta < 1$, and $0 < \epsilon \leq \frac{1}{2}$ there exist positive numbers τ_l and σ_l such that for all $\delta \leq y \leq 1$

$$\left| \frac{1}{y^2} - \sum_{l=1}^{r} \sigma_l e^{-\tau_l y^2} \right| \leq \frac{\epsilon}{y^2} \,,$$

with

$$r = \log \epsilon^{-1} \left[c_0 + c_1 \log \epsilon^{-1} + c_2 \log \delta^{-1} \right],$$

where c_0, c_1, and c_2 are constants independent of ϵ and δ.

The Inverse Laplacian

We can obtain a separated representation for Δ^{-1},
valid on the annulus $(\sum_{i=1}^{d} \xi_i^2)^{1/2} \in [\delta D, D]$ in Fourier space,
by substituting

$$-D^2 y^2 = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_d^2}.$$

We obtain

$$\Delta^{-1} \approx \frac{-1}{D^2} \sum_{l=1}^{r} \sigma_l \exp(\frac{\tau_l}{D^2} \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2}) = \frac{-1}{D^2} \sum_{l=1}^{r} \sigma_l \bigotimes_{i=1}^{d} \exp(\frac{\tau_l}{D^2} \frac{\partial^2}{\partial x_i^2}).$$

Concluding Remarks

There is much more to this story.

- Numerical Analysis in high dimensions is happening.
- Many interesting things may now be possible.
- Nature cooperates.