The Nonequispaced FFT and its Applications
A Mini Course at Helsinki University of Technology

Jens Keiner[1] Stefan Kunis[2] Antje Vollrath[1]

[1]Institut für Mathematik, Universität zu Lübeck

[2]Fakultät für Mathematik, Technische Universität Chemnitz

Program

Part I – Fourier Analysis and the FFT

Stefan, Monday, 14:15 – 16:00, Room U322

Part II – Orthogonal Polynomials

Jens, Tuesday, 12:15 – 14:00, Room U141 (Lecture Hall F)

Practice Session: 14:30 – 16:00, Room Y339b (Basics and Matlab Hands-On)

Part III – Fast Polynomial Transforms and Applications

Jens, Wednesday, 12:15 – 14:00, Room U345

Practice Session: 14:30 – 16:00, Room Y338c (C Library Hands-On)

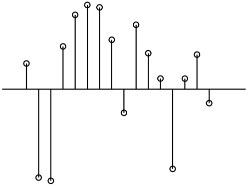
Part IV – Fourier Transforms on the Rotation Group

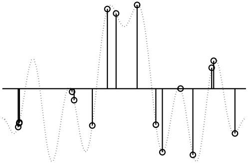
Antje, Thursday, 14:15 – 16:00, Room U322

Part V – High Dimensions and Reconstruction

Stefan, Friday, 10:15 – 12:00, Room U322

Part I – Fourier Analysis and the NFFT





Outline

1. Introduction
2. Fourier Analysis
3. Nonequispaced FFT
4. Summary

Introduction

"The Fast Fourier transform (FFT) is one of the truly great computational developments of this century. It has changed the face of science and engineering so that it is not an exaggeration to say that life as we know it would be very different without FFT."

[Charles Van Loan]

(a) Gauss 1805

(b) Runge 1903

(c) Lanczos 1942

(d) Tukey 1965

Introduction

Fast computation of

$$f_j = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi \mathrm{i} k x_j}, \qquad j = 0, \ldots, M-1,$$

and

$$\hat{h}_k = \sum_{j=0}^{M-1} f_j \, \mathrm{e}^{+2\pi \mathrm{i} k x_j}, \qquad k = -N/2, \ldots, N/2-1,$$

for $x_j \in [-1/2, 1/2)$.

In short: $\mathbf{f} = \mathbf{A}\hat{\mathbf{f}}$, $\hat{\mathbf{h}} = \mathbf{A}^{\mathsf{H}}\mathbf{f}$ with $\mathbf{A} \in \mathbb{C}^{M \times N}$, $a_{j,k} = \mathrm{e}^{-2\pi \mathrm{i} k x_j}$.

Discrete Fourier transform: $x_j = j/N$, $j = -N/2, \ldots, N/2-1$, computation by FFT costs $\mathcal{O}(N \log N)$ flops.

Torus $\mathbb{T} = [-1/2, 1/2)$, Hilbert space $L^2(\mathbb{T})$,

$$(f, g)_{L^2(\mathbb{T})} := \int_{-1/2}^{1/2} f(x)\, \overline{g(x)}\, \mathrm{d}x, \qquad \|f\|_{L^2(\mathbb{T})} = (f, f)^{\frac{1}{2}}$$

orthogonality of $\mathrm{e}_k(x) := \mathrm{e}^{2\pi\mathrm{i}kx} = \cos 2\pi kx + \mathrm{i} \sin 2\pi kx$

$$\begin{aligned}(\mathrm{e}_j, \mathrm{e}_k)_{L^2(\mathbb{T})} &= \int_{-1/2}^{1/2} \mathrm{e}^{2\pi\mathrm{i}jx}\, \mathrm{e}^{-2\pi\mathrm{i}kx}\, \mathrm{d}x = \int_{-1/2}^{1/2} \mathrm{e}^{2\pi\mathrm{i}(j-k)x}\, \mathrm{d}x \\ &= \frac{1}{2\pi\mathrm{i}(j-k)}\, \mathrm{e}^{2\pi\mathrm{i}(j-k)x}\, \Big|_{-1/2}^{1/2} = 0 \qquad (j \neq k)\end{aligned}$$

Fourier Analysis - Fourier series

$f \in L^2(\mathbb{T})$ can represented by the complex Fourier series

$$f(x) \;=\; \sum_{k=-\infty}^{\infty} c_k(f)\, \mathrm{e}^{2\pi \mathrm{i} k x}$$

with Fourier coefficients

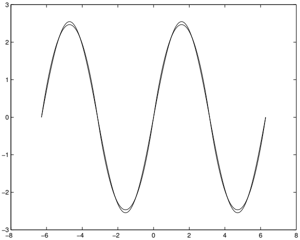
$$\begin{aligned} c_k(f) \;&=\; (f, \mathrm{e}_k)_{L^2(\mathbb{T})} \\ &=\; \int_{-1/2}^{1/2} f(x) \mathrm{e}^{-2\pi \mathrm{i} k x}\ \mathrm{d}x \end{aligned}$$

Theorem: Let f be a continuous one-periodic function with

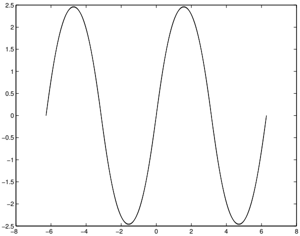
$$\sum_{k=-\infty}^{\infty} |c_k(f)| < \infty\,,$$

then the Fourier series converges absolutely and uniformly.

Fourier Analysis - Fourier series, Example



(a) $N = 1$



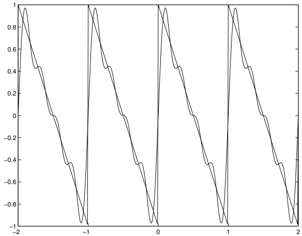
(b) $N = 2$

Fourier series

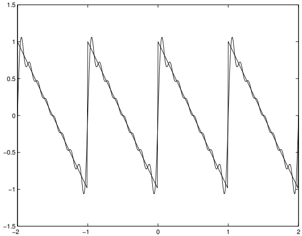
$$\frac{\pi}{8}\sum_{k=1}^{N}\frac{\sin((2k-1)x)}{(2k-1)^3}$$

of the 2π–periodic function

$$f(x) = \begin{cases} x(\pi - x) & x \in [0, \pi) \\ (\pi - x)(2\pi - x) & x \in [\pi, 2\pi) \end{cases}$$



(a) $N = 4$



(b) $N = 8$

Fourier series

$$\sum_{k=1}^{N} \frac{2}{\pi} \frac{\sin(2\pi k x)}{k}$$

of the 1–periodic function

$$f(x) = -2x + 1$$

Linearity

$$\begin{aligned} c_k(f+g) &= c_k(f) + c_k(g) \\ c_k(\lambda f) &= \lambda\, c_k(f) \end{aligned}$$

Symmetry

$$\begin{aligned} c_k(h) &= c_{-k}(f), \qquad && h(x) := f(-x) \\ c_k(h) &= \overline{c_{-k}(f)}, \qquad && h(x) := \overline{f(x)} \end{aligned}$$

Shift and modulation

$$\begin{aligned} c_k(h) &= \mathrm{e}^{2\pi \mathrm{i} x_0 k}\, c_k(f), \qquad && h(x) := f(x - x_0) \\ c_k(h) &= c_{k-k_0}(f), \qquad && h(x) := \mathrm{e}^{-2\pi \mathrm{i} k_0 x} f(x) \end{aligned}$$

Differentiation

$$c_k(h) = (2\pi \mathrm{i} k)^m\, c_k(f), \qquad h(x) := f^{(m)}(x)$$

Hilbert space $\ell^2(\mathbb{Z})$,

$$(\mathbf{a}, \mathbf{b})_{\ell^2} := \sum_{k \in \mathbb{Z}} a_k \overline{b_k}, \qquad \|\mathbf{a}\|_{\ell^2} := (\mathbf{a}, \mathbf{b})_{\ell^2}^{\frac{1}{2}}.$$

For $f, g \in L^2(\mathbb{T})$ and

$$\mathbf{c}(f) := (c_k(f))_{k \in \mathbb{Z}}, \quad \mathbf{c}(g) := (c_k(g))_{k \in \mathbb{Z}} \in \ell^2$$

we have

$$(\mathbf{c}(f), \mathbf{c}(g))_{\ell^2} = (f, g)_{L^2(\mathbb{T})}, \quad \|\mathbf{c}(f)\|_{\ell^2} = \|f\|_{L^2(\mathbb{T})}.$$

Theorem: Let f be a one-periodic function with absolutely convergent Fourier series with Fourier coefficients

$$c_k(f) := \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x)\, \mathrm{e}^{-2\pi \mathrm{i} k x} \mathrm{d}x.$$

The discrete Fourier coefficients

$$\hat{f}_k := \sum_{j=-N/2}^{N/2-1} f\left(\frac{j}{N}\right) \mathrm{e}^{-2\pi \mathrm{i} jk/N}$$

fulfil the aliasing relation

$$\frac{1}{N}\hat{f}_k = c_k(f) + \sum_{\substack{r\in\mathbb{Z}\\ r\neq 0}} c_{k+rN}(f).$$

Proof: Substitute the Fourier series f into the definition of $\hat{f}_k$.

$$\begin{aligned}\frac{1}{N}\hat{f}_k &= \frac{1}{N}\sum_{j=-N/2}^{N/2-1}\sum_{l\in\mathbb{Z}} c_l(f)\,\mathrm{e}^{2\pi\mathrm{i}lj/N}\,\mathrm{e}^{-2\pi\mathrm{i}jk/N}\\ &= \sum_{l\in\mathbb{Z}} c_l(f)\,\frac{1}{N}\sum_{j=-N/2}^{N/2-1}\mathrm{e}^{2\pi\mathrm{i}j(l-k)/N}\\ &= \sum_{l\in\mathbb{Z}} c_l(f)\,\frac{1}{N}\sum_{j=0}^{N-1}\mathrm{e}^{2\pi\mathrm{i}j(l-k)/N}.\end{aligned}$$

The assertion follows from

$$\frac{1}{N}\sum_{j=0}^{N-1}\mathrm{e}^{2\pi\mathrm{i}j(l-k)/N} = \begin{cases}1 & \text{if} \quad \frac{l-k}{N}\in\mathbb{Z}\\ 0 & \text{otherwise.}\end{cases}$$

Lemma:

$$\frac{1}{N}\sum_{j=0}^{N-1} \mathrm{e}^{2\pi \mathrm{i} j(l-k)/N} = \begin{cases} 1 & \text{if} \quad \frac{l-k}{N} \in \mathbb{Z} \\ 0 & \text{otherwise.} \end{cases}$$

In case $\frac{l-k}{N} \in \mathbb{Z}$, this holds because all terms in the sum are 1.
In case $\frac{l-k}{N} \notin \mathbb{Z}$, we apply $\sum_{k=0}^{N-1} q^k = \frac{q^N - 1}{q-1}$. This yields

$$\sum_{j=0}^{N-1} \mathrm{e}^{2\pi \mathrm{i} j(l-k)/N} = \frac{\mathrm{e}^{2\pi \mathrm{i}(l-k)} - 1}{\mathrm{e}^{2\pi \mathrm{i}(l-k)/N} - 1} = \frac{0}{\mathrm{e}^{2\pi \mathrm{i}(l-k)/N} - 1} = 0$$

since $\mathrm{e}^{2\pi \mathrm{i}(l-k)/N} \neq 1$.

Corollary: If f is a one-periodic function of which only the lowest N Fourier coefficients are non-zero, i.e.,

$$f(x) = \sum_{k=-N/2}^{N/2-1} c_k(f)\, \mathrm{e}^{2\pi \mathrm{i} k x},$$

then the approximation $\frac{1}{N}\hat{f}_k$ for the Fourier coefficients is exact for $k = -N/2, \dots, N/2-1$.

Definitions:

Index-set

$$I_N^d := [-N/2, N/2)^d \cap \mathbb{Z}^d$$

torus

$$\mathbb{T}^d := \left[-\frac{1}{2}, \frac{1}{2}\right)^d$$

inner product

$$\mathbf{kx} = k_1 x_1 + k_2 x_2 + \ldots + k_d x_d$$

Fourier Analysis - Fourier series, d-variate

Theorem: Let $f \in L^2(\mathbb{T}^d)$ be a one-periodic function with absolutely convergent Fourier series

$$f(\mathbf{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^d} c_{\boldsymbol{k}}(f)\, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}}$$

with Fourier coefficients

$$c_{\boldsymbol{k}}(f) := \int_{\mathbb{T}^d} f(\mathbf{x})\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}} \mathrm{d}\mathbf{x}.$$

If the $c_{\boldsymbol{k}}(f)$ are approximated by the discrete Fourier coefficients

$$\hat{f}_{\boldsymbol{k}} := \sum_{\boldsymbol{j} \in I_N^d} f\left(\frac{\mathbf{j}}{N}\right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{j}\boldsymbol{k}/N},$$

then the following aliasing relation holds

$$c_{\boldsymbol{k}}(f) \approx \frac{1}{N^d} \hat{f}_{\boldsymbol{k}} = c_{\boldsymbol{k}}(f) + \sum_{\substack{\boldsymbol{r} \in \mathbb{Z}^d \\ \boldsymbol{r} \neq \mathbf{0}}} c_{\boldsymbol{k}+N\boldsymbol{r}}(f).$$

Fourier Analysis - DFT

The discrete Fourier transform (DFT) of $\mathbf{f} = (f_j)_{j=-N/2}^{N/2-1} \in \mathbb{C}^N$ is

$$\hat{f}_k := \sum_{j=-N/2}^{N/2-1} f_j \, \mathrm{e}^{-2\pi \mathrm{i} jk/N} \quad (k = -N/2, \dots, N/2-1) \, .$$

Matrix–vector form, $\hat{\mathbf{f}} := (\hat{f}_j)_{j=-N/2}^{N/2}$, $\mathbf{F}_N := \left(\mathrm{e}^{-2\pi \mathrm{i} kj/N}\right)_{j,k=-N/2}^{N/2-1}$ is

$$\hat{\mathbf{f}} = \mathbf{F}_N \mathbf{f}.$$

Theorem: The inverse discrete Fourier transform (IDFT) of the vector $\hat{\mathbf{f}} \in \mathbb{C}^N$ is given by

$$f_j = \frac{1}{N} \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{2\pi \mathrm{i} jk/N} \quad (j = -N/2, \dots, N/2-1) \, .$$

Proof: Substitute one sum into the other, i.e.,

$$\begin{aligned}
\sum_{j=-N/2}^{N/2-1} f_j\, \mathrm{e}^{-2\pi \mathrm{i} jk/N} &= \sum_{j=-N/2}^{N/2-1} \frac{1}{N} \sum_{r=-N/2}^{N/2-1} \hat{f}_r\, \mathrm{e}^{2\pi \mathrm{i} jr/N}\, \mathrm{e}^{-2\pi \mathrm{i} jk/N} \\
&= \frac{1}{N} \sum_{r=-N/2}^{N/2-1} \hat{f}_r \left(\sum_{j=-N/2}^{N/2-1} \mathrm{e}^{2\pi \mathrm{i} jr/N}\, \mathrm{e}^{-2\pi \mathrm{i} jk/N} \right) \\
&= \hat{f}_k .
\end{aligned}$$

Again, the identity follows from the orthogonality relation

$$\sum_{j=-N/2}^{N/2-1} \mathrm{e}^{2\pi \mathrm{i} j(r-k)/N} = \begin{cases} N & \text{if } r = k, \\ 0 & \text{otherwise.} \end{cases}$$

Observation: $\mathbf{F}_N$ contains only N different values, since $\mathrm{e}^{-2\pi \mathrm{i}k/N}$ is N periodic

(Unshifted) Fourier matrix $F_N = (\mathrm{e}^{-2\pi \mathrm{i}jk/N})_{j,k=0}^{N-1}$

Examples:

$$\mathbf{F}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \mathbf{F}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix},$$

$$\mathbf{F}_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \theta & \theta^2 \\ 1 & \theta^2 & \theta \end{pmatrix}, \quad \theta := \mathrm{e}^{-2\pi \mathrm{i}/3}.$$

Fourier Analysis - DFT, FFT

The DFT takes $\mathcal{O}(N^2)$ floating point operations (flops).

The Fast Fourier Transform (FFT) takes only $\mathcal{O}(N \log N)$ flops by using a divide-and-conquer approach. Reduce one problem of size N to two problems of size $N/2$ at the cost of $\mathcal{O}(N)$, i.e.,

$$\mathbf{F}_N = \begin{bmatrix} \mathit{odd} - \mathit{even} \\ \mathit{permutation} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{N/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{N/2} & \mathbf{I}_{N/2} \\ \mathbf{W} & -\mathbf{W} \end{bmatrix}$$

where $\mathbf{W} = \text{diag}(1, \mathrm{e}^{-2\pi\mathrm{i}1/N}, \mathrm{e}^{-2\pi\mathrm{i}2/N}, \ldots, \mathrm{e}^{-2\pi\mathrm{i}(N/2-1)/N})$.

Software: FFTW package by Frigo & Johnson (www.fftw.org).

Compute DFT of size $N = 2^n$, $n \in \mathbb{N}$, $w_N := \mathrm{e}^{-2\pi\mathrm{i}/N}$

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j w_N^{jk} \qquad (k = 0, \dots, N-1).$$

Decimation-in-frequency (Sande-Tukey), divide the above sum

$$\hat{f}_k = \sum_{j=0}^{\frac{N}{2}-1} f_j w_N^{jk} + \sum_{j=0}^{\frac{N}{2}-1} f_{\frac{N}{2}+j} w_N^{\left(\frac{N}{2}+j\right)k} \qquad (k = 0, \dots, N-1).$$

Case 1: Even $k = 2l$

$$\hat{f}_{2l} = \sum_{j=0}^{\frac{N}{2}-1} f_j w_N^{2jl} + \sum_{j=0}^{\frac{N}{2}-1} f_{\frac{N}{2}+j} w_N^{\left(\frac{N}{2}+j\right)2l} \qquad \left(l = 0, \ldots, \frac{N}{2} - 1\right)$$

note that

$$w_N^{\left(\frac{N}{2}+j\right)2l} = \mathrm{e}^{-2\pi \mathrm{i} l} \mathrm{e}^{-2\pi \mathrm{i} jl/(N/2)} = w_{\frac{N}{2}}^{jl}$$

hence

$$\begin{aligned} \hat{f}_{2l} &= \sum_{j=0}^{\frac{N}{2}-1} f_j w_{\frac{N}{2}}^{jl} + \sum_{j=0}^{\frac{N}{2}-1} f_{\frac{N}{2}+j} w_{\frac{N}{2}}^{jl}, \\ &= \sum_{j=0}^{\frac{N}{2}-1} \left(f_j + f_{\frac{N}{2}+j}\right) w_{\frac{N}{2}}^{jl} \qquad \left(l = 0, \ldots, \frac{N}{2} - 1\right) \end{aligned}$$

Case 2: Odd $k = 2l + 1$

$$\hat{f}_{2l+1} = \sum_{j=0}^{\frac{N}{2}-1} f_j w_N^{j(2l+1)} + \sum_{j=0}^{\frac{N}{2}-1} f_{\frac{N}{2}+j} w_N^{\left(\frac{N}{2}+j\right)(2l+1)} \quad (l = 0, \dots, \frac{N}{2} - 1)$$

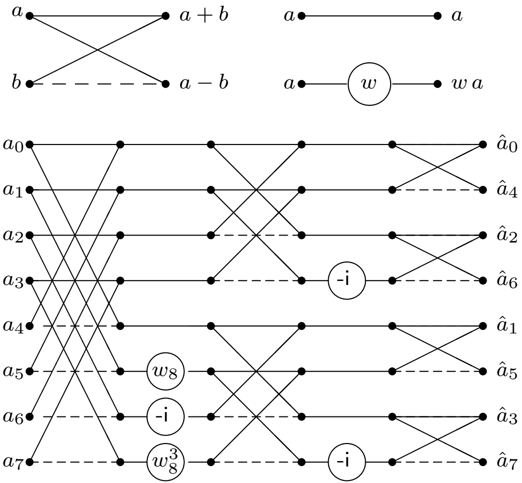
note that

$$w_N^{\left(\frac{N}{2}+j\right)(2l+1)} = \mathrm{e}^{-2\pi\mathrm{i}\frac{N}{2}\frac{2l+1}{N}} w_N^j w_{\frac{N}{2}}^{jl} = -w_N^j w_{\frac{N}{2}}^{jl}$$

hence (with twiddle factors w_N^j)

$$\begin{aligned} \hat{f}_{2l+1} &= \sum_{j=0}^{\frac{N}{2}-1} f_j w_{\frac{N}{2}}^{jl} w_N^j - \sum_{j=0}^{\frac{N}{2}-1} f_{\frac{N}{2}+j} w_{\frac{N}{2}}^{jl} w_N^j, \\ &= \sum_{j=0}^{\frac{N}{2}-1} \left(f_j - f_{\frac{N}{2}+j} \right) w_N^j w_{\frac{N}{2}}^{jl} \quad (l = 0, \dots, \frac{N}{2} - 1) \end{aligned}$$

DFT(N) takes N additions, $\frac{N}{2}$ multiplications, and 2 DFT$(\frac{N}{2})$ in total: $\mathcal{O}(N \log N)$ flops



Banach space $L^p = L^p(\mathbb{R})$, $1 \leq p \leq \infty$,

$$||f||_{L^p} := \left(\int_{-\infty}^{\infty} |f(x)|^p \, \mathsf{d}x \right)^{1/p}$$

The Fourier transform $\hat{f} : \mathbb{R} \to \mathbb{C}$ of $f \in L^1(\mathbb{R})$ is given by

$$\hat{f}(v) := \int_{-\infty}^{\infty} f(t) \mathrm{e}^{-2\pi \mathrm{i} v t} \, \mathsf{d}t$$

Characteristic function, $L > 0$

$$f(x) := \begin{cases} 1 & \text{if} \quad |x| < L, \\ \frac{1}{2} & \text{if} \quad x = \pm L, \\ 0 & \quad \text{else.} \end{cases}$$

$$\begin{aligned} \hat{f}(v) &= \int_{-L}^{L} \mathrm{e}^{-2\pi \mathrm{i} v x} \, \mathrm{d}x = -\frac{1}{2\pi \mathrm{i} v} \mathrm{e}^{-2\pi \mathrm{i} v x} \Big|_{-L}^{L} \\ &= 2L \operatorname{sinc}(2\pi L v) \end{aligned}$$

with the Sinc-function

$$\operatorname{sinc} x := \begin{cases} \frac{\sin x}{x} & \text{if} \quad x \neq 0, \\ 1 & \text{if} \quad x = 0. \end{cases}$$

Gaussian $f(x) = \mathrm{e}^{-x^2}$ has the Fourier transform

$$\hat{f}(v) = \sqrt{\pi}\mathrm{e}^{-v^2\pi^2}.$$

Proof:

$$\begin{aligned}
\hat{f}(v) &= \int_{-\infty}^{\infty} \mathrm{e}^{-x^2}\mathrm{e}^{-2\pi \mathrm{i} v x}\,\mathrm{d}x \\
&= \mathrm{e}^{-\pi^2 v^2}\int_{-\infty}^{\infty} \mathrm{e}^{-(x+\pi \mathrm{i} v)^2}\,\mathrm{d}x \\
&= \mathrm{e}^{-\pi^2 v^2}\int_{-\infty}^{\infty} \mathrm{e}^{-z^2}\,\mathrm{d}z \\
&= \sqrt{\pi}\mathrm{e}^{-\pi^2 v^2}.
\end{aligned}$$

$$\begin{aligned}
\left(\int_{-\infty}^{\infty} \mathrm{e}^{-t^2}\ \mathsf{d}t\right)^2 &= \int_{-\infty}^{\infty} \mathrm{e}^{-x^2}\ \mathsf{d}x \int_{-\infty}^{\infty} \mathrm{e}^{-y^2}\ \mathsf{d}y \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathrm{e}^{-(x^2+y^2)}\ \mathsf{d}x\ \mathsf{d}y \\
&= \int_{0}^{2\pi} \int_{0}^{\infty} \mathrm{e}^{-r^2} r\ \mathsf{d}r\ \mathsf{d}\varphi \\
&= \frac{1}{2} \int_{0}^{\infty} \mathrm{e}^{-s}\ \mathsf{d}s \int_{0}^{2\pi} \mathsf{d}\varphi \\
&= -\pi \mathrm{e}^{-s}\Big|_{0}^{\infty} \\
&= \pi.
\end{aligned}$$

Theorem: Let $\varphi \in L^2(\mathbb{R}) \cap L^1(\mathbb{R})$ given, such that the periodisation

$$\tilde{\varphi}(x) := \sum_{r\in\mathbb{Z}} \varphi(x+r)$$

has an uniformly convergent Fourier series

$$\tilde{\varphi}(x) = \sum_{k\in\mathbb{Z}} c_k(\tilde{\varphi}) \mathrm{e}^{2\pi \mathrm{i} k x}, \qquad c_k(\tilde{\varphi}) := \int\limits_{-1/2}^{1/2} \tilde{\varphi}(x) \mathrm{e}^{-2\pi \mathrm{i} k x} \, \mathsf{d}x \, .$$

Then

$$\int\limits_{\mathbb{R}} \varphi(x) \mathrm{e}^{-2\pi \mathrm{i} k x} \, \mathsf{d}x =: \hat{\varphi}(k) = c_k(\tilde{\varphi}).$$

Proof:

$$
\begin{aligned}
c_k(\tilde{\varphi}) &= \int\limits_{-1/2}^{1/2} \sum_{r\in\mathbb{Z}} \varphi(x+r)\mathrm{e}^{-2\pi\mathrm{i}kx}\,\mathrm{d}x \\
&= \sum_{\boldsymbol{r}\in\mathbb{Z}} \int\limits_{-1/2}^{1/2} \varphi(x+r)\mathrm{e}^{-2\pi\mathrm{i}kx}\,\mathrm{d}x \\
&= \sum_{r\in\mathbb{Z}} \int\limits_{-1/2+r}^{1/2+r} \varphi(y)\mathrm{e}^{-2\pi\mathrm{i}ky}\underbrace{\mathrm{e}^{2\pi\mathrm{i}kr}}_{=1}\,\mathrm{d}y \\
&= \int\limits_{\mathbb{R}} \varphi(y)\mathrm{e}^{-2\pi\mathrm{i}ky}\,\mathsf{d}y \\
&= \hat{\varphi}(k)\,.
\end{aligned}
$$

Theorem: Let $\varphi \in L^2(\mathbb{R}^d) \cap L^1(\mathbb{R}^d)$ given, such that

$$\tilde{\varphi}(\mathbf{x}) := \sum_{\boldsymbol{r} \in \mathbb{Z}^d} \varphi(\mathbf{x} + \mathbf{r})$$

has an uniformly convergent Fourier series

$$\tilde{\varphi}(\mathbf{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^d} c_{\boldsymbol{k}}(\tilde{\varphi}) \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}}, \qquad c_{\boldsymbol{k}}(\tilde{\varphi}) := \int_{\mathbb{T}^d} \tilde{\varphi}(\mathbf{x}) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}} \, \mathrm{d}\mathbf{x} \,.$$

Then

$$\int_{\mathbb{R}^d} \varphi(\mathbf{x}) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\boldsymbol{x}} \; \mathrm{d}\mathbf{x} =: \hat{\varphi}(\mathbf{k}) = c_{\boldsymbol{k}}(\tilde{\varphi}).$$

Proof:

$$
\begin{aligned}
c_{\boldsymbol{k}}(\tilde{\varphi}) &= \int_{\mathbb{T}^d} \sum_{\boldsymbol{r}\in\mathbb{Z}^d} \varphi(\mathbf{x}+\mathbf{r})\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}}\,\mathrm{d}\mathbf{x} \\
&= \sum_{\boldsymbol{r}\in\mathbb{Z}^d} \int_{\mathbb{T}^d} \varphi(\mathbf{x}+\mathbf{r})\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}}\,\mathrm{d}\mathbf{x} \\
&= \sum_{\boldsymbol{r}\in\mathbb{Z}^d} \int_{-1/2+r_1}^{1/2+r_1} \cdots \int_{-1/2+r_d}^{1/2+r_d} \varphi(\mathbf{y})\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}(\boldsymbol{y}-\boldsymbol{r})}\,\mathrm{d}\mathbf{y} \\
&= \int_{\mathbb{R}^d} \varphi(\mathbf{y})\mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{y}}\,\mathrm{d}\mathbf{y} \\
&= \hat{\varphi}(\mathbf{k})\,.
\end{aligned}
$$

freq. \ time	continuous	discrete
continuous	Fourier transform	"semidiscrete" Fourier transform
discrete	Fourier series	discrete Fourier transform

Fourier transform on $\mathbb{R}$

forward: $$\hat{f}(v) = \int_{-\infty}^{\infty} f(x)\mathrm{e}^{-2\pi \mathrm{i} v x}\,\mathsf{d}x$$

inverse: $$f(x) = \int_{-\infty}^{\infty} \hat{f}(v)\mathrm{e}^{2\pi \mathrm{i} v x}\,\mathsf{d}x$$

periodicity: none

"semidiscrete" Fourier transform

forward: $\hat{f}(v) = \sum_{j=-\infty}^{\infty} f(j)\mathrm{e}^{-2\pi \mathrm{i} v j}$

inverse: $f(j) = \int_{-1/2}^{1/2} \hat{f}(v)\mathrm{e}^{2\pi \mathrm{i} v j}\,\mathsf{d}v$

periodicity: $\hat{f}(v) = \hat{f}(v+1)$

Fourier series

$$
\begin{aligned}
&\text{forward:} && c_k(f) = \int\limits_{-1/2}^{1/2} f(x)\mathrm{e}^{-2\pi \mathrm{i} k x}\,\mathsf{d}x \\
&\text{inverse:} && f(x) = \sum_{k=-\infty}^{\infty} c_k(f)\mathrm{e}^{2\pi \mathrm{i} k x}\,\mathsf{d}x \\
&\text{periodicity:} && f(x) = f(x+1)
\end{aligned}
$$

discrete Fourier transform (DFT)

forward: $$\hat{f}_k = \sum_{j=0}^{N-1} f_j \mathrm{e}^{-2\pi \mathrm{i} jk/N}$$

inverse: $$f_j = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}_k \mathrm{e}^{2\pi \mathrm{i} jk/N}$$

periodicity: $\hat{f}_k = \hat{f}_{k+rN}\,; f_j = f_{j+rN}$

Nonequispaced FFT - $d = 1$

Fast computation of

$$f_j = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi\mathrm{i}kx_j}, \qquad j = 0, \dots, M-1,$$

and

$$\hat{h}_k = \sum_{j=0}^{M-1} f_j \, \mathrm{e}^{+2\pi\mathrm{i}kx_j}, \qquad k = -N/2, \dots, N/2-1,$$

for $x_j \in [-1/2, 1/2)$.

In short: $\mathbf{f} = \mathbf{A}\hat{\mathbf{f}}$, $\hat{\mathbf{h}} = \mathbf{A}^{\mathsf{H}}\mathbf{f}$ with $\mathbf{A} \in \mathbb{C}^{M \times N}$, $a_{j,k} = \mathrm{e}^{-2\pi\mathrm{i}kx_j}$.

Nonequispaced FFT - Taylor expansion $d = 1$

Evaluate at the nodes x_j

$$f(x) = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi \mathrm{i} k x}.$$

1. compute Fourier coefficients of derivatives, $l = 0, \ldots, m-1$,

$$\hat{g}_k^{[l]} = (-2\pi \mathrm{i} k)^l \hat{f}_k$$

2. compute m oversampled ($n = \sigma N$) fast Fourier transforms

$$g^{[l]} = \mathsf{FFT}(\hat{g}^{[l]})$$

3. expand $f(x_j) \approx p^{[j',m]}(x_j)$ about its nearest grid point

$$p^{[j',m]}(x_j) = g\left(\frac{j'}{n}\right) + g'\left(\frac{j'}{n}\right)\left(x_j - \frac{j'}{n}\right) + \frac{g''\left(\frac{j'}{n}\right)}{2}\left(x_j - \frac{j'}{n}\right)^2 + \ldots$$

Bernstein type inequality

$$|f'(x)| = |\sum_{k=-N/2}^{N/2-1} \hat{f}_k(-2\pi \mathrm{i} k)\mathrm{e}^{-2\pi \mathrm{i} k x}| \le \pi N \sum_{k=-N/2}^{N/2-1} |\hat{f}_k|.$$

Error estimate

$$\begin{aligned} |f(x_j) - p^{[j',m]}(x_j)| &\le \|f^{(m)}\|_\infty \frac{|x_j - j'/n|^m}{m!} \\ &\le (\pi N)^m \sum_{k=-N/2}^{N/2-1} |\hat{f}_k| \cdot \frac{1}{2^m \sigma^m N^m m!} \\ &\le C_{\sigma,m} \|\hat{\mathbf{f}}\|_1. \end{aligned}$$

Takes $\mathcal{O}((N\log N + M)|\log\varepsilon|)$ floating point operations.

Nonequispaced FFT - Taylor expansion $d = 1$

Matrix factorisation $\mathbf{P} \in \mathbb{C}^{M \times N}$,

$$\mathbf{P} = [\mathbf{X}^0 \mathbf{X}^1 \dots \mathbf{X}^{m-1}][\mathbf{F}_n \dots \mathbf{F}_n][\mathbf{D}^0 \mathbf{D}^1 \dots \mathbf{D}^{m-1}]$$

with "diagonal" matrices $\mathbf{D} \in \mathbb{C}^{N \times N}$, $d^l_{k,k} = (-2\pi \mathrm{i} k)^l$, and $\mathbf{X} \in \mathbb{R}^{M,n}$, $x^l_{j,j'} = (x_j - j'/n)^l / l!$.

Error estimate

$$|a_{j,k} - p_{j,k}| \leq C_{\sigma,m}.$$

Normwise error estimate

$$\|\mathbf{A} - \mathbf{P}\|_2 \leq C_{\sigma,m} \sqrt{MN}.$$

Nonequispaced FFT - Ansatz

Evaluate at the nodes x_j

$$f(x) = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi \mathrm{i} k x}.$$

Set $n := \sigma N$, oversampling factor $\sigma > 1$, and approximate f by

$$s_1(x) := \sum_{l=-n/2}^{n/2-1} g_l \; \tilde{\varphi}(x - \frac{l}{n})$$

where $\tilde{\varphi} = \sum_{r \in \mathbb{Z}} \varphi(\cdot - r)$ is a 1–periodic window function.
Switching to the frequency domain

$$s_1(x) = \sum_{k=-\infty}^{\infty} c_k(s_1) \, \mathrm{e}^{-2\pi \mathrm{i} k x}, \qquad c_k(s_1) := \int_{-1/2}^{1/2} s_1(x) \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x$$

Nonequispaced FFT - Ansatz

$$
\begin{aligned}
c_k(s_1) &= \int_{-1/2}^{1/2} s_1(x) \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x \\
&= \int_{-1/2}^{1/2} \sum_{l=-n/2}^{n/2-1} g_l \, \tilde{\varphi}(x - \frac{l}{n}) \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x \\
&= \sum_{l=-n/2}^{n/2-1} g_l \int_{-1/2}^{1/2} \tilde{\varphi}(\underbrace{x - \frac{l}{n}}_{y}) \mathrm{e}^{2\pi \mathrm{i} k x} \, \mathrm{d}x \\
&= \sum_{l=-n/2}^{n/2-1} g_l \mathrm{e}^{-2\pi \mathrm{i} k l/n} \int_{-1/2-l/n}^{1/2-l/n} \tilde{\varphi}(y) \mathrm{e}^{2\pi \mathrm{i} k y} \, \mathrm{d}y \\
&= \hat{g}_k \, c_k(\tilde{\varphi})
\end{aligned}
$$

Nonequispaced FFT - Ansatz

Hence

$$s_1(x) = \sum_{k=-\infty}^{\infty} \hat{g}_k \, c_k(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i} k x}$$

with discrete Fourier coefficients of g_l

$$\hat{g}_k := \sum_{l=-n/2}^{n/2-1} g_l \, \mathrm{e}^{2\pi \mathrm{i} k l / n}$$

and Fourier coefficients of $\tilde{\varphi}$

$$c_k(\tilde{\varphi}) = \int_{-1/2}^{1/2} \tilde{\varphi}(x) \mathrm{e}^{-2\pi \mathrm{i} k v} \, \mathrm{d}x \quad = \hat{\varphi}(k).$$

Nonequispaced FFT - First approximation

Compare $f(x) = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi \mathrm{i} k x}$ and

$$s_1(x) = \sum_{k=-\infty}^{\infty} \hat{g}_k \, c_k(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i} k x}$$

$$= \sum_{r=-\infty}^{\infty} \sum_{k=-n/2}^{n/2-1} \underbrace{\hat{g}_{k+nr}}_{\hat{g}_k} \, c_{k+nr}(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i}(k+nr)x}$$

$$= \sum_{k=-n/2}^{n/2-1} \hat{g}_k \, c_k(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i} k x} + \sum_{\substack{r=-\infty \\ r \neq 0}}^{\infty} \sum_{k=-n/2}^{n/2-1} \hat{g}_k \, c_{k+nr}(\tilde{\varphi}) \, \mathrm{e}^{-2\pi \mathrm{i}(k+nr)x}$$

Set

$$\hat{g}_k := \begin{cases} \hat{f}_k / c_k(\tilde{\varphi}) & k = -N/2, \ldots, N/2-1, \\ 0 & k = -n/2, \ldots, -N/2-1; N/2, \ldots, n/2-1. \end{cases}$$

Nonequispaced FFT - Second approximation

Suppose $\tilde{\varphi}$ is small outside $[-m/n, m/n]$ with $m \ll n$,

$$s_1(x) = \sum_{l=-n/2}^{n/2-1} g_l \, \tilde{\varphi}(x - \frac{l}{n}).$$

Approximate φ by compactly supported function

$$\psi(x) := \begin{cases} \varphi(x) & \text{if } x \in [-m/n, m/n], \\ 0 & \text{else}, \end{cases} \qquad \tilde{\psi}(x) := \sum_{r \in \mathbb{Z}} \psi(x+r).$$

For $j = -M/2, \ldots, M/2-1$ compute

$$f(x_j) \approx s_1(x_j) \approx s(x_j) := \sum_{l=\lfloor x_j n \rfloor - m}^{\lceil x_j n \rceil + m} g_l \, \tilde{\psi}\left(x_j - \frac{l}{n}\right).$$

Nonequispaced FFT - Algorithm

1. For $k = -N/2, \dots, N/2-1$ compute
$$\hat{g}_k := \hat{f}_k / c_k(\tilde{\varphi}).$$
2. For $l = -n/2, \dots, n/2-1$ compute by FFT of size n
$$g_l := \frac{1}{n} \sum_{k=-N/2}^{N/2-1} \hat{g}_k \, \mathrm{e}^{-2\pi \mathrm{i} kl/n}.$$
3. For $j = 0, \dots, M-1$ compute
$$s(x_j) := \sum_{l=\lfloor x_j n \rfloor - m}^{\lceil x_j n \rceil + m} g_l \, \tilde{\psi}\left(x_j - \frac{l}{n}\right).$$

Floating point operations

$$\mathcal{O}(N + n \log n + (2m+1)M) = \mathcal{O}(n \log n + mM)$$

Nonequispaced FFT - Interpretation

Convolution based algorithm

1. deconvolve f with the window function

$$\hat{g} \leftarrow \hat{f}/\hat{\varphi}$$

2. compute one oversampled fast Fourier transform

$$g \leftarrow \mathsf{FFT}(\hat{g})$$

3. convolve with the window function and evaluate

$$f(x_j) \leftarrow (g \star \tilde{\varphi})(x_j)$$

Joint approximation

$$\mathrm{e}^{-2\pi\mathrm{i}kx} \approx \frac{1}{n\hat{\varphi}\,(k)} \sum_{l=\lfloor xn \rfloor - m}^{\lceil xn \rceil + m} \tilde{\varphi}\,(x - l/n)\,\mathrm{e}^{-2\pi\mathrm{i}kl/n}$$

Nonequispaced FFT- Matrix-vector form

$\mathbf{A}$ is factorised approximately as $\mathbf{A} \approx \mathbf{BFD}$

1. $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix:

$$\mathbf{D} := \text{diag}\left(\frac{1}{n\, c_k(\tilde{\varphi})}\right)_{k=-N/2}^{N/2-1}$$

2. $\mathbf{F} \in \mathbb{C}^{n \times N}$ is a truncated Fourier matrix:

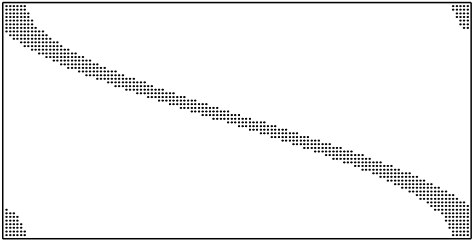
$$\mathbf{F} := \left(\mathrm{e}^{-2\pi \mathrm{i} kl/n}\right)_{l=-n/2,\ k=-N/2}^{n/2-1 \quad N/2-1}$$

3. $\mathbf{B} \in \mathbb{R}^{M \times n}$ is a sparse matrix with $2m+1$ non-zeros per row:

$$\mathbf{B} := (b_{j,l})_{j=0\ \ l=-n/2}^{M-1\ n/2-1}$$

where

$$b_{j,l} = \begin{cases} \tilde{\psi}\left(x_j - \frac{l}{n}\right) & \text{if } l \in \{\lfloor x_j n \rfloor - m, \ldots, \lceil x_j n \rceil + m\} \\ 0 & \text{otherwise.} \end{cases}$$



Spy of the matrix $\mathbf{B} \in \mathbb{R}^{64\times 128}$, Legendre nodes x_j, cut-off $m = 5$.

The factorisation that was derived for $\mathbf{A}$ allows us to derive

$$\mathbf{A}^{\mathsf{H}} \approx \mathbf{D}^{\top}\mathbf{F}^{\mathsf{H}}\mathbf{B}^{\top}.$$

Nonequispaced FFT- Multivariate version

Of course, a d-variate FFT is available.

Use the window function $\varphi : \mathbb{R}^d \to \mathbb{R}$,

$$\varphi(\mathbf{x}) := \prod_{t=1}^{d} \varphi(x_t), \qquad \mathbf{x} = (x_1, x_2, \dots, x_d)^{\mathsf{T}}$$

and note

$$\hat{\varphi}(\mathbf{k}) = \prod_{t=1}^{d} \hat{\varphi}(k_t) \qquad \mathbf{k} = (k_1, \dots, k_d)^{\mathsf{T}}.$$

Nonequispaced FFT - Error estimates

The two approximations yield

$$E(x_j) := |f(x_j) - s(x_j)| \leq \; E_{\mathrm{a}}(x_j) + E_{\mathrm{t}}(x_j)$$

with aliasing and truncation error

$$E_{\mathrm{a}}(x_j) := |f(x_j) - s_1(x_j)|, \quad E_{\mathrm{t}}(x_j) := |s_1(x_j) - s(x_j)|$$

Theorem: Let $\|\hat{\mathbf{f}}\|_1 := \sum_{k=-N/2}^{N/2-1} |\hat{f}_k|$, then

$$E_{\mathrm{a}}(x_j) \leq \|\hat{\mathbf{f}}\|_1 \max_{k \in I_N^1} \sum_{\substack{r=-\infty \\ r \neq 0}}^{\infty} \left| \frac{c_{k+nr}(\tilde{\varphi})}{c_k(\tilde{\varphi})} \right|$$

and

$$E_{\mathrm{t}}(x_j) \leq \frac{\|\hat{\mathbf{f}}\|_1}{n} \max_{k \in I_N^1} \frac{1}{|c_k(\tilde{\varphi})|} \sum_{\left|x_j + \frac{r}{n}\right| \geq \frac{m}{n}} \left| \varphi\left(x_j + \frac{r}{n}\right) \right|$$

Proof:

$$\begin{aligned}
E_{\mathrm{a}}(x_j) &:= |f(x_j) - s_1(x_j)| \\
&= \left| \sum_{\substack{r=-\infty \\ r\neq 0}}^{\infty} \sum_{k=-n/2}^{n/2-1} \hat{g}_k \, c_{k+nr}(\tilde{\varphi}) \, \mathrm{e}^{-2\pi\mathrm{i}(k+nr)x} \right| \\
&= \left| \sum_{\substack{r=-\infty \\ r\neq 0}}^{\infty} \sum_{k=-N/2}^{N/2-1} \frac{\hat{f}_k}{c_k(\tilde{\varphi})} \, c_{k+nr}(\tilde{\varphi}) \, \mathrm{e}^{-2\pi\mathrm{i}(k+nr)x} \right| \\
&\leq \sum_{k=-N/2}^{N/2-1} \left| \hat{f}_k \right| \sum_{\substack{r=-\infty \\ r\neq 0}}^{\infty} \left| \frac{c_{k+nr}(\tilde{\varphi})}{c_k(\tilde{\varphi})} \right| \\
&\leq \|\hat{\mathbf{f}}\|_1 \max_{k=-N/2,\ldots,N/2-1} \sum_{\substack{r=-\infty \\ r\neq 0}}^{\infty} \left| \frac{c_{k+nr}(\tilde{\varphi})}{c_k(\tilde{\varphi})} \right| .
\end{aligned}$$

Nonequispaced FFT - Error estimates

Due to $g_l = \frac{1}{n} \sum\limits_{k \in I_N^1} \frac{\hat{f}_k}{\hat{\varphi}(k)} \mathrm{e}^{-2\pi \mathrm{i} kl/n}$, we have

$$
\begin{aligned}
E_{\mathrm{t}}(x_j) &= \left| \sum_{l=-n/2}^{n/2-1} g_l \left(\tilde{\varphi}\left(x_j - \frac{l}{n}\right) - \tilde{\psi}\left(x_j - \frac{l}{n}\right)\right) \right| \\
&\le \frac{1}{n} \left| \sum_{l \in I_n^1} \sum_{k \in I_N^1} \frac{\hat{f}_k}{\hat{\varphi}(k)} \mathrm{e}^{-2\pi \mathrm{i} kl/n} \left(\tilde{\varphi}\left(x_j - \frac{l}{n}\right) - \tilde{\psi}\left(x_j - \frac{l}{n}\right)\right) \right| \\
&\le \frac{1}{n} \left| \sum_{k \in I_N^1} \frac{\hat{f}_k}{\hat{\varphi}(k)} \sum_{l \in I_n^1} \left(\tilde{\varphi}\left(x_j - \frac{l}{n}\right) - \tilde{\psi}\left(x_j - \frac{l}{n}\right)\right) \mathrm{e}^{-2\pi \mathrm{i} kl/n} \right| \\
&\le \max_{k \in I_N^1} \frac{\|\hat{\mathbf{f}}\|_1}{n\,|\hat{\varphi}(k)|} \left| \sum_{l \in I_n^1} (\tilde{\varphi}\left(x_j - \frac{l}{n}\right) - \tilde{\psi}\left(x_j - \frac{l}{n}\right)) \mathrm{e}^{-2\pi \mathrm{i} kl/n} \right|.
\end{aligned}
$$

We proceed by

$$
\begin{aligned}
& \sum_{l\in I_n^1} \left(\tilde{\varphi}\left(x_j - \frac{l}{n}\right) - \tilde{\psi}\left(x_j - \frac{l}{n}\right)\right) \mathrm{e}^{-2\pi \mathrm{i} kl/n} \\
= & \sum_{l\in I_n^1} \left(\sum_{r\in\mathbb{Z}} \varphi\left(x_j - \frac{l}{n} + r\right) \right. \\
& \left. -\varphi\left(x_j - \frac{l}{n} + r\right) \ \chi_{[-\frac{m}{n},\frac{m}{n}]}\left(x_j - \frac{l}{n} + r\right) \right) \mathrm{e}^{-2\pi \mathrm{i} kl/n} \\
= & \sum_{r\in\mathbb{Z}} \left(\varphi\left(x_j + \frac{r}{n}\right) - \varphi\left(x_j + \frac{r}{n}\right) \ \chi_{[-\frac{m}{n},\frac{m}{n}]}\left(x_j + \frac{r}{n}\right)\right) \mathrm{e}^{-2\pi \mathrm{i} kr/n} \\
= & \sum_{|x_j + \frac{r}{n}| \ge \frac{m}{n}} \varphi\left(x_j + \frac{r}{n}\right) \mathrm{e}^{-2\pi \mathrm{i} kr/n}.
\end{aligned}
$$

Finally

$$\begin{aligned} E_{\mathrm{t}}(x_j) &\leq \frac{\|\hat{\mathbf{f}}\|_1}{n} \max_{k \in I_N} \frac{1}{|\hat{\varphi}(k)|} \left| \sum_{\left|x_j + \frac{r}{n}\right| \geq \frac{m}{n}} \varphi\left(x_j + \frac{r}{n}\right) \mathrm{e}^{-2\pi \mathrm{i} k r / n} \right| \\ &\leq \frac{\|\hat{\mathbf{f}}\|_1}{n} \max_{k \in I_N^1} \frac{1}{|\hat{\varphi}(k)|} \sum_{\left|x_j + \frac{r}{n}\right| \geq \frac{m}{n}} \left| \varphi\left(x_j + \frac{r}{n}\right) \right| . \end{aligned}$$

Corollary: For non-negative, even, and monotone decreasing φ:

$$E_{\mathrm{t}}(x_j) \leq \frac{\|\hat{\mathbf{f}}\|_1}{n} \max_{k \in I_N^1} \frac{2}{|\hat{\varphi}(k)|} \left(\varphi\left(\frac{m}{n}\right) + \int_m^{\infty} \varphi\left(\frac{x}{n}\right) \, \mathrm{d}x \right) .$$

$E_{\mathrm{t}} = 0$ [G. Beylkin, G. Steidl]

Centered cardinal B-spline of order $m \in \mathbb{N}$

$$M_1(x) := \begin{cases} 1 & \text{if} \quad x \in [-1/2, 1/2), \\ 0 & \text{else}, \end{cases}$$

$$M_{m+1}(x) := \int_{-1/2}^{1/2} M_m(x-t)\,\mathrm{d}t$$

have $\mathrm{supp} M_m = [-m/2, m/2]$ and

$$\hat{M}_1(v) = \int_{-1/2}^{1/2} \mathrm{e}^{-2\pi \mathrm{i} v x}\,\mathrm{d}x = \mathrm{sinc}(\pi v).$$

Lemma: $\hat{M}_m(k) = (\mathrm{sinc}(\pi k))^m$ for $m \in \mathbb{N}$.

Proof: By induction

$$\begin{aligned}
\hat{M}_{m+1}(k) &= \int_{\mathbb{R}} M_{m+1}(x) \mathrm{e}^{-2\pi \mathrm{i} x k} \,\mathrm{d}x \\
&= \int_{\mathbb{R}} \int_{-1/2}^{1/2} M_m(\underbrace{x-t}_{y}) \mathrm{e}^{-2\pi \mathrm{i} x k} \,\mathrm{d}t \,\mathrm{d}x \\
&= \int_{-1/2}^{1/2} \underbrace{\int_{\mathbb{R}} M_m(y) \mathrm{e}^{-2\pi \mathrm{i} y k} \,\mathrm{d}y}_{\hat{M}_m(k)} \, \mathrm{e}^{-2\pi \mathrm{i} t k} \,\mathrm{d}t \\
&= (\mathrm{sinc}(\pi k))^m \, \mathrm{sinc}(\pi k) \\
&= (\mathrm{sinc}(\pi k))^{m+1} \,.
\end{aligned}$$

Lemma: For $0 < u < 1$ and $m \in \mathbb{N}$ it holds that

$$\sum_{r \in \mathbb{Z} \setminus \{0\}} \left(\frac{u}{u+r} \right)^{2m} < \frac{4m}{2m-1} \left(\frac{u}{u-1} \right)^{2m} .$$

Proof: For $r \geq 0$ holds $\left(\frac{u}{u+r}\right)^{2m} \leq \left(\frac{u}{u-r}\right)^{2m}$ and

$$\begin{aligned}
\sum_{r \in \mathbb{Z}} \left(\frac{u}{u+r} \right)^{2m} &\leq 1 + 2 \left(\frac{u}{u-1} \right)^{2m} + 2 \sum_{r=2}^{\infty} \left(\frac{u}{u-r} \right)^{2m} \\
&\leq 1 + 2 \left(\frac{u}{u-1} \right)^{2m} + 2 \int_1^{\infty} \left(\frac{u}{u-x} \right)^{2m} \mathrm{d}x \\
&= 1 + 2 \left(\frac{u}{u-1} \right)^{2m} \left(1 + \frac{1-u}{2m-1} \right) \\
&< 1 + 2 \left(\frac{u}{u-1} \right)^{2m} \left(1 + \frac{1}{2m-1} \right) .
\end{aligned}$$

Theorem: Let $f(x_j)$, $j = 0, \ldots, M-1$ be computed by the NFFT with $\varphi(x) := M_{2m}(nx)$ and $n := \sigma N$ $(\sigma > 1)$. Then the approximation error can be estimated

$$E_\infty := \max_{j \in I_M^1} E(x_j) \le \|\hat{\mathbf{f}}\|_1 \frac{4m}{2m-1} \left(\frac{1}{2\sigma - 1}\right)^{2m}$$

where $\hat{\mathbf{f}} := (\hat{f}_k)_{k \in I_N^1}$.

Proof: $E_{\mathrm{t}} = 0$ since

$$\text{supp} \quad \varphi \subseteq [-\frac{m}{\sigma N}, \frac{m}{\sigma N}].$$

Moreover

$$
\begin{aligned}
\hat{\varphi}(k) = c_k(\tilde{\varphi}) &= \int_{\mathbb{R}} \varphi(x) \mathrm{e}^{-2\pi \mathrm{i} k x} \, \mathsf{d}x \\
&= \int_{\mathbb{R}} M_{2m}(\underbrace{\sigma N x}_{y}) \mathrm{e}^{-2\pi \mathrm{i} k x} \, \mathsf{d}x \\
&= \frac{1}{\sigma N} \int_{\mathbb{R}} M_{2m}(y) \mathrm{e}^{2\pi \mathrm{i} k y/(\sigma N)} \, \mathsf{d}y \\
&= \frac{1}{\sigma N} \left(\operatorname{sinc} \frac{\pi k}{\sigma N} \right)^{2m} .
\end{aligned}
$$

Since

$$\begin{aligned} \sigma N \hat{\varphi}(k + r\sigma N) &= \left(\frac{\sin(k\pi/(\sigma N))}{k\pi/(\sigma N) + r\pi} \right)^{2m} \\ &= \left(\frac{\sin(k\pi/(\sigma N))}{k\pi/(\sigma N)} \right)^{2m} \left(\frac{k\pi/(\sigma N)}{k\pi/(\sigma N) + r\pi} \right)^{2m} \\ &= \sigma N \hat{\varphi}(k) \left(\frac{k/(\sigma N)}{k/(\sigma N) + r} \right)^{2m} \end{aligned}$$

and due to the above Lemma

$$E_\infty \le \|\hat{\mathbf{f}}\|_1 \, \frac{4m}{2m-1} \max_{k \in I_N^1} \frac{(k/(\sigma N))^{2m}}{(k/(\sigma N) - 1)^{2m}}.$$

Since $u/(u-1)$ increases for $u \in [0, 1/2]$, the assertion follows for $k = N/2$.

$E_{\mathrm{t}} \approx E_{\mathrm{a}}$ [Dutt & Rokhlin, G. Steidl]

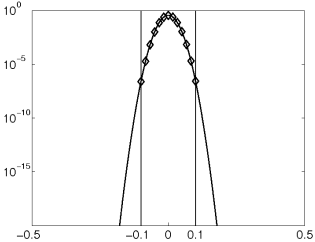
Theorem: Let $f(x_j)$, $j = 0, \ldots, M-1$, be computed by the NFFT with $\sigma \geq 3/2$ and

$$\varphi(x) := \frac{1}{\sqrt{\pi b}} \mathrm{e}^{-(\sigma N x)^2/b},$$

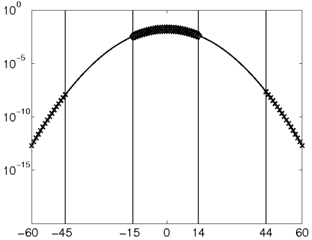
where $b := \frac{2\sigma}{2\sigma-1}\frac{m}{\pi}$. Then the approximation error can be estimated

$$E_\infty \leq 4\mathrm{e}^{-m\pi\left(1-\frac{1}{2\sigma-1}\right)} \|\hat{\mathbf{f}}\|_1.$$

Nonequispaced FFT- Error estimates, Gaussian



(a) Gaussian window function $\varphi(x) = c\mathrm{e}^{-\alpha x^2}$, sampled on $2m+1$ nodes $-\frac{m}{n}, , \dots , \frac{m}{n}$ ($\diamond$).



(b) Fourier transform $\hat{\varphi}$ with "pass" ($\diamond$), "transition", and "stop" band ($\times$).

Parameters are set to $N = 30$, $\sigma = 2$, $n = 60$, $m = 6$.

Nonequispaced FFT- Error estimates, Gaussian

In place of a proof: $c_k(\tilde{\varphi}) = \hat{\varphi}(k) = \frac{1}{\sigma N}\mathrm{e}^{-\left(\frac{\pi k}{\sigma N}\right)^2 b}$
The errors can be estimated by

$$\begin{aligned} E_{\mathrm{a}}(x_j) &\leq \|\hat{\mathbf{f}}\|_1 \mathrm{e}^{-b\pi^2\left(1-\frac{1}{\sigma}\right)} \\ &\cdot \left(1 + \frac{\sigma}{(2\sigma-1)b\pi^2} + \mathrm{e}^{-2b\pi^2/\sigma}\left(1 + \frac{\sigma}{(2\sigma+1)b\pi^2}\right)\right), \end{aligned}$$

$$E_{\mathrm{t}}(x_j) \leq \|\hat{\mathbf{f}}\|_1 \cdot \mathrm{e}^{-b\pi^2\left(\left(\frac{m}{b\pi}\right)^2 - \left(\frac{1}{2\sigma}\right)^2\right)} \cdot \frac{2}{\sqrt{\pi b}}\left(1 + \frac{b}{2m}\right).$$

Choose $b = \frac{2\sigma}{2\sigma-1}\frac{m}{\pi}$ to get

$$\left(\frac{m}{b\pi}\right)^2 - \left(\frac{1}{2\sigma}\right)^2 = 1 - \frac{1}{\sigma}.$$

Nonequispaced FFT- Error estimates, summary

$E_{\mathrm{a}} = 0$: Sinc function [Potts], Kaiser-Bessel function [Fourmont, Potts]

Theorem: Approximation error

$$E(x_j) := |f(x_j) - s(x_j)| \leq C(\sigma, m)\|\hat{\mathbf{f}}\|_1,$$

with

$$C(\sigma, m) := \begin{cases} 4\left(\frac{1}{2\sigma-1}\right)^{2m} & \text{B-spline,} \\ 4\,\mathrm{e}^{-m\pi(1-1/(2\sigma-1))} & \text{Gaussian,} \\ \frac{3}{m-1}\left(\frac{\sigma}{2\sigma-1}\right)^{2m-1} & \text{Sinc,} \\ 4\pi(\sqrt{m}+m)\sqrt[4]{1-\frac{1}{\sigma}}\mathrm{e}^{-m2\pi\sqrt{1-1/\sigma}} & \text{Kaiser-Bessel.} \end{cases}$$

Corollary: Precision ε for fixed $\sigma > 1$ when $m \sim |\log \varepsilon|$.

Nonequispaced FFT - Summary

Fast Fourier tranform (FFT) [Cooley, Tukey 1965; Frigo, Johnson 1997-] computes

$$f_{\mathbf{j}} = \sum_{k \in I_N^d} \hat{f}_{\mathbf{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \mathbf{k}\mathbf{j}/N}, \qquad \mathbf{j} \in I_N^d,$$

in $\mathcal{O}\left(N^d \log N\right)$ flops, see www.fftw.org.

Nonequispaced FFT [Dutt, Rokhlin 1993; Beylkin 1995-; Potts, Steidl, Tasche 1997-; Greengard, Lee 2004; Keiner, Kunis, Potts 2002-] computes

$$f_j = \sum_{k \in I_N^d} \hat{f}_{\mathbf{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \mathbf{k}\mathbf{x}_j}, \qquad j = 0, \ldots, M-1,$$

in $\mathcal{O}\left(N^d \log N + |\log \varepsilon|^d \, M\right)$ flops, see www.tu-chemnitz.de/∼potts/nfft.

Nonequispaced FFT - Matlab

NDFT:

```
x=rand(M,1)-1/2;
f_hat=rand(N,1)+i*rand(N,1);
f=exp(-2*pi*i*x*(-N/2):(N/2-1))*f_hat;
```

NFFT, Taylor expansion

```
freq=-2*pi*i*(-(N/2):(N/2-1))';
ix=round(n*(x+0.5));
dx=x-(ix/n-0.5);
ix=mod(ix,n)+1;
for l=0:m
  g_hat=[zeros((n-N)/2,1);f_hat.*(freq.^l);...
         zeros((n-N)/2,1)];
  g=fftshift(fft(fftshift(g_hat)));
  f=f+g(ix).*(dx.^l)/prod(1:l);
end;
```

NFFT, Gaussian window

```
freq=(-(N/2):(N/2-1))';
b=2*sigma*m / ((2*sigma-1)*pi);
inv_phi_hat=exp(b*(pi*freq/n).^2);
g_hat=[zeros((n-N)/2,1);f_hat.*(inv_phi_hat);...
       zeros((n-N)/2,1)];
g=fft(fftshift(g_hat));
for j=1:M
  c_j=n*x(j);
  u_j=floor(c_j-m);
  o_j=ceil(c_j+m);
  supp_j=mod(u_j:o_j,n)+1;
  psi_j=(pi*b)^(-1/2)*exp(-(n*x(j)-(u_j:o_j)).^2/b);
  f(j)=psi_j*g(supp_j);
end;
```

Nonequispaced FFT - Matlab

Computing nonequispaced DFT and FFT

```
ft_opt1.method='direct';
f1=ndft(f_hat,x,N,ft_opt1,'notransp');

ft_opt2.method='gaussian';
ft_opt2.m=6;
ft_opt2.sigma=2;
f2=nfft(f_hat,x,N,ft_opt2,'notransp');
```

and adjoints

```
h_hat1=ndft(f_hat,x,N,ft_opt1,'transp');
h_hat2=nfft(f_hat,x,N,ft_opt2,'transp');
```

Nonequispaced FFT - C

Simple example

```
nfft_plan p;
int N=14;
int M=19;

nfft_init_1d(&p,N,M);

nfft_vrand_shifted_unit_double(p.x,p.M_total);
if(p.nfft_flags & PRE_ONE_PSI)
  nfft_precompute_one_psi(&p);

nfft_vrand_unit_complex(p.f_hat,p.N_total);

nfft_trafo(&p);
nfft_finalize(&p);
```

Nonequispaced FFT - C

Data structure `nfft_plan`

```
int d;
int *N;
int N_total;
int M_total;

double complex *f_hat;
double complex *f;
double *x;

double *sigma;
int m;

unsigned nfft_flags;
unsigned fftw_flags;
...
```

Nonequispaced FFT - C, precomputation flags

For $j = 0, \ldots, M-1$ compute

$$f_j \approx \sum_{|\mathbf{l} - \mathbf{x}_j n| \le m} \left(\frac{1}{n^d} \sum_{\mathbf{k} \in I_N^d} \left(\frac{\hat{f}_{\mathbf{k}}}{\hat{\varphi}(\mathbf{k})} \right) \mathrm{e}^{-2\pi \mathrm{i} \mathbf{k} \mathbf{l}/n} \right) \tilde{\psi} \left(\mathbf{x}_j - \frac{\mathbf{l}}{n} \right).$$

Evaluate $\tilde{\psi}$ and $\hat{\varphi}$ on the fly or precompute and store the values?

$$\hat{\varphi}(\mathbf{k}) = \prod_{t=1}^{d} \hat{\varphi}(k_t) \qquad \mathbf{k} = (k_1, \ldots, k_d)^{\mathsf{T}}.$$

Method	memory	evaluations
-	-	$N_0 \cdot \ldots \cdot N_{d-1}$
`PRE_PHI_HUT`	$N_0 + \ldots + N_{d-1}$	-

Nonequispaced FFT - C, precomputation flags

$$\tilde{\psi}(\mathbf{x}) := \prod_{t=1}^{d} \tilde{\psi}(x_t), \qquad \mathbf{x} = (x_1, x_2, \ldots, x_d)^\mathsf{T}$$

Method	memory	evaluations
-	-	$m^d M$
PRE_PSI	dmM	-
PRE_FULL_PSI	$m^d M$	-

lookup table and linear interpolation PRE_LIN_PSI

Nonequispaced FFT - C, precomputation flags

fast Gaussian FG_PSI, PRE_FG_PSI [Greengard, Lee]
for $d = 1$ and a fixed x_j, $l' \in \{\lfloor x_j n \rfloor - m, \ldots, \lceil x_j n \rceil + m\}$

$$\sqrt{\pi b} \cdot \varphi\left(x_j - \frac{l'}{n}\right) = \mathrm{e}^{-\frac{(nx_j - l')^2}{b}} = \mathrm{e}^{-\frac{(nx_j - u)^2}{b}} \left(\mathrm{e}^{\frac{2(nx_j - u)}{b}}\right)^l \mathrm{e}^{-\frac{l^2}{b}} .$$

where $u = \min I_{n,m}(x_j)$ and $l = 0, \ldots, 2m$.

```
nfft_init_guru(&p, d, N, M, n, m,
               PRE_PHI_HUT| PRE_PSI|
               MALLOC_F_HAT| MALLOC_X| MALLOC_F|
               FFTW_INIT| FFT_OUT_OF_PLACE,
               FFTW_ESTIMATE| FFTW_DESTROY_INPUT);
```

Nonequispaced FFT - C

- doc (API docs)
- include (C headers)
- util (utility functions)
- kernel
  - fpt (fast polynomial transform)
  - nfct (nonequispaced fast cosine transform)
  - nfft (nonequispaced fast Fourier transform)
  - nfsft (nonequispaced fast spherical Fourier transform)
  - nfst (nonequispaced fast sine transform)
  - nnfft (nonequispaced in space and frequency FFT)
  - nsfft (nonequispaced sparse fast Fourier transform)
  - solver (inverse transforms)
- examples (for each in kernel)
- applications
  - fastgauss (fast Gauss transform)
  - fastsum (summation schemes)
  - fastsumS2 (summation on the sphere)
  - polarFFT (fast polar Fourier transform)
  - radon (radon transform)

Summary

Fast computation of

$$f_j = \sum_{k=-N/2}^{N/2-1} \hat{f}_k \, \mathrm{e}^{-2\pi \mathrm{i} k x_j}, \qquad j = 0, \ldots, M-1,$$

and

$$\hat{h}_k = \sum_{j=0}^{M-1} f_j \, \mathrm{e}^{+2\pi \mathrm{i} k x_j}, \qquad k = -N/2, \ldots, N/2-1,$$

for $x_j \in [-1/2, 1/2)$.

In short: $\mathbf{f} = \mathbf{A}\hat{\mathbf{f}}$, $\hat{\mathbf{h}} = \mathbf{A}^{\mathsf{H}}\mathbf{f}$ with $\mathbf{A} \in \mathbb{C}^{M \times N}$, $a_{j,k} = \mathrm{e}^{-2\pi \mathrm{i} k x_j}$.

Computational costs: $\mathcal{O}(N \log N + |\log \varepsilon| M)$ instead of $\mathcal{O}(NM)$.

Program

Part I – Fourier Analysis and the FFT
Stefan, Monday, 14:15 – 16:00, Room U322

Part II – Orthogonal Polynomials
Jens, Tuesday, 12:15 – 14:00, Room U141 (Lecture Hall F)

Practice Session: 14:30 – 16:00, Room Y339b (Basics and Matlab Hands-On)

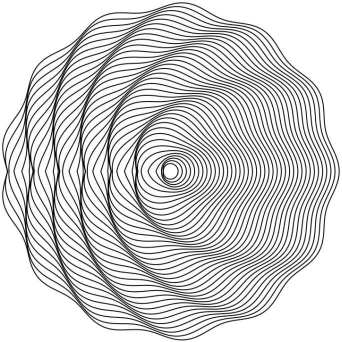
Part III – Fast Polynomial Transforms and Applications
Jens, Wednesday, 12:15 – 14:00, Room U345

Practice Session: 14:30 – 16:00, Room Y338c (C Library Hands-On)

Part IV – Fourier Transforms on the Rotation Group
Antje, Thursday, 14:15 – 16:00, Room U322

Part V – High Dimensions and Reconstruction
Stefan, Friday, 10:15 – 12:00, Room U322

Part II – Orthogonal Polynomials and Algorithms



Introduction

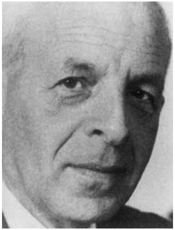
"Orthogonal polynomials are of great importance in mathematical physics, approximation theory, the theory of numerical quadrature, etc., and are the subject of an enormous literature."

[Nico M. Temme]

A.-M. Legendre
1752 – 1833

C. Hermite
1822 – 1901

E. T. Whittaker
1873 – 1956



G. Szegő
1895 – 1985

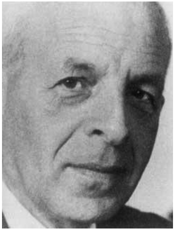
"Orthogonal polynomials are of great importance in mathematical physics, approximation theory, the theory of numerical quadrature, etc., and are the subject of an enormous literature."

[Nico M. Temme]

A.-M. Legendre
1752 – 1833

C. Hermite
1822 – 1901

E. T. Whittaker
1873 – 1956



G. Szegő
1895 – 1985

Outline

1. Orthogonal Polynomials
2. Classical Orthogonal Polynomials
3. Discrete Polynomial Transforms

Orthogonal Polynomials

A brief definition:

Orthogonal polynomials are polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ *defined over a range* $[a,b]$ *that obey an orthogonality relation,*

$$\int_a^b p_n(x)\,p_m(x)\,w(x)\,\mathrm{d}x = \delta_{n,m}\,h_n, \qquad h_n > 0.$$

Weisstein, Eric W. „Orthogonal Polynomials.“, MathWorld

Orthogonal Polynomials

A more general definition:

Let $\lambda(x)$ be a nondecreasing function on the real line $\mathbb{R}$ with finite limits $x \to \pm\infty$ and an induced positive measure $\mathrm{d}\lambda$ having finite moments

$$\mu_n = \mu_n(\mathrm{d}\lambda) := \int_{\mathbb{R}} x^n \mathrm{d}\lambda(x), \qquad n = 0, 1, 2, \ldots,$$

with $\mu_0 > 0$. Then for any two polynomials f, g, one may define an inner product as

$$\langle f, g \rangle = \int_{\mathbb{R}} f(x) g(x) \, \mathrm{d}\lambda(x).$$

Orthogonal polynomials are polynomials $\{p_n\}_{n \in \mathbb{N}_0}$ that are orthogonal with respect to such an inner product.

Gautschi, Walter „Orthogonal Polynomials – Computation and Approximation“

Orthogonal Polynomials – Basic Properties

Let a sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ be given, i.e.

$$\langle p_n, p_m \rangle = \delta_{n,m} h_n$$

What fundamental properties can we deduce?

Orthogonal Polynomials – Basic Properties

Let a sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ be given, i.e.

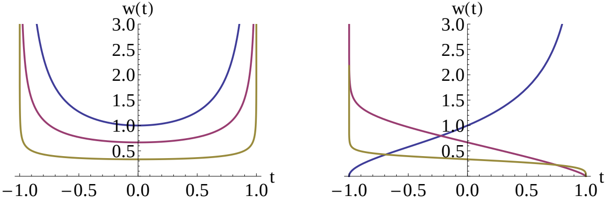
$$\langle p_n, p_m\rangle = \delta_{n,m} h_n$$

What fundamental properties can we deduce?

- **Symmetry** A measure $\mathrm{d}\lambda(x) = w(x)\,\mathrm{d}x$ is *symmetric*, iff

$$\operatorname{supp}(w) = [-a, a], a > 0, \text{ and } w(-x) = w(x).$$

Symmetric measures on $[-1,1]$ Non-symmetric measures on $[-1,1]$



If the measure $\mathrm{d}\lambda$ is symmetric, then $p_n(-x) = (-1)^n p_n(x)$.

Orthogonal Polynomials – Basic Properties

- **Basis** The set $\{p_j : 0 \leq j \leq n\}$ constitutes a basis for the space $\mathbb{P}_n$ of polynomials of degree at most n.

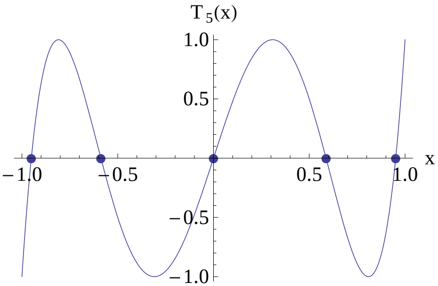
- **Zeros** All zeros of p_n are real, simple and located in the interior of the support interval of $\mathrm{d}\lambda$.

Orthogonal Polynomials – Basic Properties

- **Basis** The set $\{p_j : 0 \leq j \leq n\}$ constitutes a basis for the space $\mathbb{P}_n$ of polynomials of degree at most n.

- **Zeros** All zeros of p_n are real, simple and located in the interior of the support interval of $\mathrm{d}\lambda$.

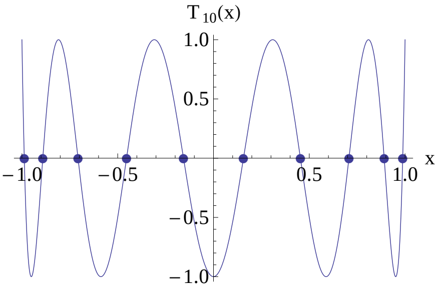
Chebyshev zeros for x in $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



- **Basis** The set $\{p_j : 0 \leq j \leq n\}$ constitutes a basis for the space $\mathbb{P}_n$ of polynomials of degree at most n.

- **Zeros** All zeros of p_n are real, simple and located in the interior of the support interval of $\mathrm{d}\lambda$.

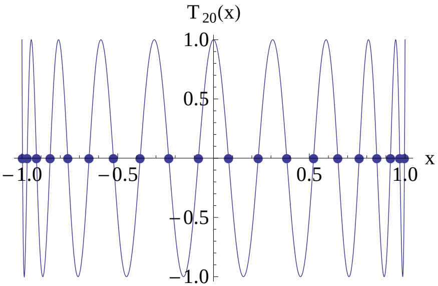
Chebyshev zeros for x in $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



- **Basis** The set $\{p_j : 0 \leq j \leq n\}$ constitutes a basis for the space $\mathbb{P}_n$ of polynomials of degree at most n.

- **Zeros** All zeros of p_n are real, simple and located in the interior of the support interval of $\mathrm{d}\lambda$.

Chebyshev zeros for x in $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$

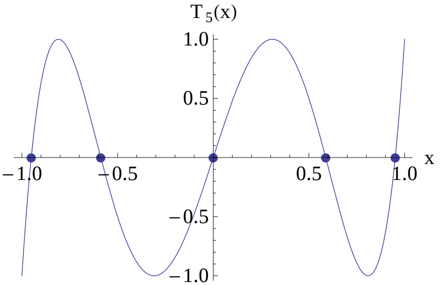


Orthogonal Polynomials – Basic Properties

- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

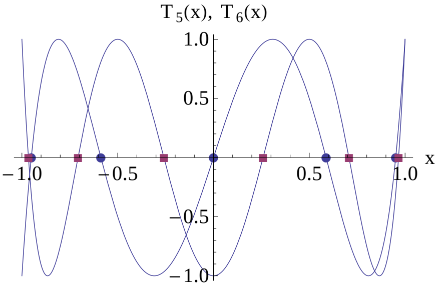
- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

Chebyshev zeros for x in $[-1,1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

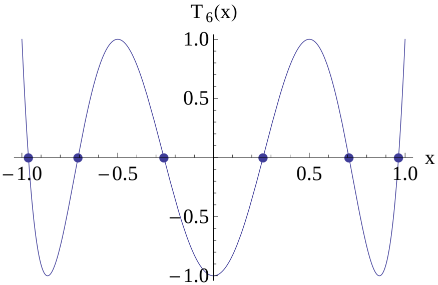
Chebyshev zeros for x in $[-1,1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



Orthogonal Polynomials – Basic Properties

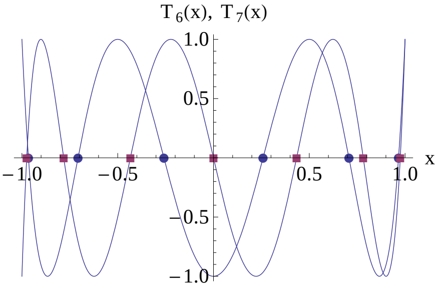
- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

Chebyshev zeros for x in $[-1,1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



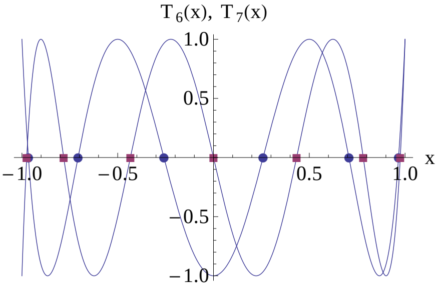
- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

Chebyshev zeros for x in $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



- **Interlacing** The zeros of p_{n+1} alternate with those of p_n.

Chebyshev zeros for x in $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$



- **Three-term recurrence** Orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfy a three-term recurrence with initial conditions:

$$p_{n+1}(x) = (a_n x - b_n)p_n(x) - c_n p_{n-1}(x), \qquad n = 0, 1, \dots ,$$
$$p_{-1}(x) = 0, \qquad p_0(x) = k_0.$$

Example: *Chebyshev polynomials of first kind* T_n are orthogonal with respect to the inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(x)\, g(x)\, \frac{1}{\sqrt{1-x^2}}\, \mathrm{d}x.$$

Orthogonal Polynomials – Basic Properties

Example: *Chebyshev polynomials of first kind* T_n are orthogonal with respect to the inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(x)\, g(x)\, \frac{1}{\sqrt{1-x^2}}\, \mathrm{d}x.$$

- **Definition:** $T_n(x) = \cos(n \arccos x)$ for $x \in [-1, 1]$.
- **Symmetry:** $T_n(-x) = (-1)^n T_n(x)$. ✓
- **Basis:** The set $\left\{T_j : 0 \leq j \leq n\right\}$ is a basis for $\mathbb{P}_n$. ✓
- **Zeros:** $T_n\left(\cos\left(\frac{(2j+1)\pi}{2n+2}\right)\right) = 0$ for $j = 0, 1, \ldots, n-1$. ✓
- **Three-term recurrence:** ✓

$$T_{n+1}(x) = (2 - \delta_{n,0})\, x\, T_n(x) - T_{n-1}(x), \qquad n = 0, 1, \ldots,$$
$$T_{-1}(x) = 0, \qquad T_0(x) = 1.$$

Orthogonal Polynomials – Basic Properties

One can rewrite the three-term recurrence

$$p_{n+1}(x) = (a_n x - b_n)p_n(x) - c_n p_{n-1}(x), \qquad n = 0, 1, \ldots ,$$

in the alternative form

$$x\, p_n(x) = \bar{a}_n p_{n+1}(x) + \bar{b}_n p_n(x) + \bar{c}_n p_{n-1}(x), \qquad n = 0, 1, \ldots .$$

Definition

The *Jacobi matrix* is the infinite tridiagonal matrix

$$\mathbf{J}_\infty = \begin{pmatrix} \bar{b}_0 & \bar{a}_0 & & 0 \\ \bar{c}_1 & \bar{b}_2 & \bar{a}_1 & \\ & \bar{c}_2 & \bar{b}_2 & \ddots \\ 0 & & \ddots & \ddots \end{pmatrix}.$$

Its $n \times n$ principal minor matrix is denoted $\mathbf{J}_n$.

One can also put the alternative recurrence

$$x\,p_n(x) = \bar{a}_n p_{n+1}(x) + \bar{b}_n p_n(x) + \bar{c}_n p_{n-1}(x), \qquad n = 0, 1, \ldots\,,$$

in matrix-vector form

$$x\,\mathbf{p}(x) = \mathbf{J}_n \mathbf{p}(x) + \bar{a}_n p_n(x) \mathbf{e}_n, \qquad n = 0, 1, \ldots\,,$$

$$\mathbf{p}(x) = \left(p_0(x), p_1(x), \ldots, p_{n-1}(x)\right)^{\mathrm{T}}, \qquad \mathbf{e}_n = (0, 0, \ldots, 0, 1)^{\mathrm{T}}.$$

Theorem

The zeros $\tau_{n,j}$, $j = 0, 1, \ldots, n-1$, of p_n are the eigenvalues of $\mathbf{J}_n$. The corresponding eigenvectors are $\mathbf{p}(\tau_{n,j})$, $j = 0, \ldots, n-1$.

Proof.

Put $x = \tau_{n,j}$, for $j = 0, 1, \ldots, n-1$ and note that $\mathbf{p}(\tau_{j,n}) \neq \mathbf{0}$. ■

Outline

1. Orthogonal Polynomials
2. Classical Orthogonal Polynomials
3. Discrete Polynomial Transforms

Classical Orthogonal Polynomials – Introduction

How do orthogonal polynomials actually arise?

Time-dependent PDE

↓ Remove time variable

Time-independent PDE

↓ Coordinate transformation

New time-independent PDE

↓ Separation of variables

Multiple ODEs

↓ Boundary or other conditions

Polynomial solutions

Time-dependent PDE $\xrightarrow{\text{Remove time variable}}$ Time-independent PDE

Typical PDEs that arise in mathematical physics:

Laplace's equation:	$\Delta u = 0$
Heat equation:	$\Delta u = u_t$
Wave equation:	$\Delta u = u_{tt}$

$$u = u(t, x_1, x_2, \ldots, x_n)$$

$$\Delta = \sum_{j=1}^{n} \frac{\mathrm{d}^2}{\mathrm{d}x_j^2}$$

Time-dependent PDE $\xrightarrow{\text{Remove time variable}}$ Time-independent PDE

Typical PDEs that arise in mathematical physics:

Laplace's equation:	$\Delta u = 0$
Heat equation:	$\Delta u = u_t$
Wave equation:	$\Delta u = u_{tt}$

$$u = u(t, x_1, x_2, \ldots, x_n)$$

$$\Delta = \sum_{j=1}^{n} \frac{\mathrm{d}^2}{\mathrm{d}x_j^2}$$

How to remove the time variable?

- Pull out time dependency, e.g. $u(t, x_1, x_2, \ldots, x_n) = \mathrm{e}^{\mathrm{i}kt} v(x_1, x_2, \ldots, x_n)$
- Fourier or Laplace transformation
- New equation: Helmholtz $\Delta v + k^2 v = 0$, Schrödinger equation

Classical Orthogonal Polynomials – Introduction

Time-dependent PDE $\xrightarrow{\text{Remove time variable}}$ Time-independent PDE

Typical PDEs that arise in mathematical physics:

Laplace's equation:	$\Delta u = 0$
Heat equation:	$\Delta u = u_t$
Wave equation:	$\Delta u = u_{tt}$

$$u = u(t, x_1, x_2, \dots, x_n)$$

$$\Delta = \sum_{j=1}^{n} \frac{\mathrm{d}^2}{\mathrm{d}x_j^2}$$

How to remove the time variable?

- Pull out time dependency, e.g. $u(t, x_1, x_2, \dots, x_n) = \mathrm{e}^{\mathrm{i}kt} v(x_1, x_2, \dots, x_n)$
- Fourier or Laplace transformation
- New equation: Helmholtz $\Delta v + k^2 v = 0$, Schrödinger equation

Example: 2D-Laplace $\frac{\mathrm{d}^2}{\mathrm{d}x^2} v + \frac{\mathrm{d}^2}{\mathrm{d}y^2} v = 0$, $v = v(x, y)$

already time-independent

Classical Orthogonal Polynomials – Introduction

Time-independent PDE $\xrightarrow{\text{New coordinates}}$ New time-independent PDE

2D-Cartesian Laplace $\dfrac{\mathrm{d}^2}{\mathrm{d}x^2}v + \dfrac{\mathrm{d}^2}{\mathrm{d}y^2}v = 0,\ v = v(x,y)$

$\Big\downarrow\ x = r\cos\theta,\ y = r\sin\theta$

2D-Polar Laplace $\dfrac{\mathrm{d}^2}{\mathrm{d}r^2}v + \dfrac{1}{r}\dfrac{\mathrm{d}}{\mathrm{d}r}v + \dfrac{1}{r^2}\dfrac{\mathrm{d}^2}{\mathrm{d}\theta^2}v = 0,\ v = v(r,\theta)$

Classical Orthogonal Polynomials – Introduction

$$\boxed{\text{Time-independent PDE}} \xrightarrow{\text{New coordinates}} \boxed{\text{New time-independent PDE}}$$

$$\boxed{\text{2D-Cartesian Laplace} \quad \frac{\mathrm{d}^2}{\mathrm{d}x^2}v + \frac{\mathrm{d}^2}{\mathrm{d}y^2}v = 0, \ v = v(x,y)}$$

$$\Big\downarrow \ x = r\cos\theta, \ y = r\sin\theta$$

$$\boxed{\text{2D-Polar Laplace} \quad \frac{\mathrm{d}^2}{\mathrm{d}r^2}v + \frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r}v + \frac{1}{r^2}\frac{\mathrm{d}^2}{\mathrm{d}\theta^2}v = 0, \ v = v(r,\theta)}$$

Under certain conditions, we can seek solutions of the form

$$v(r,\theta) = f(r)g(\theta).$$

New time-independent PDE $\xrightarrow{\text{Separation of variables}}$ Multiple ODEs

2D-Polar Laplace $\dfrac{\mathrm{d}^2}{\mathrm{d}r^2}v + \dfrac{1}{r}\dfrac{\mathrm{d}}{\mathrm{d}r}v + \dfrac{1}{r^2}\dfrac{\mathrm{d}^2}{\mathrm{d}\theta^2}v = 0, \; v = v(r,\theta)$

$\downarrow$ $v(r,\theta) = f(r)g(\theta)$

$$r^2\frac{f''}{f} + r\frac{f'}{f} + \frac{g''}{g} = 0, \qquad f = f(r), \; g = g(\theta)$$

$\downarrow$ Terms in f and g are constant.

$$r^2\frac{f''}{f} + r\frac{f'}{f} = -\frac{g''}{g} = k^2, \qquad k \in \mathbb{N}_0$$

Function $g(\theta)$ must be 2π-periodic!

Solve individual ODEs:

- $\dfrac{g''}{g} = -k^2$ has solutions $g(\theta) = \mathrm{e}^{\pm \mathrm{i}k\theta}$
- $r^2 \dfrac{f''}{f} + r\dfrac{f'}{f} = -k^2$ has solutions $f(r) = r^{\pm k}$

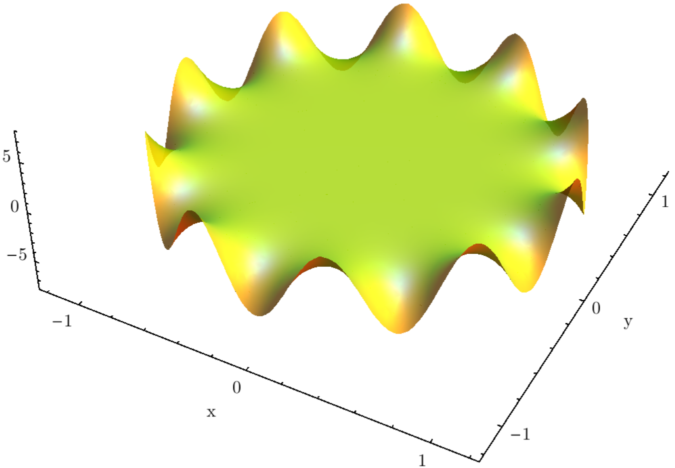
Laplace's equation $\Delta v = 0$ has fundamental solutions of the form

$$v(r, \theta) = r^{\pm k} \mathrm{e}^{\pm \mathrm{i}k\theta}.$$

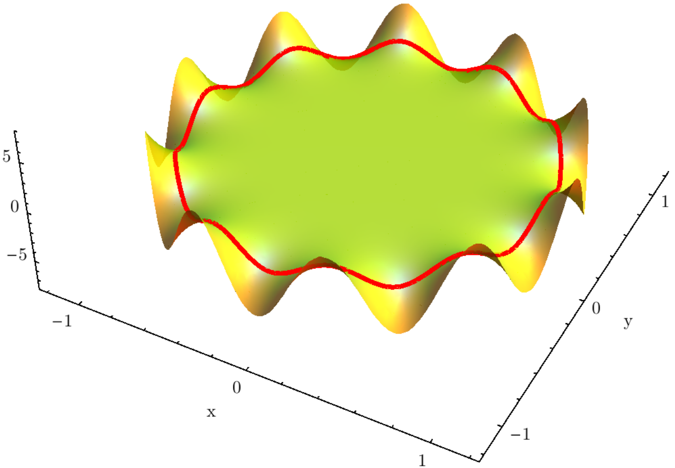
Often, the solution with r^{-k} can be removed because of the pole at the origin.
Connection to Fourier series:

On the unit circle, i.e. for $r = 1$, one obtains the complex exponentials $\mathrm{e}^{\pm \mathrm{i}k\theta}$!

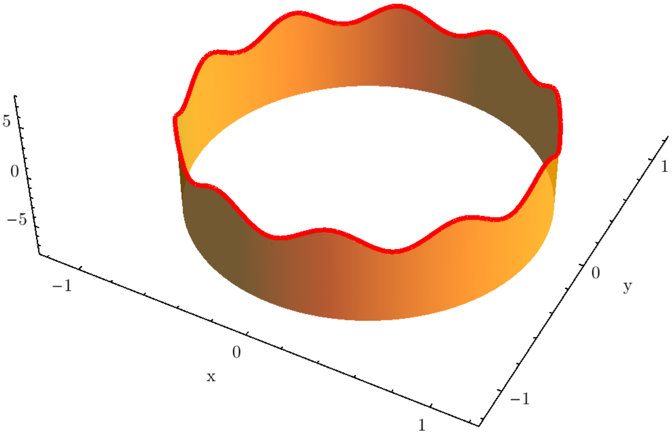
Real part of solution $v(r, \theta) = r^k \mathrm{e}^{\mathrm{i}k\theta}$ for $k = 10$.



Real part of solution $v(r, \theta) = r^k e^{ik\theta}$ for $k = 10$.



Real part of solution $v(r, \theta) = r^k e^{ik\theta}$ for $k = 10$.



...arise as solution to ODEs of the form

$$\boxed{\sigma y'' + \tau y' + \lambda_n y = 0, \qquad \sigma \in \mathbb{P}_2, \tau \in \mathbb{P}_1, \lambda_n \in \mathbb{R}, n \in \mathbb{N}_0, y = y(x),}$$

that follow from solving PDEs using separation of variables.

...arise as solution to ODEs of the form

$$\boxed{\sigma y'' + \tau y' + \lambda_n y = 0, \qquad \sigma \in \mathbb{P}_2, \tau \in \mathbb{P}_1, \lambda_n \in \mathbb{R}, n \in \mathbb{N}_0, y = y(x),}$$

that follow from solving PDEs using separation of variables.

Example:

- Legendre differential equation ($\to$ Legendre polynomials P_n)

$$(1 - x^2)y'' - 2xy' + n(n+1)y = 0$$

This equation arises when solving the Laplace equation $\Delta v = 0$ in spherical coordinates (r, θ, ϕ) with boundary conditions that have axial symmetry (no dependence on ϕ). Then the solution v can be expanded as

$$v(r, \theta) = \sum_{k=0}^{\infty} \left(a_k r^k + b_k r^{-k}\right) P_k(x).$$

Further examples:

- Chebyshev differential eq. ($\rightarrow$ Chebyshev polynomials T_n)

$$(1-x^2)y'' - xy' + n^2 y = 0$$

- Hermite differential equation ($\rightarrow$ Hermite polynomials H_n)

$$y'' - 2xy' + 2ny = 0$$

This equation is equivalent to the Schrödinger equation for a harmonic oscillator in quantum mechanics.

...arise as solution to ODEs of the form

$$\boxed{\sigma y'' + \tau y' + \lambda_n y = 0, \quad \sigma \in \mathbb{P}_2, \tau \in \mathbb{P}_1, \lambda_n \in \mathbb{R}, n \in \mathbb{N}_0, y = y(x),}$$

that follow from solving PDEs using separation of variables.

What can be derived from the differential equation:

- Polynomial solution $p_n \in \mathbb{P}_n \iff \lambda_n = -n\tau' - \frac{n(n-1)}{2}\sigma''$.
- The sequence $\{p_n'\}_{n\in\mathbb{N}_0}$ satisfies an ODE of similar type.
- There is a *Rodrigues formula*, that is,

$$p_n(x) = \frac{B_n}{w(x)} \frac{\mathrm{d}^n}{\mathrm{d}x^n}\big(\sigma^n(x) w(x)\big), \qquad B_n \in \mathbb{R}.$$

- The function $w(x) = \frac{1}{\sigma(x)} \mathrm{e}^{\int \frac{\tau(x)}{\sigma(x)}\,\mathrm{d}x}$ gives the self-adjoint form

$$(\sigma w y')' + \lambda_n w y = 0.$$

Classical Orthogonal Polynomials

All classical polynomials characterized by choice of σ and τ!

- Distinguish three cases for σ (up to constant factors):

$$\sigma(x) = \begin{cases} (b-x)(x-a), \\ (x-a), \\ 1 \end{cases}$$

Classical Orthogonal Polynomials

All classical polynomials characterized by choice of σ and τ!

- Distinguish three cases for σ (up to constant factors):

$$\sigma(x) = \begin{cases} (b-x)(x-a), \\ (x-a), \\ 1 \end{cases}$$

- Write $\tau \in \mathbb{P}_1$ with two degrees of freedom, say, α, β.

Classical Orthogonal Polynomials

All classical polynomials characterized by choice of σ and τ!

- Distinguish three cases for σ (up to constant factors):

$$\sigma(x) = \begin{cases} (b-x)(x-a), \\ (x-a), \\ 1 \end{cases}$$

- Write $\tau \in \mathbb{P}_1$ with two degrees of freedom, say, α, β.
- Determine $w(x)$ for each case:

$$w(x) = \begin{cases} (b-x)^\alpha (x-a)^\beta, & \text{if } \sigma(x) = (b-x)(x-a), \\ (x-a)^\alpha \mathrm{e}^{\beta x}, & \text{if } \sigma(x) = (x-a), \\ \mathrm{e}^{\alpha x^2 + \beta x}, & \text{if } \sigma(x) = 1. \end{cases}$$

All classical polynomials characterized by choice of σ and τ!

$\vdots$

- Determine $w(x)$ for each case:

$$w(x) = \begin{cases} (b-x)^\alpha (x-a)^\beta, & \text{if } \sigma(x) = (b-x)(x-a), \\ (x-a)^\alpha \mathrm{e}^{\beta x}, & \text{if } \sigma(x) = (x-a), \\ \mathrm{e}^{\alpha x^2 + \beta x}, & \text{if } \sigma(x) = 1. \end{cases}$$

After linear change of variable:

$$w(x) = \begin{cases} (1-x)^\alpha (1+x)^\beta, & \text{if } \sigma(x) = 1 - x^2, \\ x^\alpha \mathrm{e}^{-x}, & \text{if } \sigma(x) = x, \\ \mathrm{e}^{-x^2}, & \text{if } \sigma(x) = 1. \end{cases}$$

We have omitted the case when σ has a double root which gives rise to the *Bessel polynomials*.

Classical Orthogonal Polynomials

What is the measure $\mathrm{d}\lambda$ with respect to which the polynomials corresponding to different values $\lambda_n = -n\tau' - \frac{n(n-1)}{2}\sigma''$ are orthogonal?

Theorem (Orthogonality)

If $w(x)$, as defined before, satisfies

$$\sigma(x)w(x)x^n\Big|_{x=a,b} = 0, \qquad n = 0, 1, \dots\,,$$

at the endpoints of an interval $[a, b]$. Then the polynomials p_n corresponding to different values λ_n are orthogonal with respect to the measure $\mathrm{d}\lambda(x) = w(x)\,\chi_{[a,b]}(x)\,\mathrm{d}x$, i.e.

$$\int_a^b p_n(x)p_m(x)\,w(x)\,\mathrm{d}x = \delta_{n,m}h_n, \quad h_n > 0.$$

Classical Orthogonal Polynomials

Proof.

Take the differential equations for p_n and p_m,

$$\left(\sigma w\, p_n'\right)' + \lambda_n w\, p_n = 0, \qquad \left(\sigma w\, p_m'\right)' + \lambda_m w\, p_m = 0.$$

Multiply the first by p_m, the second by p_n, and subtract,

$$(\lambda_n - \lambda_m)\, w\, p_n\, p_m = p_n\,(\sigma w\, p_m)' - p_m\,(\sigma w\, p_n)' = \frac{\mathrm{d}}{\mathrm{d}x}\big(\sigma w\, W(p_n, p_m)\big)$$

with the *Wronskian* $W(p_n, p_m) := \begin{vmatrix} p_n & p_m \\ p_n' & p_m' \end{vmatrix}$. Integrate both sides,

$$\int_a^b p_n(x) p_m(x) w(x)\, \mathrm{d}x = \frac{1}{\lambda_n - \lambda_m}\big[\sigma(x) w(x) W(p_n(x), p_m(x))\big]_a^b.$$

Since $W(p_n(x), p_m(x)) \in \mathbb{P}$, the right hand side vanishes. ∎

Classical Orthogonal Polynomials – Examples

$$\sigma y'' + \tau y' + \lambda_n y = 0, \qquad \int_a^b p_n(x)\, p_m(x)\, w(x)\, \mathrm{d}x = \delta_{n,m} h_n.$$

$\sigma(x)$

$$\sigma(x) = 1, \quad w(x) = \mathrm{e}^{-x^2} \longrightarrow$$ Hermite polynomials H_n, $[a,b] = (-\infty, \infty)$

$$\sigma(x) = x, \quad w(x) = x^\alpha \mathrm{e}^{-x} \longrightarrow$$ Laguerre polynomials $L_n^{(\alpha)}$, $[a,b] = [0, \infty)$

$$\sigma(x) = 1 - x^2, \quad w(x) = (1-x)^\alpha (1+x)^\beta \longrightarrow$$ Jacobi polynomials $P_n^{(\alpha,\beta)}$, $[a,b] = [-1, 1]$

Hermite polynomials H_n

- are orthogonal on $(-\infty, \infty)$
- $\sigma(x) = 1$, $\tau(x) = -2x$, $w(x) = \mathrm{e}^{-x^2}$, $\lambda_n = 2n$
- Differential equation

$$y'' + -2xy' + 2ny = 0$$

- Rodrigues formula

$$H_n(x) = (-1)^n \mathrm{e}^{x^2} \frac{\mathrm{d}^n}{\mathrm{d}x^n}\left(\mathrm{e}^{-x^2}\right),$$

- Three-term recurrence

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x),$$
$$H_{-1}(x) = 0, \quad H_0(x) = 1.$$

Laguerre polynomials $L_n^{(\alpha)}$

- are orthogonal on $[0, \infty)$
- $\sigma(x) = x$, $\tau(x) = -x + \alpha + 1$, $w(x) = x^{\alpha} \mathrm{e}^{-x}$, $\lambda_n = n$, $-1 < \alpha$
- Differential equation

$$xy'' + (-x + \alpha + 1)y' + ny = 0$$

- Rodrigues formula

$$L_n^{(\alpha)}(x) = \frac{1}{n!} x^{-\alpha} \mathrm{e}^{x} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left(x^{\alpha+n} \mathrm{e}^{-x} \right),$$

- Three-term recurrence

$$(n+1)L_{n+1}^{(\alpha)}(x) = (-x + (2n + \alpha + 1))L_n^{(\alpha)}(x) - (n + \alpha)L_{n-1}^{(\alpha)}(x),$$
$$L_{-1}^{(\alpha)}(x) = 0, \quad L_0^{(\alpha)}(x) = 1.$$

Classical Orthogonal Polynomials – Examples

Jacobi polynomials $P_n^{(\alpha,\beta)}$

- are orthogonal on $[-1,1]$
- $\sigma(x)=1-x^2$, $\tau(x)=-(\alpha+\beta+2)x+\beta-\alpha$,

  $w(x)=(1-x)^{\alpha}(1+x)^{\beta}$, $\lambda_n=n(n+\alpha+\beta+1)$, $-1<\alpha,\beta$
- Differential equation

$$(1-x^2)y''-((\alpha+\beta+2)x+\alpha-\beta)y'+n(n+\alpha+\beta+1)y=0$$

- Rodrigues formula

$$P_n^{(\alpha,\beta)}(x)=\frac{(-1)^n}{2^n n!}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left((1-x)^{n+\alpha}(1+x)^{n+\beta}\right).$$

- Three-term recurrence too complicated to put here.

Important special cases of Jacobi polynomials:

- Legendre polynomials P_n:

$$P_n(x) = P_n^{(0,0)}(x),$$

- Chebyshev polynomials of first and second kind T_n and U_n:

$$T_n(x) = \frac{\Gamma(1/2)\Gamma(n+1)}{\Gamma(n+1/2)} P_n^{(-1/2,-1/2)}(x) = \cos(n\theta),$$

$$U_n(x) = \frac{\Gamma(1/2)\Gamma(n+2)}{2\Gamma(n+3/2)} P_n^{(1/2,1/2)}(x) = \frac{\sin((n+1)\theta)}{\sin\theta},$$

- Gegenbauer/Ultraspherical polynomials $C_n^{(\alpha)}$:

$$C_n^{(\alpha)}(x) = \frac{\Gamma(\alpha+\frac{1}{2})\Gamma(n+2\alpha)}{\Gamma(n+\alpha+\frac{1}{2})\Gamma(2\alpha)} P_n^{(\alpha-\frac{1}{2},\alpha-\frac{1}{2})}(x), \qquad \alpha \neq 0.$$

Outline

1 Orthogonal Polynomials

2 Classical Orthogonal Polynomials

3 Discrete Polynomial Transforms

Discrete Polynomial Transforms

The road to *discrete polynomial transforms*:

Discrete Fourier Tranform

$$\sum_{k=0}^{N-1} \hat{f}_k e^{-2\pi i \frac{kj}{N}}$$

$\downarrow$ Allow arbitrary nodes

Nonequispaced Discrete Fourier Transform

$$\sum_{k=-N/2}^{N/2-1} \hat{f}_k e^{-2\pi i k x_j}$$

$\downarrow$ Allow other basis

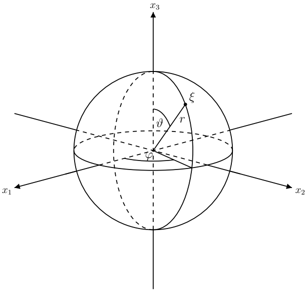
Discrete Polynomial Transforms

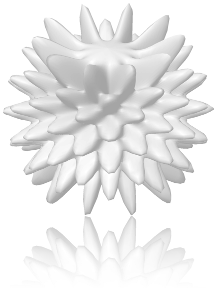
$$\sum_{k=0}^{N} \hat{f}_k p_k(x_j)$$

Discrete Fourier transform on the sphere $\mathbb{S}^2$ (NDSFT)

Compute the sums

$$f(\vartheta_j, \varphi_j) = \sum_{\ell=0}^{N} \sum_{m=-\ell}^{\ell} \hat{f}_\ell^m P_\ell^m(\cos\vartheta_j) \mathrm{e}^{\mathrm{i}m\varphi_j}, \qquad j = 1, 2, \ldots, M.$$





Discrete Fourier transform on $\mathrm{SO}(3)$ (NDSOFT)

Compute the sums

$$f(\alpha_j, \beta_j, \gamma_j) = \sum_{\ell=0}^{N} \sum_{m=-\ell}^{\ell} \sum_{n=-\ell}^{\ell} \hat{f}_\ell^{m,n} \mathrm{e}^{-\mathrm{i}m\alpha_j} d_\ell^{m,n}(\cos\beta_j) \mathrm{e}^{-\mathrm{i}n\gamma_j}$$

for $j = 1, 2, \ldots, M$.

Antje's talk...

Discrete Polynomial Transforms – Definition

Definition (Discrete Polynomial Transform)

Given a sequence of orthogonal polynomials/functions $\{p_n\}_{n\in\mathbb{N}}$, coefficients $\hat{f}_k$, $k = 0, \ldots, N$, and nodes x_j, $j = 1, \ldots, M$, compute

$$f_j = \sum_{k=0}^{N} \hat{f}_k p_k(x_j), \qquad j = 1, 2, \ldots, M.$$

Goal: fast and numerically stable algorithms

Discrete Polynomial Transforms – Matrix-Vector Notation

- The equations

$$f_j = \sum_{k=0}^{N} \hat{f}_k p_k(x_j), \qquad j = 1, 2, \ldots, M,$$

correspond to the matrix-vector product

$$\mathbf{f} = \mathbf{P}\,\hat{\mathbf{f}}, \qquad \mathbf{P} = (p_k(x_j))_{j=1,k=0}^{M,N} \in \mathbb{R}^{M\times(N+1)}.$$

- The *transposed* problem reads

$$\hat{h}_k = \sum_{j=1}^{M} f_j p_k(x_j), \qquad k = 0, 1, \ldots, N,$$

or, equivalently,

$$\tilde{\mathbf{f}} = \mathbf{P}^{\mathrm{T}}\,\mathbf{f}.$$

- Rationale: Can be used to recover the expansion coefficients $\hat{f}_k$ if a suitable quadrature formula to discretise the inner product $\hat{f}_k = \int_{-1}^{1} f(x) p_k(x)\, \mathrm{d}\lambda(x)$ is available.

Discrete Polynomial Transforms – Comparison to NFFT

What is the benchmark algorithm against which any other algorithm should be valuated?

Comparison with nonequispaced discrete Fourier transform

$$\mathbf{f} = \mathbf{A}\,\hat{\mathbf{f}},\ \mathbf{A} = \left(\mathrm{e}^{\mathrm{i}kx_j}\right)_{j=1,k=-N/2}^{M,N/2-1}$$

Method	Direct (online)	Direct (precomputed)	NFFT ϵ = accuracy
Time	$\mathcal{O}(NM)$	$\mathcal{O}(NM)$	$\mathcal{O}(N\log N + \log(1/\varepsilon)M)$
Memory	$\mathcal{O}(1)$	$\mathcal{O}(NM)$	$\mathcal{O}(1) - \mathcal{O}(N + \log(1/\varepsilon)M)$

Discrete Polynomial Transforms – Comparison to NFFT

What is the benchmark algorithm against which any other algorithm should be valuated?

Comparison with nonequispaced discrete Fourier transform

$$\mathbf{f} = \mathbf{A}\,\hat{\mathbf{f}},\ \mathbf{A} = \left(\mathrm{e}^{\mathrm{i}kx_j}\right)_{j=1,k=-N/2}^{M,N/2-1}$$

Method	Direct (online)	Direct (precomputed)	NFFT ϵ = accuracy
Time	$\mathcal{O}(NM)$	$\mathcal{O}(NM)$	$\mathcal{O}(N \log N + \log(1/\varepsilon)M)$
Memory	$\mathcal{O}(1)$	$\mathcal{O}(NM)$	$\mathcal{O}(1) - \mathcal{O}(N + \log(1/\varepsilon)M)$

- Direct method (online evaluation): slowest of all methods, constant amount of memory
- Direct method (precomputation): fastest for small transform sizes, large amount of memory needed
- NFFT: fastest for large enough transforms, moderate and adjustable memory requirements

Discrete Polynomial Transforms – Algorithms

What is the benchmark algorithm against which any other algorithm should be valuated?

Discrete polynomial transform

$$\mathbf{f} = \mathbf{P}\,\hat{\mathbf{f}},\ \mathbf{P} = (p_k(x_j))_{j=1,k=0}^{M,N}$$

What is the benchmark algorithm against which any other algorithm should be valuated?

Discrete polynomial transform

$$\mathbf{f} = \mathbf{P}\,\hat{\mathbf{f}},\ \mathbf{P} = (p_k(x_j))_{j=1,k=0}^{M,N}$$

- Direct method (online evaluation, precomputation)

  p_k **alone cannot be evaluated in constant time!**

Discrete Polynomial Transforms – Algorithms

What is the benchmark algorithm against which any other algorithm should be valuated?

Discrete polynomial transform

$$\mathbf{f} = \mathbf{P}\,\hat{\mathbf{f}},\ \mathbf{P} = (p_k(x_j))_{j=1,k=0}^{M,N}$$

- Direct method (online evaluation, precomputation)

  p_k **alone cannot be evaluated in constant time!**

- Fast method

  What is a fast polynomial transform?

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Discrete Polynomial Transforms – Clenshaw Algorithm

Each entry $p_k(x_j)$ in the matrix $\mathbf{P}$ cannot be evaluated in constant time, but each inner product between a single row of $\mathbf{P}$ and the vector $\hat{\mathbf{f}}$ can be computed $\mathcal{O}(N)$ arithmetic operations

$$f(x_j) = (p_0(x_j), p_1(x_j), \ldots, p_N(x_j)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right) = \sum_{k=0}^{N} \hat{f}_k p_k(x_j).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

For x fixed, evaluate a finite linear combination of polynomials,

$$f(x) = \sum_{j=0}^{N} \hat{f}_j p_n(x),$$

that satisfy a three-term recurrence

$$p_{n+1}(x) = (a_n x - b_n) p_n(x) - c_n p_{n-1}(x), \qquad n = 0, 1, \ldots\,,$$
$$p_{-1}(x) = 0, \qquad p_0(x) = k_0.$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k$, $k = 0, 1, \ldots, N$, $x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$
$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ \ x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

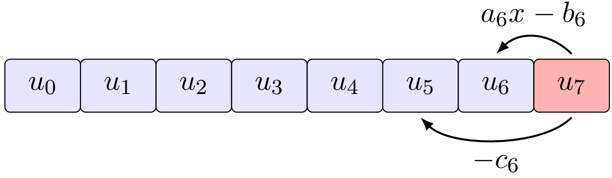
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ \ x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

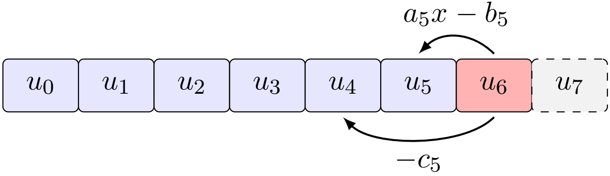
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ x \in \mathbb{R}$.

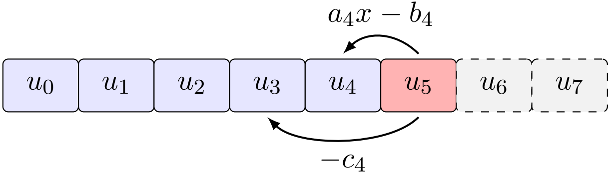
for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$
$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k$, $k = 0, 1, \ldots, N$, $x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

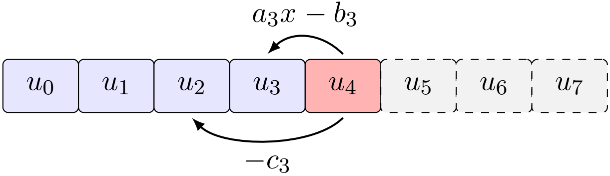
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ \ x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

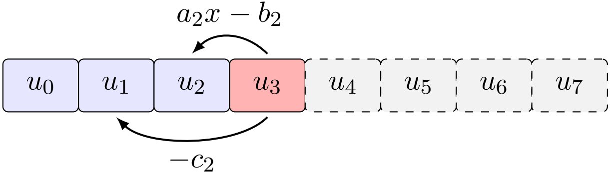
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ \ x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

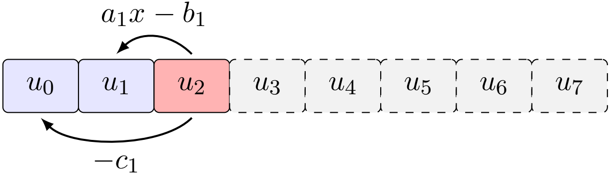
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$

$f(x) = k_0 u_0$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k,\ k = 0, 1, \ldots, N,\ \ x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

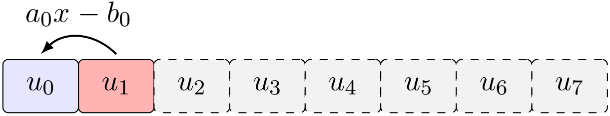
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

Clenshaw Algorithm (Clenshaw 1955, Smith 1965)

Input: $u_k = \hat{f}_k$, $k = 0, 1, \ldots, N$, $x \in \mathbb{R}$.

for $i = N-1, N-2, \ldots, 1$ **do**

$$u_i \mathrel{+}= (a_i x - b_i)\, u_{i+1}$$

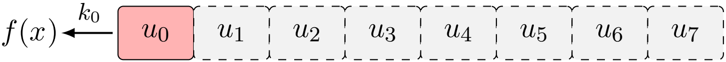
$$u_{i-1} \mathrel{-}= c_i u_{i+1}$$

end for

$$u_0 \mathrel{+}= (a_0 x - b_0)\, u_1$$

$$f(x) = k_0 u_0$$

Output: $f(x)$, **Time:** $\mathcal{O}(N)$, **Memory:** $\mathcal{O}(1) - \mathcal{O}(N)$.



Discrete Polynomial Transforms – Clenshaw Algorithm

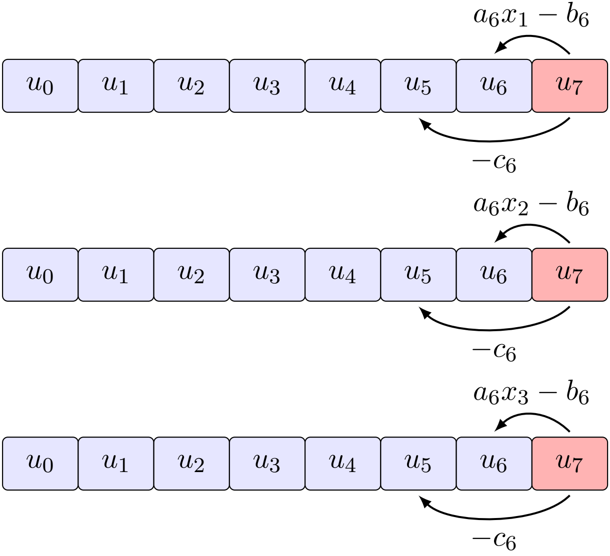
Repeat procedure for each node:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

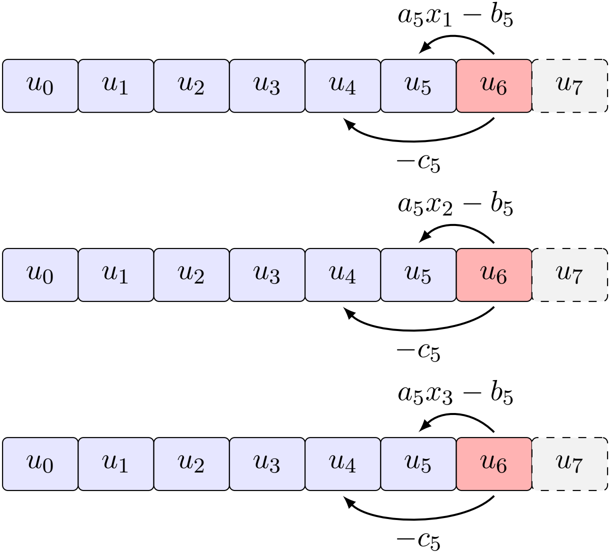
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Repeat procedure for each node:

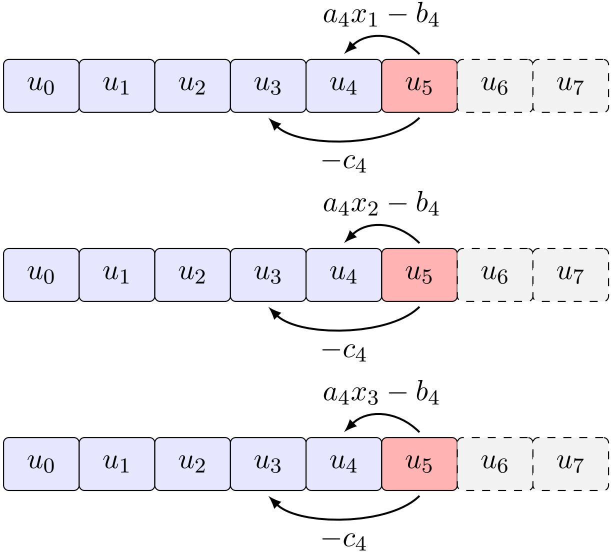


Repeat procedure for each node:



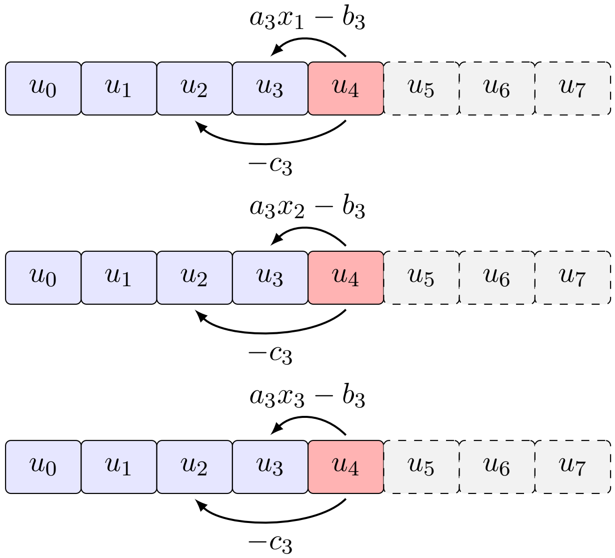
Discrete Polynomial Transforms – Clenshaw Algorithm

Repeat procedure for each node:

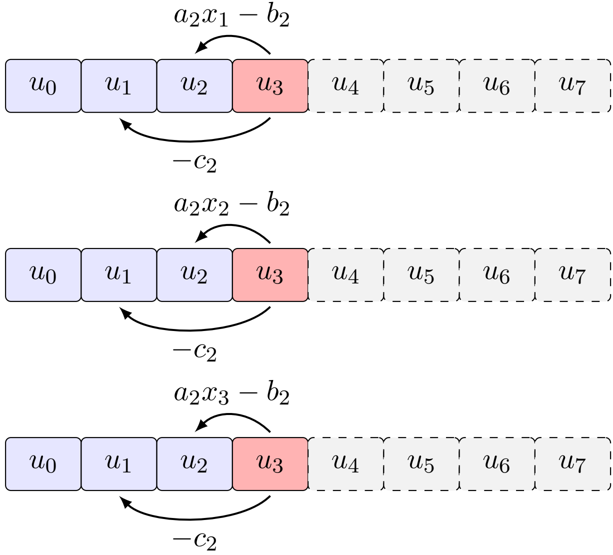


Discrete Polynomial Transforms – Clenshaw Algorithm

Repeat procedure for each node:

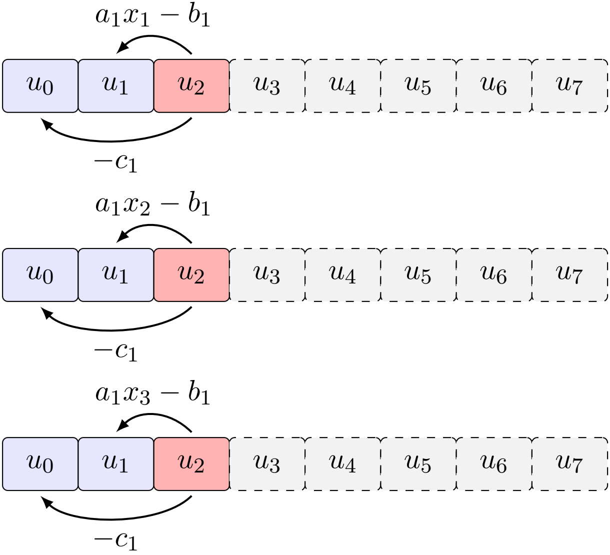


Repeat procedure for each node:



Discrete Polynomial Transforms – Clenshaw Algorithm

Repeat procedure for each node:



Discrete Polynomial Transforms – Clenshaw Algorithm

Repeat procedure for each node:

$$a_0 x_1 - b_0$$

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

$$a_0 x_2 - b_0$$

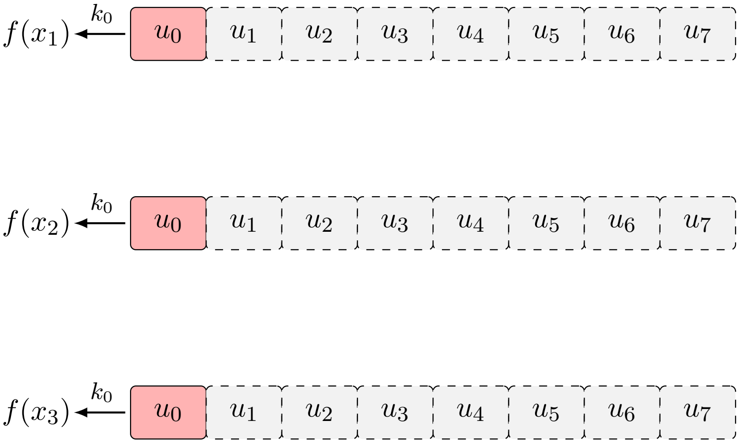
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

$$a_0 x_3 - b_0$$

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Discrete Polynomial Transforms – Clenshaw Algorithm

Repeat procedure for each node:



Discrete Polynomial Transforms – Clenshaw Algorithm

Summary:

- Time: $\mathcal{O}(NM)$ ✗
- Memory: $\mathcal{O}(1) - \mathcal{O}(N)$ ✓
- Numerically stable (Barrio, 2002) ✓

$$\left|\tilde{f}(x) - f(x)\right| \leq u \sum_{k=0}^{N} |\hat{f}_k||p_k(x)| + \mathcal{O}(u^2), \quad u = \text{unit roundoff}$$

- Requirements: Three-term recurrence

How to derive the *transposed* version of this algorithm?

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$f(x) = (p_0(x), p_1(x), \dots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \dots, \hat{f}_N\right)$$
$$= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}} \, \hat{\mathbf{f}},$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

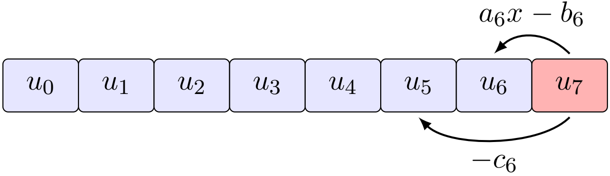
corresponds to

$$f(x) = \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{=\mathbf{p}(x)^{\mathrm{T}}} \hat{\mathbf{f}}.$$

The Clenshaw algorithm computes

$$f(x) = (p_0(x), p_1(x), \ldots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right)$$
$$= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

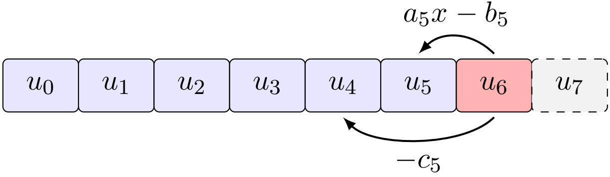
$$f(x) = \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$\begin{aligned} f(x) &= (p_0(x), p_1(x), \dots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \dots, \hat{f}_N\right) \\ &= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}}, \end{aligned}$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

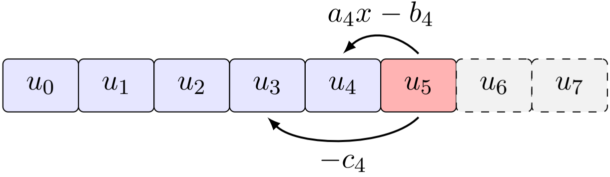
$$f(x) = \underbrace{\qquad\qquad\qquad\qquad \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$f(x) = (p_0(x), p_1(x), \dots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \dots, \hat{f}_N\right)$$
$$= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



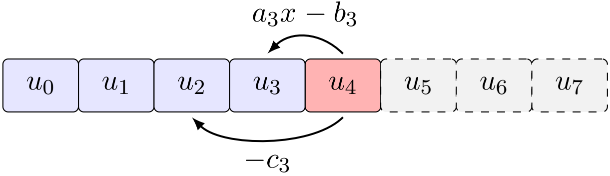
corresponds to

$$f(x) = \underbrace{\qquad\qquad\qquad\qquad \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

The Clenshaw algorithm computes

$$\begin{aligned} f(x) &= (p_0(x), p_1(x), \ldots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right) \\ &= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}}, \end{aligned}$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

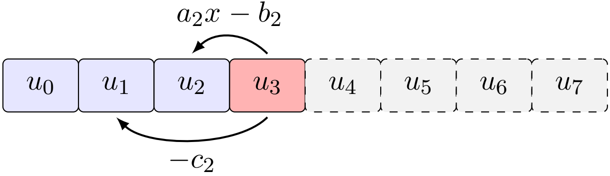
$$f(x) = \underbrace{\qquad\qquad\qquad\qquad \mathbf{P}_x^{(3)}\, \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$f(x) = (p_0(x), p_1(x), \dots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \dots, \hat{f}_N\right)$$
$$= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

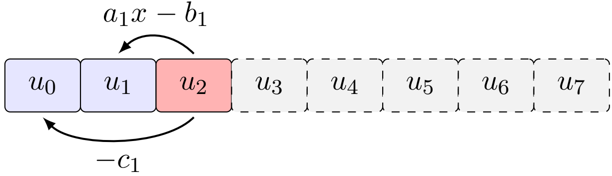
$$f(x) = \underbrace{\mathbf{P}_x^{(2)}\, \mathbf{P}_x^{(3)}\, \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$
\begin{aligned}
f(x) &= (p_0(x), p_1(x), \ldots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right) \\
&= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},
\end{aligned}
$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

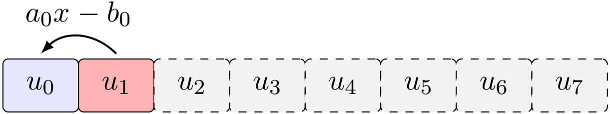
$$
f(x) = \underbrace{\qquad\qquad \mathbf{P}_x^{(1)}\, \mathbf{P}_x^{(2)}\, \mathbf{P}_x^{(3)}\, \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.
$$

Discrete Polynomial Transforms – Clenshaw Algorithm

The Clenshaw algorithm computes

$$f(x) = (p_0(x), p_1(x), \ldots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right)$$
$$= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



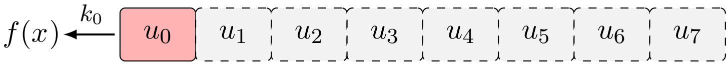
corresponds to

$$f(x) = \underbrace{\mathbf{P}_x^{(0)}\, \mathbf{P}_x^{(1)}\, \mathbf{P}_x^{(2)}\, \mathbf{P}_x^{(3)}\, \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.$$

The Clenshaw algorithm computes

$$
\begin{aligned}
f(x) &= (p_0(x), p_1(x), \ldots, p_N(x)) \cdot \left(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_N\right) \\
&= \sum_{k=0}^{N} \hat{f}_k p_k(x) = \mathbf{p}(x)^{\mathrm{T}}\, \hat{\mathbf{f}},
\end{aligned}
$$

essentially by factorizing $\mathbf{p}(x)^{\mathrm{T}}$ in particular way:



corresponds to

$$
f(x) = \underbrace{k_0\, \mathbf{P}_x^{(0)}\, \mathbf{P}_x^{(1)}\, \mathbf{P}_x^{(2)}\, \mathbf{P}_x^{(3)}\, \mathbf{P}_x^{(4)}\, \mathbf{P}_x^{(5)}\, \mathbf{P}_x^{(6)}}_{=\mathbf{p}(x)^{\mathrm{T}}}\, \hat{\mathbf{f}}.
$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Recap:

Clenshaw Algorithm for Multiple Nodes

Input: $\hat{f}_k$, $k = 0, 1, \ldots, N$, $x_j \in \mathbb{R}$, $j = 1, 2, \ldots, M$.

for $j = 1, 2, \ldots, M$ **do**

$$f(x_j) = k_0 \, \mathbf{P}_{x_j}^{(0)} \, \mathbf{P}_{x_j}^{(1)} \, \ldots \mathbf{P}_{x_j}^{(N-1)} \, \hat{\mathbf{f}}$$

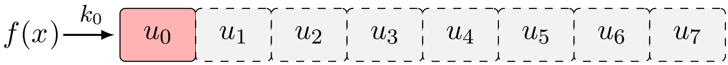
end for

Output: $f(x_j)$, $j = 1, 2, \ldots, M$

Time: $\mathcal{O}(NM)$, **Memory:** $\mathcal{O}(1)$.

Obtain transposed algorithm by transposing the matrix factorization.

Transposing the factorization:

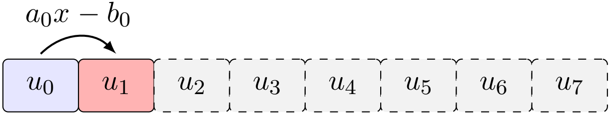


corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k_0}_{=\mathbf{p}(x)} f(x).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

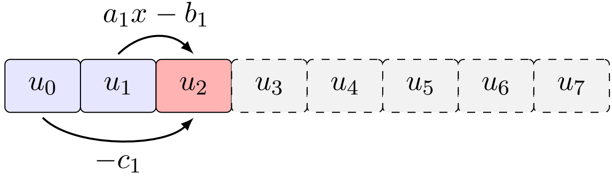
Transposing the factorization:



corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad {\mathbf{P}_x^{(0)}}^{\mathrm{T}} k_0}_{=\mathbf{p}(x)} f(x).$$

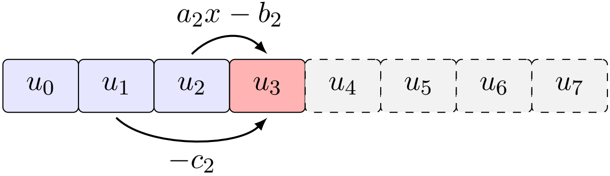
Transposing the factorization:



corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{P}_x^{(1)}{}^{\mathrm{T}}\, \mathbf{P}_x^{(0)}{}^{\mathrm{T}}\, k_0}_{=\mathbf{p}(x)} f(x).$$

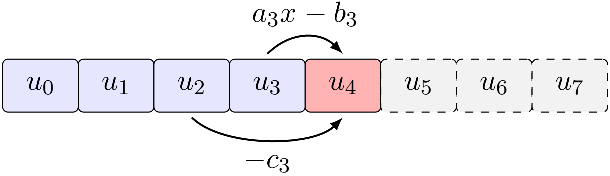
Transposing the factorization:



corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\qquad\qquad\qquad\qquad \mathbf{P}_x^{(2)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(1)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(0)}{}^{\mathrm{T}}\,k_0}_{=\mathbf{p}(x)}\,f(x).$$

Transposing the factorization:

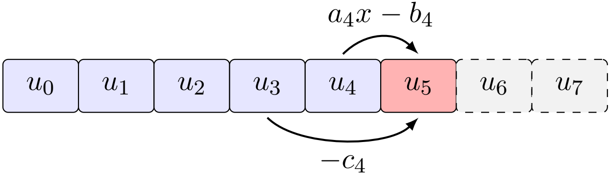


corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\qquad\qquad\qquad\qquad \mathbf{P}_x^{(3)}{}^{\mathrm{T}}\, \mathbf{P}_x^{(2)}{}^{\mathrm{T}}\, \mathbf{P}_x^{(1)}{}^{\mathrm{T}}\, \mathbf{P}_x^{(0)}{}^{\mathrm{T}}\, k_0}_{=\mathbf{p}(x)} f(x).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Transposing the factorization:

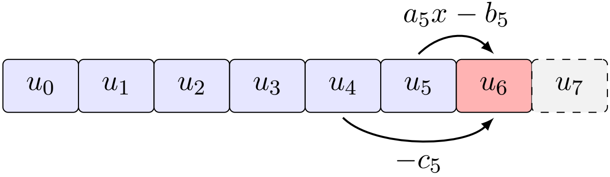


corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\mathbf{P}_x^{(4)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(3)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(2)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(1)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(0)}{}^{\mathrm{T}}\,k_0}_{=\mathbf{p}(x)}\,f(x).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Transposing the factorization:

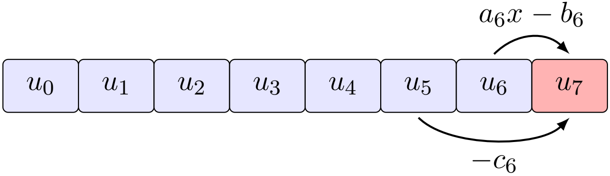


corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{{\mathbf{P}_x^{(5)}}^{\mathrm{T}} {\mathbf{P}_x^{(4)}}^{\mathrm{T}} {\mathbf{P}_x^{(3)}}^{\mathrm{T}} {\mathbf{P}_x^{(2)}}^{\mathrm{T}} {\mathbf{P}_x^{(1)}}^{\mathrm{T}} {\mathbf{P}_x^{(0)}}^{\mathrm{T}} k_0}_{=\mathbf{p}(x)} f(x).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Transposing the factorization:



corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{\mathbf{P}_x^{(6)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(5)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(4)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(3)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(2)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(1)}{}^{\mathrm{T}}\,\mathbf{P}_x^{(0)}{}^{\mathrm{T}}\,k_0}_{=\mathbf{p}(x)}\,f(x).$$

Discrete Polynomial Transforms – Clenshaw Algorithm

Transposing the factorization:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

corresponds to

$$\hat{\mathbf{h}}(x) = \underbrace{{\mathbf{P}_x^{(6)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(5)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(4)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(3)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(2)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(1)}}^{\mathrm{T}}\,{\mathbf{P}_x^{(0)}}^{\mathrm{T}}\,k_0}_{=\mathbf{p}(x)}\,f(x).$$

$\boxed{\text{In general: } \hat{\mathbf{h}}(x) \neq \hat{\mathbf{f}}}$

Discrete Polynomial Transforms – Clenshaw Algorithm

Transposed Clenshaw Algorithm for Multiple Nodes

Input: $x_j \in \mathbb{R}$, $f(x_j) \in \mathbb{R}$, $j = 1, 2, \ldots, M$.

for $j = 1, 2, \ldots, M$ **do**

$\quad \hat{\mathbf{h}}(x_j) = {\mathbf{P}_{x_j}^{(N-1)}}^{\mathrm{T}} {\mathbf{P}_{x_j}^{(N-2)}}^{\mathrm{T}} \ldots {\mathbf{P}_{x_j}^{(0)}}^{\mathrm{T}} k_0 \, f(x_j)$

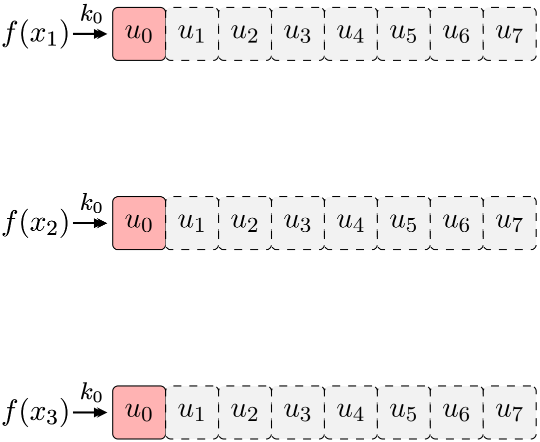
end for

$\hat{\mathbf{h}} = \hat{\mathbf{h}}(x_1) + \hat{\mathbf{h}}(x_2) + \ldots + \hat{\mathbf{h}}(x_M)$

Output: $\hat{\mathrm{h}}$, **Time:** $\mathcal{O}(NM)$, **Memory:** $\mathcal{O}(N)$.

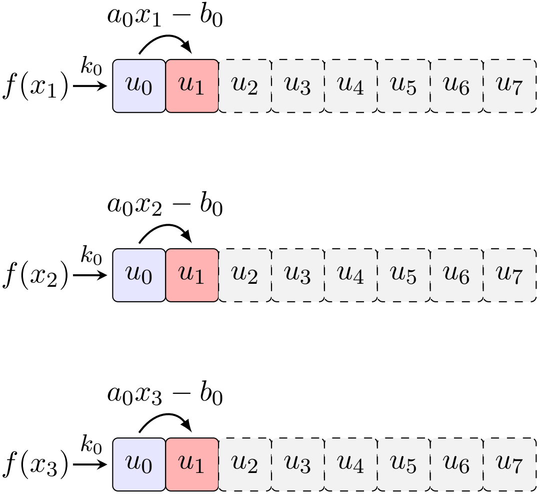
$\boxed{\text{In general: } \hat{\mathbf{h}} \neq \hat{\mathbf{f}}}$

Transposed Clenshaw algorithm for multiple nodes:



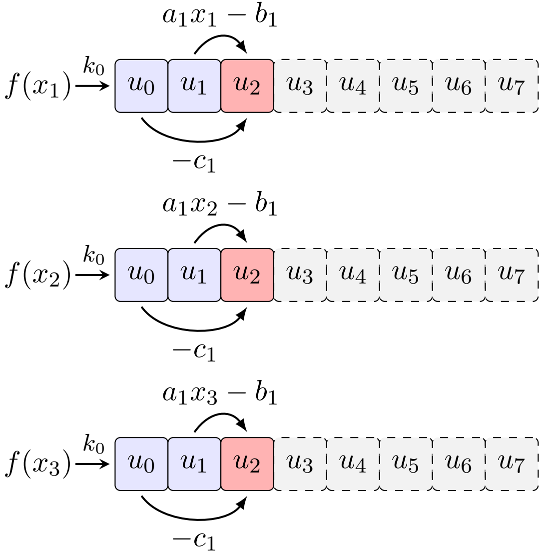
Discrete Polynomial Transforms – Clenshaw Algorithm

Transposed Clenshaw algorithm for multiple nodes:



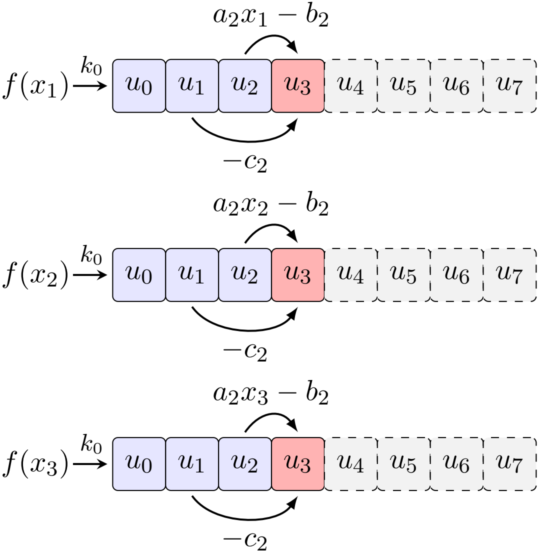
Discrete Polynomial Transforms – Clenshaw Algorithm

Transposed Clenshaw algorithm for multiple nodes:



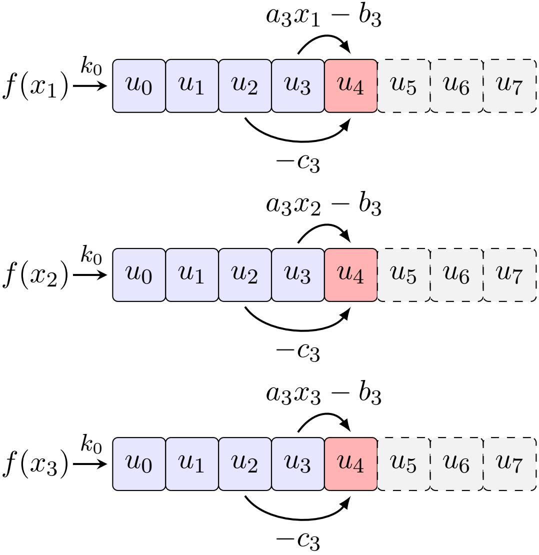
Discrete Polynomial Transforms – Clenshaw Algorithm

Transposed Clenshaw algorithm for multiple nodes:

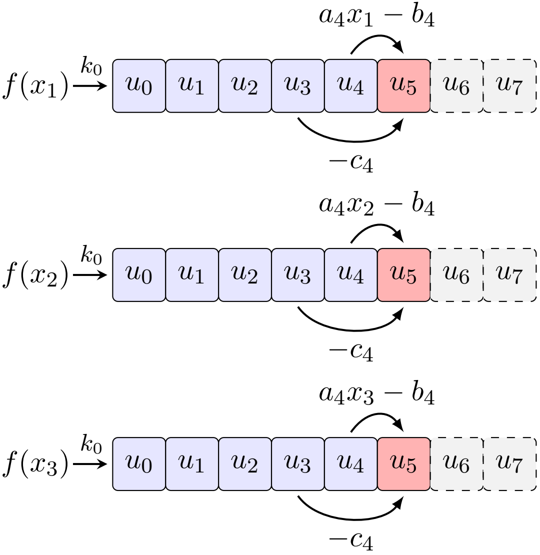


Discrete Polynomial Transforms – Clenshaw Algorithm

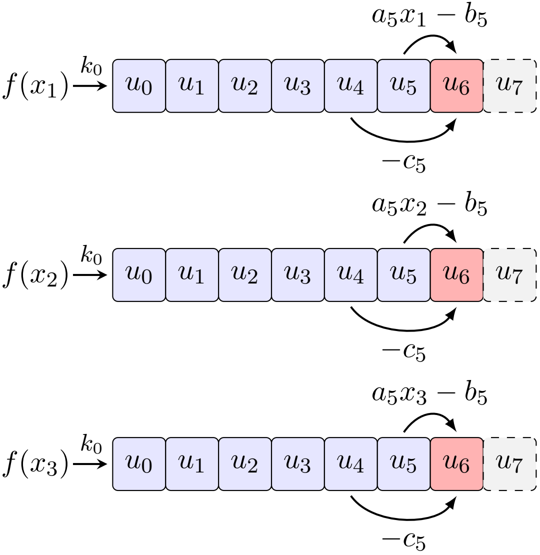
Transposed Clenshaw algorithm for multiple nodes:



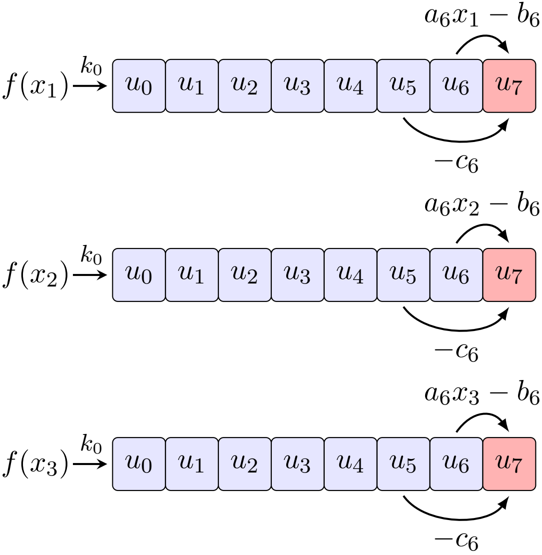
Transposed Clenshaw algorithm for multiple nodes:



Transposed Clenshaw algorithm for multiple nodes:



Transposed Clenshaw algorithm for multiple nodes:



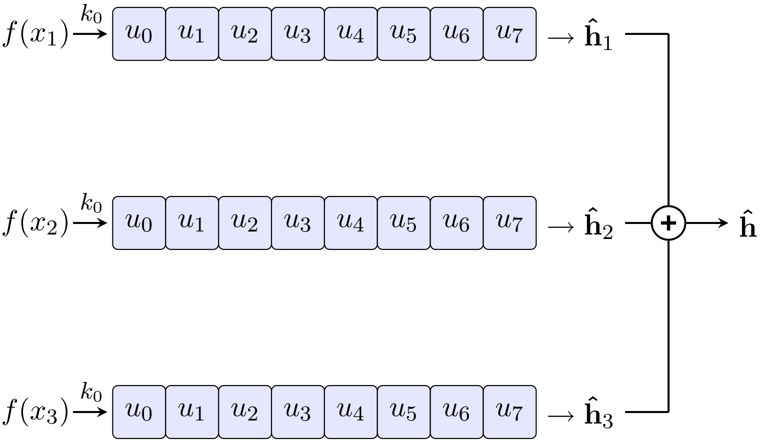
Transposed Clenshaw algorithm for multiple nodes:

$$f(x_1) \xrightarrow{k_0} \boxed{u_0}\boxed{u_1}\boxed{u_2}\boxed{u_3}\boxed{u_4}\boxed{u_5}\boxed{u_6}\boxed{u_7} \to \hat{\mathbf{h}}_1$$

$$f(x_2) \xrightarrow{k_0} \boxed{u_0}\boxed{u_1}\boxed{u_2}\boxed{u_3}\boxed{u_4}\boxed{u_5}\boxed{u_6}\boxed{u_7} \to \hat{\mathbf{h}}_2$$

$$f(x_3) \xrightarrow{k_0} \boxed{u_0}\boxed{u_1}\boxed{u_2}\boxed{u_3}\boxed{u_4}\boxed{u_5}\boxed{u_6}\boxed{u_7} \to \hat{\mathbf{h}}_3$$

Transposed Clenshaw algorithm for multiple nodes:



Program

Part I – Fourier Analysis and the FFT
Stefan, Monday, 14:15 – 16:00, Room U322

Part II – Orthogonal Polynomials
Jens, Tuesday, 12:15 – 14:00, Room U141 (Lecture Hall F)

Practice Session: 14:30 – 16:00, Room Y339b (Basics and Matlab Hands-On)

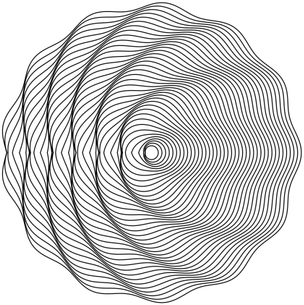
Part III – Fast Polynomial Transforms and Applications
Jens, Wednesday, 12:15 – 14:00, Room U345

Practice Session: 14:30 – 16:00, Room Y338c (C Library Hands-On)

Part IV – Fourier Transforms on the Rotation Group
Antje, Thursday, 14:15 – 16:00, Room U322

Part V – High Dimensions and Reconstruction
Stefan, Friday, 10:15 – 12:00, Room U322

Part III – Fast Polynomial Transforms and their Applications



Outline

1. Recap
2. Discrete Polynomial Transforms
3. Discrete Fourier Transform on the Sphere

Recap – Definition

Definition (Discrete Polynomial Transform)

Given a sequence of orthogonal polynomials/functions $\{p_n\}_{n\in\mathbb{N}}$, coefficients $\hat{f}_k$, $k = 0, \ldots, N$, and nodes x_j, $j = 1, \ldots, M$, compute

$$f_j = \sum_{k=0}^{N} \hat{f}_k p_k(x_j), \qquad j = 1, 2, \ldots, M.$$

Goal: fast and numerically stable algorithms

Recap – Matrix-Vector Notation

- The equations

$$f_j = \sum_{k=0}^{N} \hat{f}_k p_k(x_j), \qquad j = 1, 2, \ldots, M,$$

correspond to the matrix-vector product

$$\mathbf{f} = \mathbf{P}\,\hat{\mathbf{f}}, \qquad \mathbf{P} = (p_k(x_j))_{j=1,k=0}^{M,N} \in \mathbb{R}^{M\times(N+1)}.$$

- The *transposed* problem reads

$$\hat{h}_k = \sum_{j=1}^{M} f_j p_k(x_j), \qquad k = 0, 1, \ldots, N,$$

or, equivalently,

$$\tilde{\mathbf{f}} = \mathbf{P}^{\mathrm{T}}\,\mathbf{f}.$$

- Rationale: Can be used to recover the expansion coefficients $\hat{f}_k$ if a suitable quadrature formula to discretise the inner product $\hat{f}_k = \int_{-1}^{1} f(x) p_k(x)\,\mathrm{d}\lambda(x)$ is available.

Discrete Polynomial Transforms – Transposed Problem

The transposed problem reads: Given function samples f_j, compute the sums

$$\hat{h}_k = \sum_{j=1}^{M} f_j p_k(x_j), \qquad k = 0, 1, \ldots, N.$$

This corresponds to the matrix-vector product

$$\hat{\mathbf{h}} = \mathbf{P}^{\mathrm{T}} \mathbf{f}.$$

This can be used to recover the Fourier coefficients $\hat{\mathbf{f}}$ if function values $f_j = f(x_j)$ are known:

- Fourier coefficients $\hat{f}_k$ are calculated via the integrals

$$\hat{f}_k = \int_a^b f(x)\, p_k(x)\, w(x)\, \mathrm{d}x, \qquad k = 0, 1, \ldots, N.$$

Discrete Polynomial Transforms – Transposed Problem

- Fourier coefficients $\hat{f}_k$ are calculated via the integrals

$$\hat{f}_k = \int_a^b f(x)\, p_k(x)\, w(x)\, \mathrm{d}x, \qquad k = 0, 1, \ldots, N.$$

- Assume f to be a polynomial of degree at most N and that a quadrature rule with degree of exactness $2N$ for the nodes x_j, and with weights w_j is available. Then,

$$\hat{f}_k = \sum_{j=0}^{M} w_j f(x_j) p_k(x_j), \qquad k = 0, 1, \ldots, N.$$

- In matrix-vector notation, this reads

$$\hat{\mathbf{f}} = \mathbf{P}^{\mathrm{T}}\, \mathbf{W}\, \mathbf{f}, \qquad \mathbf{W} = \mathrm{diag}(w_j)_{j=1}^{M}.$$

Discrete Polynomial Transforms – Transposed Problem

A fast algorithm for the transposed Problem is needed to recover Fourier coefficients.

Summary:

- If f is a polynomial of degree at most N, then the exact Fourier coefficients are recovered.
- If f is not a polynomial of degree at most N, then the **computed** Fourier coefficients $\hat{\mathbf{f}}$ will contain some aliasing error (depending on f).
- In both situations, the computation of

$$\hat{\mathbf{f}} = \mathbf{P}^{\mathrm{T}}\,\mathbf{W}\,\mathbf{f}, \qquad \mathbf{W} = \operatorname{diag}(w_j)_{j=1}^{M}.$$

  computes a **projection** of f onto the polynomials $p_0, p_1, \ldots, p_N$.
- This projection, however, is usually not the **orthogonal projection**, but can often be shown to have the same order of convergence.

Discrete Polynomial Transforms – Algorithms

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Cascade Summation

(Driscoll, Healy, 1994; Potts, Steidl, Tasche, 1998;
Potts 2003; Keiner, Potts, 2006)

Discrete Polynomial Transforms – Cascade Summation

The goal is to efficiently evaluate

$$f(x) = \sum_{k=0}^{N} \hat{f}_k p_k(x).$$

The Clenshaw algorithm makes use of the three-term recurrence

$$p_{n+1}(x) = (a_n x - b_n) p_n(x) - c_n p_{n-1}(x), \qquad n = 0, 1, \dots ,$$

to evaluate the linear combination at one node at a time.

This can be made more efficient in two ways:

1. Use a more general form of the three-term recurrence.
2. Compute with polynomials instead of numbers
   (Make as many computations node-independent as possible).

Definition (Associated Polynomials)

Let $\{p_n\}_{n\in\mathbb{N}_0}$ be a sequence of orthogonal polynomials. Then the polynomials $\{p_n^{[m]}\}$, $m \in \mathbb{N}_0$, defined by the recurrence and initial conditions

$$p_{n+1}^{[m]}(x) = (a_{n+m}x - b_{n+m})p_n^{[m]}(x) - c_{n+m}p_{n-1}^{[m]}, \quad n = 0, 1, \ldots,$$
$$p_{-1}^{[m]}(x) = 0, \quad p_0^{[m]}(x) = k_0,$$

are called *associated polynomials of order* m.

One can prove that the sequence $\{p_n^{[m]}\}_{n\in\mathbb{N}_0}$ is again orthogonal.

(But usually, the corresponding measure $\mathrm{d}\lambda$ is not known.)

Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:

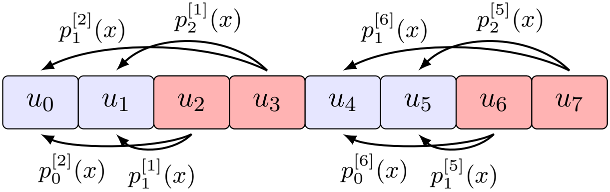
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:



Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

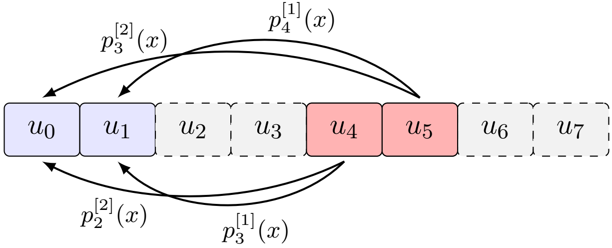
Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:



Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

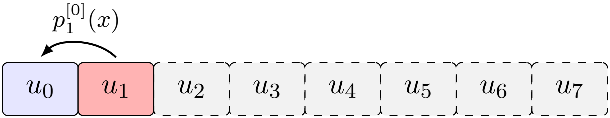
Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:



Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

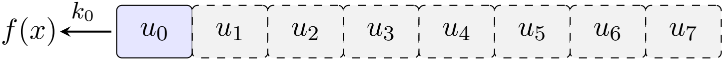
Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:



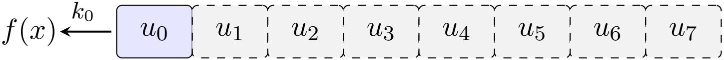
Discrete Polynomial Transforms – Cascade Summation

Theorem (Generalized Three-Term Recurrence)

A sequence of orthogonal polynomials $\{p_n\}_{n\in\mathbb{N}_0}$ satisfies the generalized three-term recurrence

$$p_{n+c}(x) = p_c^{[n]}(x)p_n(x) - c_n p_{c-1}^{[n+1]}(x)p_{n-1}(x).$$

The generalized recurrence allows to modify the procedure:



This does not yet yield a faster algorithm!

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

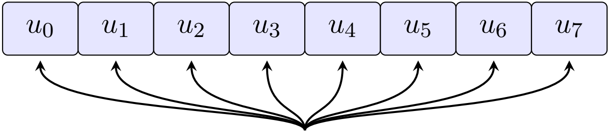
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Discrete Polynomial Transforms – Polynomial Multiplication

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

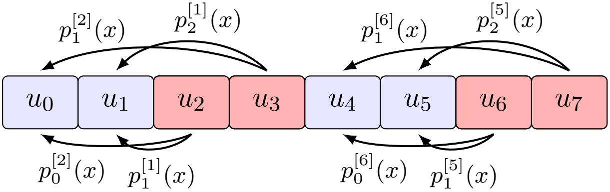


Polynomials of degree at most 0 (i.e. constants).

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:



Discrete Polynomial Transforms – Polynomial Multiplication

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

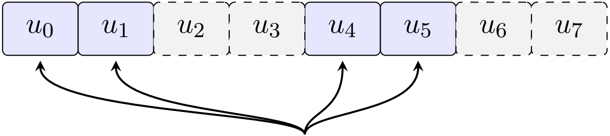
To make computations independent of x, one needs to compute with polynomials instead of numbers:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

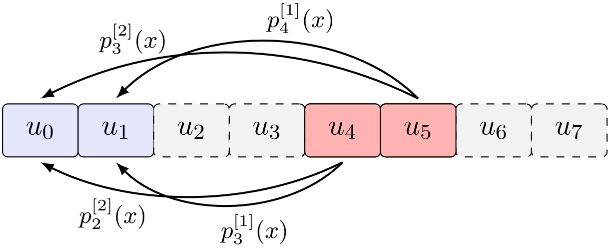


Polynomials of degree at most 3.

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:



Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

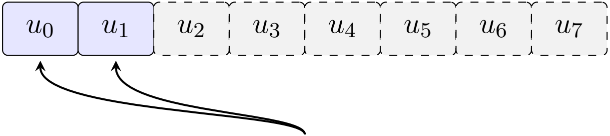
To make computations independent of x, one needs to compute with polynomials instead of numbers:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

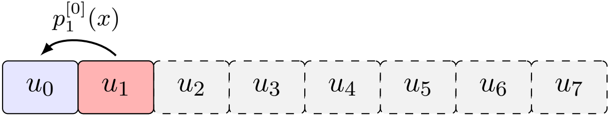


Polynomials of degree at most 7.

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:



Discrete Polynomial Transforms – Polynomial Multiplication

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

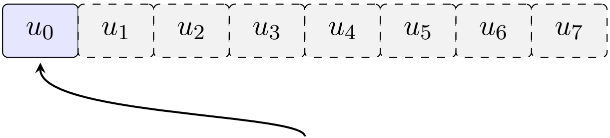
To make computations independent of x, one needs to compute with polynomials instead of numbers:

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:



Polynomial of degree exactly 7.

Discrete Polynomial Transforms – Polynomial Multiplication

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

Divide computation into two parts:

- Node-independent part (compute with actual polynomials)
- Node-dependent part (final evaluation)

To make computations independent of x, one needs to compute with polynomials instead of numbers:

How to compute with polynomials instead of numbers?

Discrete Polynomial Transforms – Polynomial Multiplication

Represent polynomial p of degree n by its Chebyshev expansion

$$p = \sum_{k=0}^{n} \alpha_k T_k, \qquad\qquad p \sim (\alpha_k)_{k=0}^{n},$$

or, equivalently, by its values at Chebyshev nodes of order n,

$$p(\tau_{n,j}) = p\Big(\cos\Big(\tfrac{2j+1}{2n+2}\pi\Big)\Big), \qquad\qquad p \sim \big(p(\tau_{n,j})\big)_{j=0}^{n}.$$

How to convert between both representations?

Read: Trefethen, "Computing numerically with functions instead of numbers"

Discrete Polynomial Transforms – Polynomial Multiplication

Efficient conversion via discrete cosine transforms:

$$\boldsymbol{\alpha} = (\alpha_k)_{k=0}^n \longrightarrow \mathbf{p} = \underbrace{\mathbf{C}}_{\text{Discrete cosine transform (DCT-III), } \mathcal{O}(n \log n)} \boldsymbol{\alpha} \longrightarrow \mathbf{p} = \big(p(\tau_{n,j})\big)_{j=0}^n$$

For larger sizes, the vector $\boldsymbol{\alpha}$ can be padded with zeros.

$$\mathbf{p} = \big(p(\tau_{n,j})\big)_{j=0}^n \longrightarrow \boldsymbol{\alpha} = \underbrace{\tfrac{1}{n}\mathbf{C}^{\mathrm{T}}}_{\text{Discrete cosine transform (DCT-II), } \mathcal{O}(n \log n)} \mathbf{p} \longrightarrow \boldsymbol{\alpha} = (\alpha_k)_{k=0}^n$$

Discrete Polynomial Transforms – Polynomial Multiplication

Input	$p \in \mathbb{P}_n$	$q \in \mathbb{P}_n$
Chebyshev coefficients	$\alpha_0, \ldots, \alpha_n$	$\beta_0, \ldots, \beta_n$
Values at Chebyshev nodes	$p(\tau_{2n,0}), \ldots, p(\tau_{2n,2n})$	$q(\tau_{2n,0}), \ldots, q(\tau_{2n,2n})$
	$\odot$	
Pointwise multiplication	$r(\tau_0), \ldots, r(\tau_{2n,2n})$	
Chebyshev coefficients	$\gamma_0, \ldots, \gamma_n$	
Output	$r = p \cdot q \in \mathbb{P}_{2n}$	

Discrete Polynomial Transforms – Polynomial Multiplication

After the Clenshaw procedure, one obtains the Chebyshev expansion of f:

$$f = \sum_{k=0}^{N} \hat{g}_k T_k \xleftarrow{k_0} \boxed{u_0} \; u_1 \; u_2 \; u_3 \; u_4 \; u_5 \; u_6 \; u_7$$

This can be evaluated at arbitrary target nodes x_j, $j = 1, 2, \ldots, M$ with a nonequispaced fast cosine transform (NFCT).

Cost:

- Node-independent part:

  Time $\mathcal{O}(N \log^2 N)$, Memory $\mathcal{O}(N \log N)$,

- Node-dependent part:

  Time $\mathcal{O}(N \log N + \log(1/\varepsilon)M)$,
  Memory $\mathcal{O}(1) - \mathcal{O}(N + \log(1/\varepsilon)M)$.

Discrete Polynomial Transforms – Cascade Summation

Summary:

- Time: $\mathcal{O}(N^2 \log N + \log(1/\varepsilon)M)$ ✓
- Memory: $\mathcal{O}(N \log N) - \mathcal{O}(N \log N + \log(1/\varepsilon)M)$ ✓
- Numerically unstable ✗
  - Depends on polynomial system,
  - Problems already for small sizes $n > 16$.
  - Instabilities caused by boundary layers of associated polynomials in combination with the use of a global interpolation method (DCT)
- Requirements: Three-term recurrence
- Best suited for polynomials on an interval
- Original Driscoll-Healy algorithm is the transposed version

Discrete Polynomial Transforms – Cascade Summation

Summary:

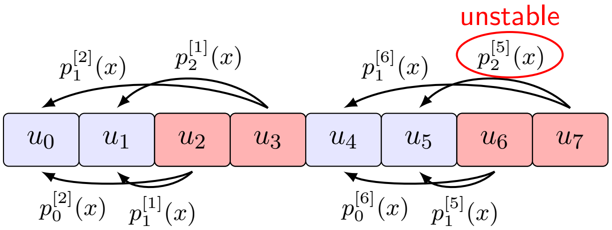
- Time: $\mathcal{O}(N^2 \log N + \log(1/\varepsilon)M)$ ✓
- Memory: $\mathcal{O}(N \log N) - \mathcal{O}(N \log N + \log(1/\varepsilon)M)$ ✓
- Numerically unstable ✗
  - Depends on polynomial system,
  - Problems already for small sizes $n > 16$.
  - Instabilities caused by boundary layers of associated polynomials in combination with the use of a global interpolation method (DCT)
- Requirements: Three-term recurrence
- Best suited for polynomials on an interval
- Original Driscoll-Healy algorithm is the transposed version

Can the stability issues be remedied?

Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

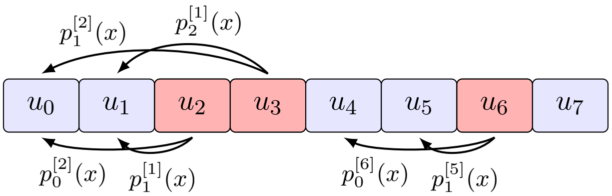
- Identify unstable multiplication steps (boundary layers)



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

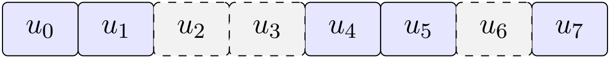
- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

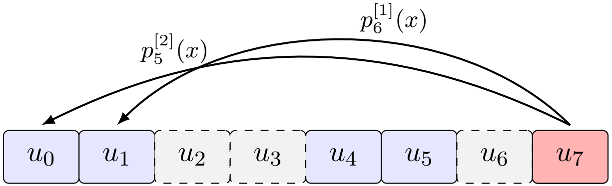
- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

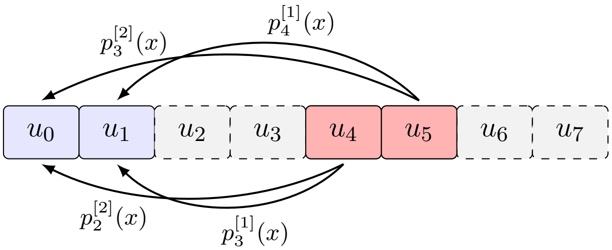
- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7

Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

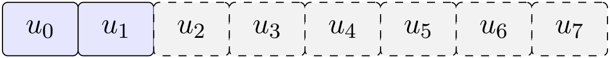
- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

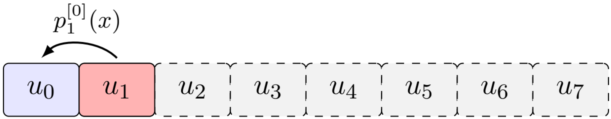
- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before



Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before



Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before

$$f \leftarrow \boxed{u_0}\ u_1\ u_2\ u_3\ u_4\ u_5\ u_6\ u_7$$

Discrete Polynomial Transforms – Stabilization

Idea (Driscoll, Healy, Rockmore, 1996; Potts, Steidl, Tasche, 2002):

- Identify unstable multiplication steps (boundary layers)
- Suspend unstable step, process rest as usual
- Map relegated polynomial to first two polynomials (expensive)
- Continue with rest as before

$$f \leftarrow \boxed{u_0}\;u_1\;u_2\;u_3\;u_4\;u_5\;u_6\;u_7$$

- Cost per stabilization step $\mathcal{O}(N \log N)$.

Discrete Polynomial Transforms – Cascade Summation

Stabilized Version

Summary:

- Time: $\mathcal{O}(N^2 \log N + \log(1/\varepsilon)M)$ (maybe) ✓
- Memory: $\mathcal{O}(N \log N) - \mathcal{O}(N \log N + \log(1/\varepsilon)M)$ ✓
- More stable than unstabilized version ✓
- Numerical stability not proven ✗
  - Depends on polynomial system
  - boundary layers still a problem
- Stabilization steps can destroy asymptotic cost bound
- $\mathcal{O}(N^2 \log N)$ behaviour lost, if more than $\mathcal{O}(\log N)$ stabilization steps. ✗
- Requirements: Three-term recurrence
- Best suited for polynomials on an interval

Discrete Polynomial Transforms – Algorithms

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Cascade Summation

(Driscoll, Healy, 1994; Potts, Steidl, Tasche, 1998; Potts 2003; Keiner, Potts, 2006)

Tridiagonal Matrices

(Tygert, 2005)

Discrete Polynomial Transforms – Tridiagonal Matrices

$$f(x) = \sum_{k=0}^{N} \hat{f}_k p_k(x).$$

Split computation into two parts:

1. Evaluate linear combination at zeros of p_{N+1}
2. Interpolate to arbitrary nodes x_j, $j = 1, 2, \ldots, M$.

First step: Recall eigenvector matrix of the Jacobi matrix J_{N+1} is

$$\mathbf{Q} = \left(p_k\left(\tau_{N+1,j}\right)\right)_{j=0,k=0}^{N,N}.$$

Thus, evaluation of $f(x)$ at zeros $\tau_{N+1,j}$ of p_{N+1} is equivalent to

$$\tilde{\mathbf{f}} = \mathbf{Q}\,\hat{\mathbf{f}}, \qquad \tilde{\mathbf{f}} = \left(f\left(\tau_{N+1,j}\right)\right)_{j=0}^{N}.$$

Can use fast divide-and-conquer method (Gu, Eisenstat, 1994) to do this step in $\mathcal{O}(N \log N \log(1/\varepsilon))$ time/memory (see later).

Discrete Polynomial Transforms – Tridiagonal Matrices

Second step: Must interpolate from zeros of p_{N+1} to target nodes x_j. Use barycentric interpolation formula

$$f(x) = \ell(x) \sum_{k=0}^{N} \frac{w_k f(\tau_{N+1,k})}{x - \tau_{N+1,k}},$$

$$\ell(x) = \prod_{k=0}^{N} (x - \tau_{N+1,k}), \quad w_k = \prod_{\substack{j=0 \\ j \neq k}}^{N} \frac{1}{\tau_{N+1,k} - \tau_{N+1,j}}.$$

- The factor $\ell(x)$ does not depend on the target nodes x_j.
- The sum can be evaluated at all target nodes using the *Fast Multipole Method* (Greengard, Rokhlin, 1987) in

$$\mathcal{O}(N \log(1/\varepsilon)) \text{ time/memory.}$$

Read: Trefethen, “Barycentric Lagrange Interpolation”;

Dutt, Gu, Rokhlin “Fast algorithms for polynomial interpolation, integration, and differentiation”

Summary:

- Time: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Memory: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Relatively stable (numerical experience) ✓
- Suited for all classical orthogonal polynomials
- Requirements: Three-term recurrence
- Should also work well for polynomials on unbounded intervals (Laguerre, Hermite)

Discrete Polynomial Transforms – Algorithms

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Cascade Summation

(Driscoll, Healy, 1994; Potts, Steidl, Tasche, 1998;
Potts 2003; Keiner, Potts, 2006)

Tridiagonal Matrices

(Tygert, 2005)

Semiseparable Matrices

(K., 2007)

Want to evaluate

$$f(x) = \sum_{k=0}^{N} \hat{f}_k p_k(x).$$

Split computation into two parts (similar to Cascade Summation):

1. Convert to Chebyshev expansion (node-independent)

$$f(x) = \sum_{k=0}^{N} \hat{g}_k T_k(x).$$

2. Evaluate at target nodes x_j using an NFCT/NFFT.

More general view of first step:

$\boxed{\text{Change of basis from } p_k \text{ to } q_k.}$

Discrete Polynomial Transforms – Semiseparable Matrices

The first step corresponds to the linear transformation

$$\hat{\mathbf{g}} = \mathbf{B}\hat{\mathbf{f}}, \text{ with } \mathbf{B} = \left(h_j^{-1}\langle q_j, p_k\rangle\right)_{j,k=0}^{N},$$

where

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\,\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x, \quad h_j = \langle q_j, q_j\rangle.$$

Idea:

- Construct a matrix $\mathbf{G}$ such that $\mathbf{B}$ contains its eigenvectors.
- Identify the structure of $\mathbf{G}$ to get a fast algorithm for multiplication with $\mathbf{B}$.

How to construct the matrix $\mathbf{G}$?

$\boxed{\text{Change of basis from } p_k \text{ to } q_k.}$

Need the following ingredients:

- Three-term recurrence

$$q_{n+1}(x) = (a_n x - b_n) q_n(x) - c_n q_{n-1}(x), \qquad n = 0, 1, \ldots\,,$$
$$q_{-1}(x) = 0, \qquad q_0(x) = k_0.$$

- Differential equations (σ needs to be the same!)

$$\sigma p_n'' + \tau p_n' + \lambda_n p_n = 0, \qquad \sigma q_n'' + \tilde{\tau} q_n' + \tilde{\lambda}_n q_n = 0$$

- Derivative identity (express q_n' through q_k, $k = 0, 1, \ldots, n-1$)

$$\frac{\mathrm{d}}{\mathrm{d}x} q_n = A_n \sum_{k=0}^{n-1} B_k q_k + C_n \sum_{k=0}^{n-1} D_k q_k,$$

A preliminary form of the derivative identities can be derived from the Rodrigues formula:

Lemma

The following identities for monic orthogonal polynomials are true:

$$\frac{\mathrm{d}}{\mathrm{d}x}\bar{L}_n^{(\alpha)} = n\bar{L}_{n-1}^{(\alpha+1)},$$
$$\frac{\mathrm{d}}{\mathrm{d}x}\bar{P}_n^{(\alpha,\beta)} = n\bar{P}_{n-1}^{(\alpha+1,\beta+1)}.$$

Proof.

Use the Rodrigues formula and identify terms. ∎

Example: For the Laguerre polynomials, the Rodrigues formula is

$$\frac{\mathrm{d}^m}{\mathrm{d}x^m} L_n^{(\alpha)}(x) = \frac{1}{n!} \frac{x^{-\alpha} \mathrm{e}^x}{x^m} \frac{\mathrm{d}^{n-m}}{\mathrm{d}x^{n-m}} \left(x^{\alpha+n} \mathrm{e}^{-x} \right).$$

Therefore,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} L_n^{(\alpha)}(x) &= \frac{1}{n!} x^{-(\alpha+1)} \mathrm{e}^x \frac{\mathrm{d}^{n-1}}{\mathrm{d}x^{n-1}} \left(x^{\alpha+n} \mathrm{e}^{-x} \right), \\ L_{n-1}^{(\alpha+1)}(x) &= \frac{1}{(n-1)!} x^{-(\alpha+1)} \mathrm{e}^x \frac{\mathrm{d}^{n-1}}{\mathrm{d}x^{n-1}} \left(x^{\alpha+n} \mathrm{e}^{-x} \right). \end{aligned}$$

Thus, for the monic variants, one obtains

$$\frac{\mathrm{d}}{\mathrm{d}x} \bar{L}_n^{(\alpha)} = n \bar{L}_{n-1}^{(\alpha+1)}.$$

The actual derivative identity is given in the following theorem:

Theorem

Let $n \geq 1$. Then for the Laguerre polynomials, one has

$$\frac{\mathrm{d}}{\mathrm{d}x} \bar{L}_n^{(\alpha)} = A_n \sum_{k=0}^{n-1} B_k \bar{L}_k^{(\alpha)},$$

with

$$A_n := (-1)^{n-1} n!, \qquad B_k := \frac{(-1)^k}{k!}.$$

Before we can prove the theorem, we need yet another result:

Theorem

For the Laguerre polynomials $L_n^{(\alpha)}$, we have

$$\bar{L}_n^{(\alpha)} = \sum_{k=0}^{n} \frac{(-1)^n n!}{(-1)^k k!} \bar{L}_k^{(\alpha-1)}$$

Comment: The proof of this theorem is another story. Identities of this form have been known for a long time, e.g. Szegő, 1975 gives a formula for Gegenbauer polynomials. The formula for the Laguerre case is derived in my upcoming PhD thesis, but almost certainly has been known before.

Discrete Polynomial Transforms – Derivative Identity

Now, the actual proof is simple:

Proof.

Use the results

$$\frac{\mathrm{d}}{\mathrm{d}x}\bar{L}_n^{(\alpha)} = n\bar{L}_{n-1}^{(\alpha+1)}, \text{ and } \bar{L}_{n-1}^{(\alpha+1)} = \sum_{k=0}^{n-1} \frac{(-1)^{n-1}(n+1)!}{(-1)^k k!}\bar{L}_k^{(\alpha)},$$

to verify that

$$\frac{\mathrm{d}}{\mathrm{d}x}\bar{L}_n^{(\alpha)} = n\bar{L}_{n-1}^{(\alpha+1)} = n\sum_{k=0}^{n-1} \frac{(-1)^{n-1}(n-1)!}{(-1)^k k!}L_k^{(\alpha)}$$

∎

$\boxed{\text{Change of basis from } p_k \text{ to } q_k.}$

Constructing $\mathbf{G}$:

$$
\begin{array}{rcc}
\text{Orthogonal polynomials (from same family)} & p_n = P_n^{(\alpha,\beta)} & q_n = P_n^{(\alpha',\beta')} \\
 & \downarrow & \downarrow \\
\textbf{Differential equation} & \begin{array}{c}\sigma p_n'' + \tau p_n' + \lambda_n p_n = 0, \\ \mathcal{D}y = -\sigma y'' - \tau y'\end{array} & \begin{array}{c}\sigma q_n'' + \tilde{\tau} q_n' + \tilde{\lambda}_n q_n = 0, \\ \tilde{\mathcal{D}}y = -\sigma y'' - \tilde{\tau} y'\end{array} \\
 & \searrow & \swarrow \\
\text{Differential operator with simple structure} & \mathcal{D}^{\Delta} y := \mathcal{D}y - \tilde{\mathcal{D}}y = (\tau - \tilde{\tau}) y' & \\
 & \Big\downarrow \textbf{ Derivative identity, three-term recurrence} & \\
\text{Matrix with certain structure} & \mathbf{G} &
\end{array}
$$

With $h_j = \langle q_j, q_j \rangle$, define the matrix $\mathbf{G}$ as

$$\mathbf{G} := \left(h_j^{-1} \langle q_j, \mathcal{D}(q_k) \rangle\right)_{j,k=0}^{N}.$$

Compare this to

$$\mathbf{B} = \left(h_j^{-1} \langle q_j, p_k \rangle\right)_{j,k=0}^{N}.$$

That is:

The source polynomial p_k has been replaced by the corresponding differential operator $\mathcal{D}$ applied to the target polynomial q_k, i.e., $\mathcal{D}(q_k)$.

Theorem (K., 2008)

The matrix $\mathbf{B} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_N)$ contains the eigenvectors of $\mathbf{G}$. Moreover, the corresponding eigenvalue for $\mathbf{b}_k$ is λ_k.

Proof.

By definition, $p_k = \sum_{j=0}^{N} b_{j,k} q_j$. Denote by $(\mathbf{G}\,\mathbf{b}_k)_j$ the $(j+1)$st component of the product $\mathbf{G}\,\mathbf{b}_k$, and by

$$\mathbf{b}_k = (b_{0,k}, b_{1,k}, \ldots, b_{N,k})^{\mathrm{T}}$$

the $(k+1)$st column of $\mathbf{G}$. Then,

$$\begin{aligned}(\mathbf{G}\,\mathbf{b}_k)_j = \sum_{\ell=0}^{N} g_{j,\ell} b_{\ell,k} \qquad &= \sum_{\ell=0}^{N} h_j^{-1} \langle q_j, \mathcal{D}(q_\ell)\rangle b_{\ell,k} \\ = h_j^{-1} \langle q_j, \mathcal{D}(\sum_{\ell=0}^{N} b_{\ell,k} q_\ell)\rangle &= h_j^{-1} \langle q_j, \mathcal{D}(p_k)\rangle\end{aligned}$$

■

Proof (continued).

We have

$$\left(\mathbf{G}\,\mathbf{b}_k\right)_j = h_j^{-1}\langle q_j, \mathcal{D}(p_k)\rangle$$

Use that p_k is an eigenfunction of $\mathcal{D}$, i.e, $\mathcal{D}(p_k) = \lambda_k p_k$,

$$\left(\mathbf{G}\,\mathbf{b}_k\right)_j = \lambda_k h_j^{-1}\langle q_j, p_k\rangle$$

Then work backward until

$$\left(\mathbf{G}\,\mathbf{b}_k\right)_j = \lambda_k \sum_{\ell=0}^{N} g_{j,\ell} b_{\ell,k} = \lambda_k \left(\mathbf{G}\,\mathbf{b}_k\right)_j .$$

$\blacksquare$

Does $\mathbf{G}$ have any structure to exploit?

To characterize $\mathbf{G}$, must define classes of structured matrices

Definition

A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called *generator representable semiseparable of semiseparability rank* r, if there exist two matrices $\mathbf{R}_1$ and $\mathbf{R}_2$, both of rank r, such that

$$\mathrm{triu}(\mathbf{A}) = \mathbf{R}_1, \qquad \mathrm{tril}(\mathbf{A}) = \mathbf{R}_2.$$

Matlab notation: $\mathrm{triu}(\mathbf{A})$ = upper triangular part of $\mathbf{A}$, $\mathrm{tril}(\mathbf{A})$ = lower triangular part of $\mathbf{A}$.
One can write

$$\mathbf{A} = \mathrm{triu}(\mathbf{X}\,\mathbf{Y}^{\mathrm{T}}, 1) + \mathrm{tril}(\mathbf{W}\,\mathbf{Z}^{\mathrm{T}}),$$

with the *generators*

$$\mathbf{X}, \mathbf{Y}, \mathbf{W}, \mathbf{Z} \in \mathbb{R}^{n \times r}.$$

Semiseparable matrices in a nutshell:

$$\mathbf{A} = \mathrm{triu}(\mathbf{X}\,\mathbf{Y}^{\mathrm{T}}, 1) + \mathrm{tril}(\mathbf{W}\,\mathbf{Z}^{\mathrm{T}}),$$

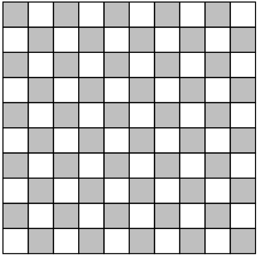
- Generator-representable semiseparable matrices are the inverses of irreducible banded matrices.
- Symmetric case: $\mathbf{X} = \mathbf{Z}$, $\mathbf{Y} = \mathbf{W}$.
- Different ranks: (p, q)-generator representable semiseparable
- Triagular forms: $p = 0$ or $q = 0$
- Diagonal free of choice: diagonal plus (p, q)-generator representable semiseparable
- Super/Subdiagonal free of choice: bi-diagonal plus (p, q)-generator representable semiseparable

Read: Vandebril, Van Barel, Golub, Mastronardi, "A bibliography on semiseparable matrices"

Definition

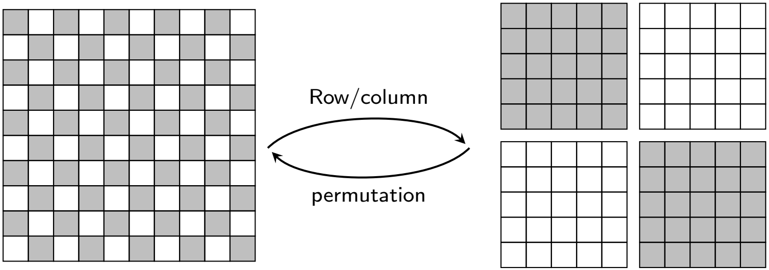
A matrix $\mathbf{A} = (a_{i,j}) \in \mathbb{R}^{n \times m}$ is called *checker board-like*, if the following condition is satisfied:

$$a_{i,j} = 0, \text{ if } i + j \text{ odd.}$$

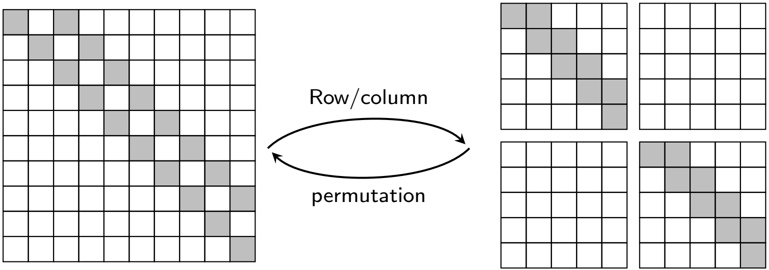


Checker board-like matrices can be interpreted as two interwoven/interlaced matrices.

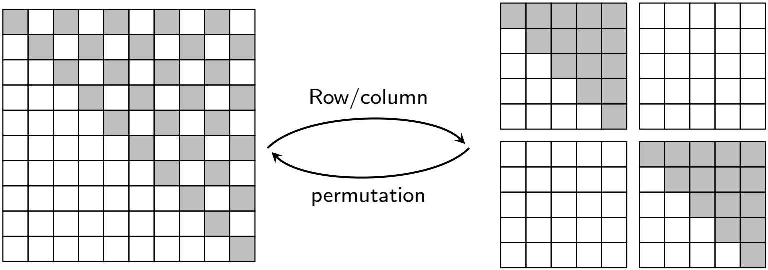
Checker-board like matrices can be split into two matrices that can be treated independently:



Checker-board like matrices can be split into two matrices that can be treated independently:



Checker-board like matrices can be split into two matrices that can be treated independently:



One can prove the following relations (K. 2008)

- If $p_n = L_n^{(\alpha)}$ and $q_n = L_n^{(\alpha')}$, then
  $\mathbf{G}$ is bi-diagonal plus $(1, 0)$-generator representable semiseparable
- If $p_n = C_n^{(\alpha)}$ and $q_n = C_n^{(\alpha')}$, then
  $\mathbf{G}$ is checker board-like diagonal plus $(1, 0)$-generator representable semiseparable
- If $p_n = P_n^{(\alpha,\beta)}$ and $q_n = P_n^{(\alpha',\beta')}$, then
  $\mathbf{G}$ is bi-diagonal plus $(2, 0)$-generator representable semiseparable

Discrete Polynomial Transforms – Semiseparable Matrices

Lemma (As an example)

Let $\{p_n\}_{n\in\mathbb{N}_0}$ be orthogonal polynomials that satisfy

$$\frac{\mathrm{d}}{\mathrm{d}x}p_n(x) = A_n \sum_{k=0}^{n-1} B_k p_k(x).$$

Then the matrix $\tilde{\mathbf{G}} = (\tilde{g}_{j,k})_{j,k=0}^{N} \in \mathbb{R}^{(N+1)\times(N+1)}$,

$$\tilde{g}_{j,k} = h_j^{-1}\langle p_j, \frac{\mathrm{d}}{\mathrm{d}x}p_k\rangle$$

is diagonal plus $(1,0)$-generator semiseparable matrix, i.e.,

$$\begin{aligned}
\mathbf{G} &= \mathrm{diag}(\mathbf{0}) + \mathrm{triu}(\mathbf{B}\,\mathbf{A}^{\mathrm{T}}, 1),\\
\mathbf{0} &= (0, 0, \ldots, 0)^{\mathrm{T}},\\
\mathbf{A} &= (A_0, A_1, \ldots, A_n)^{\mathrm{T}},\\
\mathbf{B} &= (B_0, B_1, \ldots, B_n)^{\mathrm{T}}.
\end{aligned}$$

Proof.

By orthogonality, $\langle p_j, \frac{\mathrm{d}}{\mathrm{d}x} p_k \rangle = 0$ if $k < j - 1$, i.e. $\mathbf{G}$ is a strict upper triangular matrix. For the rest of the entries, we have

$$
\begin{aligned}
g_{j,k} &= h_j^{-1} \langle p_j, \frac{\mathrm{d}}{\mathrm{d}x} p_k \rangle \\
&= h_j^{-1} \langle p_j, A_k \sum_{\ell=0}^{k-1} B_\ell p_\ell \rangle \\
&= h_j^{-1} A_k \sum_{\ell=0}^{k-1} B_\ell \langle p_j, p_\ell \rangle \\
&= h_j^{-1} A_k B_j h_j \\
&= B_j A_k.
\end{aligned}
$$

∎

Divide-and-conquer algorithm (K. 2008) for eigendecomposition of

$$\mathbf{A} = \mathrm{diag}(\mathbf{d}) + \mathrm{triu}(\mathbf{x}\,\mathbf{y}^{\mathrm{T}}).$$

Divide Phase

The matrix $\mathbf{A}$ can be written as

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix} + \mathbf{x}'\,\mathbf{y}'^{\mathrm{T}},$$

where $\mathbf{A}_1$, and $\mathbf{A}_2$ are of same type as $\mathbf{A}$ and

$$\mathbf{x}' = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{0} \end{pmatrix}, \qquad \mathbf{y}' := \begin{pmatrix} \mathbf{0} \\ \mathbf{y}_2 \end{pmatrix}.$$

Conquer phase

Assume, $\mathbf{A}_1 = \mathbf{Q}_1\,\mathbf{D}_1\,\mathbf{Q}_1^{-1}$, $\mathbf{A}_2 = \mathbf{Q}_2\,\mathbf{D}_2\,\mathbf{Q}_2^{-1}$. Then $\mathbf{A}$ can be represented as

$$\mathbf{A} = \begin{pmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{pmatrix} \left(\mathbf{D} + \mathbf{w}\,\mathbf{z}^{\mathrm{T}}\right) \begin{pmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{pmatrix},$$

where $\mathbf{D}$ is the diagonal matrix and $\mathbf{w}$, $\mathbf{z}$ are vectors defined by

$$\mathbf{D} = \mathrm{diag}(\mathbf{d}), \qquad \mathbf{w} = \begin{pmatrix} \mathbf{V}_1^{-1}\,\mathbf{x}_1 \\ \mathbf{0} \end{pmatrix}, \qquad \mathbf{z} = \begin{pmatrix} \mathbf{0} \\ \mathbf{V}_2^{\mathrm{T}}\,\mathbf{y}_2 \end{pmatrix}.$$

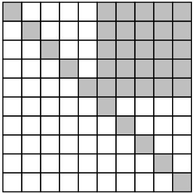
With the eigendecomposition $\mathbf{D} + \mathbf{w}\,\mathbf{z}^{\mathrm{T}} = \mathbf{U}\,\mathbf{D}\,\mathbf{U}^{-1}$, one obtains

$$\mathbf{A} = \begin{pmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{pmatrix} \mathbf{U}\,\mathbf{D}\,\mathbf{U}^{-1} \begin{pmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{pmatrix}^{-1}.$$

Discrete Polynomial Transforms – Semiseparable Matrices

$$
\mathbf{D} + \mathbf{w}\,\mathbf{z}^{\mathrm{T}} = \begin{pmatrix}
d_1 & 0 & \dots & 0 & v_1 w_{k+1} & v_1 w_{k+2} & \dots & v_1 w_n \\
0 & d_2 & \ddots & \vdots & v_2 w_{k+1} & v_2 w_{k+2} & & v_2 w_n \\
\vdots & \ddots & \ddots & 0 & \vdots & \vdots & \ddots & \vdots \\
\vdots & & \ddots & d_k & v_k w_{k+1} & v_k w_{k+2} & \dots & v_k w_n \\
\vdots & & & \ddots & d_{k+1} & 0 & \dots & 0 \\
\vdots & & & & \ddots & d_{k+2} & \ddots & \vdots \\
\vdots & & & & & \ddots & \ddots & 0 \\
0 & \dots & \dots & \dots & \dots & \dots & 0 & d_n
\end{pmatrix}
$$

$=$



Discrete Polynomial Transforms – Semiseparable Matrices

Theorem (K. 2008)

For $\mathbf{B} = \mathbf{D} + \mathbf{w}\,\mathbf{z}^{\mathrm{T}}$, we have

- $\mathbf{A} = \mathbf{U}\,\mathbf{D}\,\mathbf{U}^{-1}$ with the eigenvector matrix $\mathbf{U}$ and its inverse $\mathbf{U}^{-1}$ of the form

$$\mathbf{U} = \begin{pmatrix} \mathbf{I} & \mathbf{C} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \qquad \mathbf{U}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{C} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

  where $\mathbf{C} \in \mathbb{R}^{k \times (n-k)}$.

- The matrix $\mathbf{C}$ is defined by

$$\mathbf{C} = \left(\frac{w_i\, z_j}{d_i - d_j} \right)_{i=1,\, j=k+1}^{k,n}.$$

Discrete Polynomial Transforms – Semiseparable Matrices

Definition (Cauchy-like Matrix)

A matrix $\mathbf{C}$ of the form

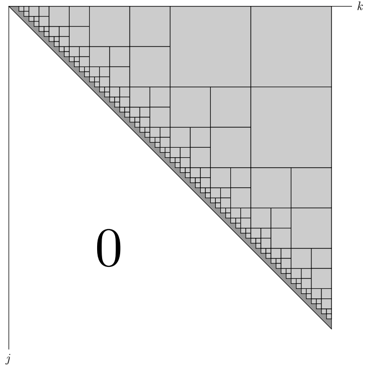
$$\mathbf{C} = \left(\frac{w_i \, z_j}{y_i - x_j} \right)$$

is called a *Cauchy-like matrix*.

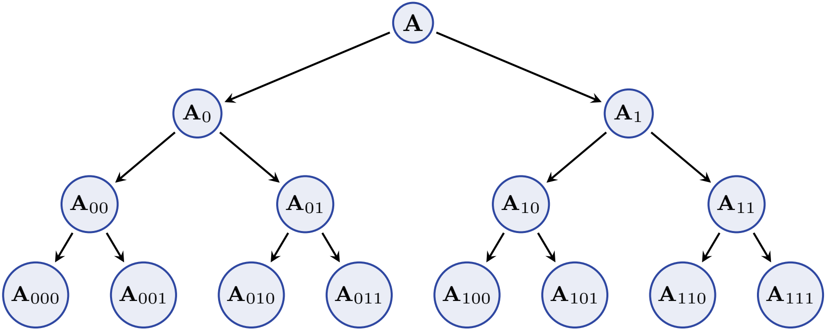
- The matrix $\mathbf{C} = \left(\frac{w_i \, z_j}{d_i - d_j} \right)_{i=1,\, j=k+1}^{k,n}$ is a **Cauchy-like** matrix.
- Matrix-vector multiplication normally takes $\mathcal{O}(k(n-k))$.
- **Fast multipole method (FMM)** exploits Cauchy-like structure to reduce the number of arithmetic operations.
- Fast summation takes $\mathcal{O}(n \log(1/\varepsilon))$ up to accuracy ε.
- Any level of accuracy ε possible.

Fast Multipole Method in a nutshell:

- Published by Rokhlin and Greengard in 1987 to speed up the calculation of long-ranged forces in the n-body problem.
- Divides matrix into smaller block.
- Uses low-rank approximations for each blocks.
- Clever organization leads to $\mathcal{O}(n \log(1/\varepsilon))$ algorithm.

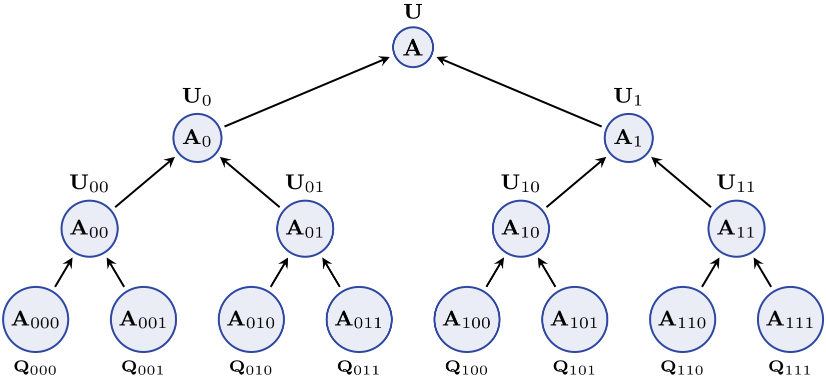


Precomputation: $\mathcal{O}(N \log N \log(1/\varepsilon)) - \mathcal{O}(N^2 \log N \log(1/\varepsilon))$



- Divide matrix recursively
- Compute smallest eigenvector matrices $\mathbf{Q}$ explicitly

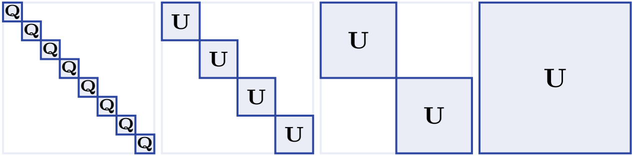
Precomputation: $\mathcal{O}(N \log N \log(1/\varepsilon)) - \mathcal{O}(N^2 \log N \log(1/\varepsilon))$



- Divide matrix recursively
- Compute smallest eigenvector matrices $\mathbf{Q}$ explicitly
- Solve rank-one modified eigenproblems
- Store data that determines each matrix $\mathbf{U}$

Discrete Polynomial Transforms – Semiseparable Matrices

The complete eigenvector matrix $\mathbf{Q}$ has the representation

$\mathbf{Q} =$ 

- Multiplication with each $\mathbf{Q}$ of **fixed size** $s \times s$ takes $\mathcal{O}(s^2)$
- The constant s is chosen beforehand
- Multiplication with each $\mathbf{U}$ of size $n \times n$ takes $\mathcal{O}(n \log(1/\varepsilon))$

In total: Multiplication with matrix $\mathbf{Q}$ takes $\mathcal{O}(N \log N \log(1/\varepsilon))$

Summary (same as for tridiagonal matrices):

- Time: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Memory: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Relatively stable (numerical experience) ✓
- Suited for all classical orthogonal polynomials
- Requirements: Three-term recurrence, differential equation, derivative identity

Discrete Polynomial Transforms – Algorithms

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Cascade Summation

(Driscoll, Healy, 1994; Potts, Steidl, Tasche, 1998;
Potts 2003; Keiner, Potts, 2006)

Tridiagonal Matrices

(Tygert, 2005)

Semiseparable Matrices

(K., 2007)

Direct Matrix Compression

(Rokhlin, 1991, K., 2007)

Discrete Polynomial Transforms – Matrix Compression

Want to evaluate

$$f(x) = \sum_{k=0}^{N} \hat{f}_k p_k(x).$$

Program (similar to Cascade Summation):

- Convert to Chebyshev expansion (node-independent)

$$f(x) = \sum_{k=0}^{N} \hat{g}_k T_k(x).$$

- Evaluate at target nodes x_j using an NFCT/NFFT.

More general view of first step:

Change of basis from p_k to q_k.

- Direct matrix compression uses the same algorithm that appears in the usual Fast Multipole Method to apply the matrix for the first step (p_k to q_k) efficiently
- For this, we need explicit expressions for the entries in the matrix $\mathbf{B} = (b_{j,k})_{j,k=0}^{N}$ that appears in

$$\hat{\mathbf{g}} = \mathbf{B}\,\hat{\mathbf{f}}.$$

Example: If p_k are the Legendre polynomials P_k and q_k are the Chebyshev polynomials of first kind T_k, we have

$$b_{j,k} = \int_{-1}^{1} T_j(x)\,P_k(x)\,\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x.$$

Discrete Polynomial Transforms – Matrix Compression

- Explicit expressions for the entries $b_{j,k}$ can be given for all classical orthogonal polynomials.
- Laguerre polynomials:

$$b_{j,k} = (-1)^{j+k} \frac{\Gamma(k-j+\alpha-\hat{\alpha})\Gamma(k+1)}{\Gamma(\alpha-\hat{\alpha})\Gamma(j+1)\Gamma(k-j+1)}, \qquad \text{with } j \leq k,$$

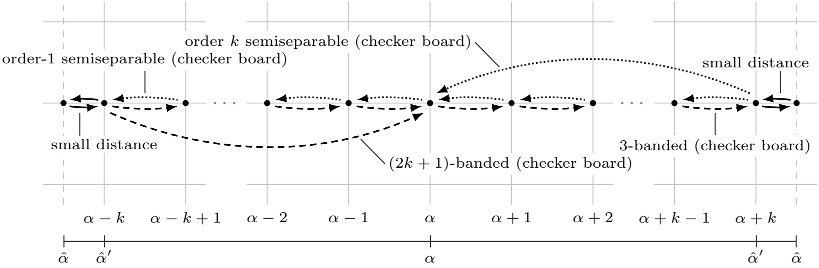
- Gegenbauer polynomials $C_k^{(\alpha)}$, $C_k^{(\beta)}$:

$$b_{j,k} = \frac{\Gamma(\beta)(j+\beta)\Gamma\left(\frac{k-j}{2}+\alpha-\beta\right)\Gamma\left(\frac{k+j}{2}+\alpha\right)}{\Gamma(\alpha)\Gamma(\alpha-\beta)\Gamma\left(\frac{k-j}{2}+1\right)\Gamma\left(\frac{k+j}{2}+\beta+1\right)}, \quad \text{with } j \leq k.$$

- Jacobi. . .

Example: Gegenbauer polynomials $C_k^{(\alpha)}$, $C_k^{(\beta)}$.
Fundamental obersavtions:

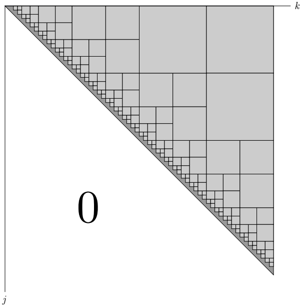
- If $|\alpha - \beta| \in \mathbb{N}$, then $\mathbf{B}$ is either banded or semiseparable.
- If $|\alpha - \beta| < 1$, then the entries $b_{j,k}$ of $\mathbf{B}$ are samples of a smooth function, allowing application of FMM.



To show "smoothness" of coefficients $b_{j,k}$ whenever $|\alpha - \beta| < 1$, we interpret $b_{j,k}$ as samples of a function $\mathcal{B}(x,y)$

$$\mathcal{B}(x,y) = \frac{\Gamma(\beta)(x+\beta)\Gamma\left(\frac{x-y}{2}+\alpha-\beta\right)\Gamma\left(\frac{x+y}{2}+\alpha\right)}{\Gamma(\alpha)\Gamma(\alpha-\beta)\Gamma\left(\frac{x-y}{2}+1\right)\Gamma\left(\frac{x+y}{2}+\beta+1\right)}.$$

Need to show that this function can be well-approximated on each square separated from the diagonal.



Formal definition for well-separated squares:

Definition

A square $S \subset \mathbb{R} \times \mathbb{R}$ defined by the formula $S = [x_0, x_0 + c] \times [y_0, y_0 + c]$ with $c > 0$ is said to be *well-separated* if $y_0 - x_0 \geq 2c$.

Theorem (K. 2008)

Let $S = [x_0, x_0 + c] \times [y_0, y_0 + c]$ with $c > 0$ be a well-separated square, $(x, y) \in S$, and $|\alpha - \beta| < 1$. Then

$$\|\mathcal{B}(\,\cdot\,, y) - \mathcal{B}_n(\,\cdot\,, y)\|_\infty = \mathcal{O}\big(\big(3 + \sqrt{8}\big)^{-n}\big),$$
$$\|\mathcal{B}(x, \,\cdot\,) - \mathcal{B}_n(x, \,\cdot\,)\|_\infty = \mathcal{O}\big(\big(3 + \sqrt{8}\big)^{-n}\big),$$

where $\mathcal{B}_n$ is the degree-n Chebyshev approximation to $\mathcal{B}$.

Improves and generalizes a previous result (Alpert, Rokhlin, 1991).

Summary:

- Time: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Memory: $\mathcal{O}((N \log N + M) \log(1/\varepsilon))$ ✓
- Relatively stable (numerical experience) ✓
- Suited for all classical orthogonal polynomials
- Requirements: Connection coefficients

Discrete Polynomial Transforms – Algorithms

Clenshaw Algorithm

(Clenshaw 1955, Smith 1965)

Cascade Summation

(Driscoll, Healy, 1994; Potts, Steidl, Tasche, 1998;
Potts 2003; Keiner, Potts, 2006)

Tridiagonal Matrices

(Tygert, 2005)

Semiseparable Matrices

(K., 2007)

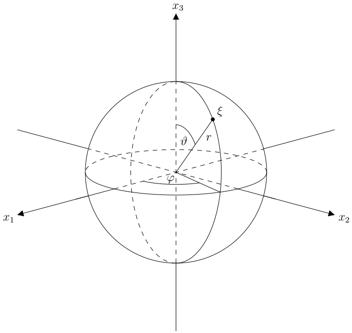
Direct Matrix Compression

(Rokhlin, 1991, K., 2007)

Sphere

Unit sphere in $\mathbb{R}^3$

$$\mathbb{S}^2 := \left\{ \mathbf{x} \in \mathbb{R}^3 : \ \|\mathbf{x}\|_2 = 1 \right\}$$
$$= \left\{ (\sin\varphi \sin\vartheta, \cos\varphi \sin\vartheta, \cos\vartheta)^\top : \vartheta \in [0,\pi], \varphi \in [-\pi,\pi) \right\}$$



Polynomials $f(\mathbf{x}) = \sum_{\mathbf{k} \in J_N} \hat{f}_{\mathbf{k}} Y_{\mathbf{k}}(\mathbf{x}) = \sum_{k=0}^{N} \sum_{n=-k}^{k} \hat{f}_k^n Y_k^n(\vartheta, \varphi)$

Spherical harmonics of degree k and order n

$$Y_{\mathbf{k}}(\mathbf{x}) = Y_k^n(\vartheta, \varphi) := \sqrt{\frac{2k+1}{4\pi}} P_k^{|n|}(\cos\vartheta) \mathrm{e}^{\mathrm{i}n\varphi}$$

Orthonormal basis of $L^2(\mathbb{S}^2)$

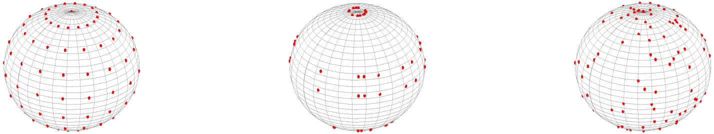
$$\begin{aligned} \delta_{k,l}\delta_{n,m} &= \int_{\mathbb{S}^2} Y_k^n(\mathbf{x}) \overline{Y_l^m(\mathbf{x})} \, \mathrm{d}\mu(\mathbf{x}) \\ &= \int_0^{2\pi} \int_0^{\pi} Y_k^n(\vartheta,\, \varphi) \overline{Y_l^m(\vartheta,\, \varphi)} \sin\vartheta \, \mathrm{d}\vartheta \mathrm{d}\varphi \end{aligned}$$

Addition theorem

$$\sum_{n=-k}^{k} \overline{Y_k^n(\mathbf{y})} Y_k^n(\mathbf{x}) = \frac{2k+1}{4\pi} P_k(\mathbf{y} \cdot \mathbf{x})$$

Discrete Fourier Transform on the Sphere

Sampling set $(\mathbf{x}_j)_{j=0,\dots,M-1} = \mathcal{X} \subset \mathbb{S}^2$



Nonequispaced Fourier matrix on the sphere

$$\mathbf{Y} = \left(Y_{\mathbf{k}}\left(\mathbf{x}_j\right)\right)_{j=0,\dots,M-1;\mathbf{k}\in J_N} \in \mathbb{C}^{M\times(N+1)^2}$$

$\hat{\mathbf{f}} \in \mathbb{C}^{(N+1)^2}$ given, compute

$$\mathbf{f} = \mathbf{Y}\hat{\mathbf{f}}, \qquad f_j = f\left(\mathbf{x}_j\right) = \sum_{\mathbf{k}\in J_N} \hat{f}_{\mathbf{k}} Y_{\mathbf{k}}\left(\mathbf{x}_j\right), \quad j = 0, \dots, M-1$$

Algorithm

1. Compute coefficients $\hat{g}_{k,n}$ in

$$f(\vartheta, \varphi) = \sum_{n=-N}^{N} \sum_{k=|n|}^{N} \hat{f}_k^n \sqrt{\frac{2k+1}{4\pi}} P_k^{|n|}(\cos\vartheta) \mathrm{e}^{\mathrm{i}n\varphi}$$

$$= \sum_{n=-N}^{N} \sum_{k=-N}^{N} \hat{g}_{k,n} \mathrm{e}^{\mathrm{i}k\vartheta} \mathrm{e}^{\mathrm{i}n\varphi}$$

2. Apply the $2d$-NFFT

Discrete Fourier Transform on the Sphere

Three-term recurrence relation

$$P_{k+1}^{n}(x) = (\alpha_k^n x + \beta_k^n)\, P_k^n(x) + \gamma_k^n P_{k-1}^n(x),$$

Multiple applications of the three-term recurrence relation

$$P_{k+c}^{n}(x) = P_c^{n,[k]}(x) P_k^n(x) + c_k^n P_{c-1}^{n,[k-1]}(x) P_{k-1}^n(x)$$

FPT - Fast polynomial transform for fixed (even) n

$$\sum_{k=|n|}^{N} \hat{f}_k^n P_k^{|n|} = \sum_{k=0}^{N} \tilde{g}_{k,n} T_k$$

takes $\mathcal{O}(N \log^2 N)$ flops

Discrete Fourier Transform on the Sphere

NFFT on the sphere [Driscoll, Healy, Rockmore 1994-; Potts, Steidl, Tasche 1998; Mohlenkamp 1999; Suda, Takami 2001; Rokhlin, Tygert 2004; K., Potts, Kunis 2002-]

$$\mathcal{O}\left(N^2 \log^2 N + |\log \varepsilon|^2 M\right)$$

NDFT on the sphere takes $\mathcal{O}\left(MN^2\right)$ (e.g. $N = 1000$, $M = N^2$)

$$\approx 1.7\text{min} \quad \text{vs.} \quad \approx 1.1\text{d}$$

Discrete Fourier Transform on the Sphere – Gaussians

Compute for $\sigma > 0$, $\mathbf{x}_j\,, \mathbf{y}_l \in \mathbb{S}^2$ the sums

$$g\left(\mathbf{x}_j\right) = \sum_{l=0}^{L-1} \alpha_l \, \mathrm{e}^{-\sigma \|\mathbf{x}_j - \mathbf{y}_l\|_2^2}$$

Truncated Fourier-Legendre expansion

$$\mathrm{e}^{-\sigma \|\mathbf{x}_j - \mathbf{y}_l\|_2^2} = \mathrm{e}^{2\sigma(\mathbf{x}_j \cdot \mathbf{y}_l - 1)} \approx \sum_{k=0}^{N} \hat{w}_k P_k\left(\mathbf{x}_j \cdot \mathbf{y}_l\right)$$

where

$$\begin{aligned} \hat{w}_k &= \frac{1}{2\pi} \int_{-1}^{1} \mathrm{e}^{2\sigma(x-1)} P_k\left(x\right) \, \mathrm{d}x \\ &= 2\sigma^{-\frac{1}{2}} \mathrm{e}^{-2\sigma} \pi^{\frac{3}{2}} I_{k+\frac{1}{2}}\left(2\sigma\right) \end{aligned}$$

Discrete Fourier Transform on the Sphere – Gaussians

Using the addition theorem for spherical harmonics

$$g_N(\mathbf{x}_j) = \sum_{l=0}^{L-1} \alpha_l \sum_{k=0}^{N} \hat{w}_k P_k(\mathbf{y}_l \cdot \mathbf{x}_j)$$

$$= \underbrace{\sum_{k=0}^{N} \sum_{n=-k}^{k} \hat{w}_k \underbrace{\left(\sum_{l=0}^{L-1} \alpha_l \overline{Y_k^n(\mathbf{y}_l)} \right)}_{\text{adjoint NFSFT}} Y_k^n(\mathbf{x}_j)}_{\text{NFSFT}}$$

Approximation error

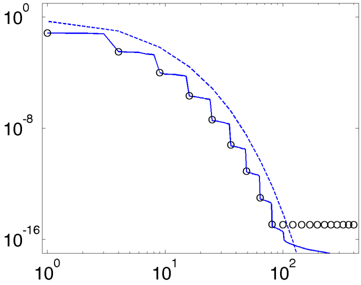
$$\frac{\|g - g_N\|_\infty}{\|\alpha\|_1} \leq \frac{\sqrt{\pi\sigma}\,(\mathrm{e}^\sigma - 1)\,\sigma^{N-\frac{1}{2}}}{\Gamma\left(N + \frac{1}{2}\right)}$$

Total number of floating point operations

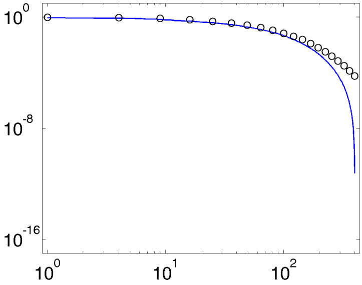
$$\mathcal{O}\left(N^2 \log^2 N + |\log \varepsilon_{\text{NFFT}}|\,(M + L)\right) \text{ vs. } \mathcal{O}(LM)$$

Discrete Fourier Transform on the Sphere – Gaussians

Comparison to truncated SVD, $L = M = 400$ pseudo random nodes



$\sigma = 0.1$



$\sigma = 10$

TSVD (blue), Approximation ($\circ$), and Error estimate (blue, dashed)

Discrete Fourier Transform on the Sphere – Gaussians

Computation time

$L = M$	direct alg.	w/pre-comp.	FS, NFSFT	error E_∞
2^6	0.00001 s	0.00008 s	0.62 s	$7.7 \cdot 10^{-14}$
2^8	0.00025 s	0.0014 s	0.62 s	$4.1 \cdot 10^{-14}$
2^{10}	0.04 s	0.021 s	0.65 s	$3.6 \cdot 10^{-14}$
2^{12}	6.4 s	0.35 s	0.72 s	$1.3 \cdot 10^{-14}$
2^{14}	1.6 min	*5.6 s	1.0 s	$5.5 \cdot 10^{-15}$
2^{16}	27.6 min	*1.5 min	2.3 s	$2.9 \cdot 10^{-15}$
2^{18}	7.2 h	*23.3 min	7.5 s	$1.9 \cdot 10^{-15}$
2^{20}	*4.8 d	*6.4 h	28 s	–
2^{21}	*19.7 d	*1.0 d	55 s	–

* = estimated

Program

Part I – Fourier Analysis and the FFT
Stefan, Monday, 14:15 – 16:00, Room U322

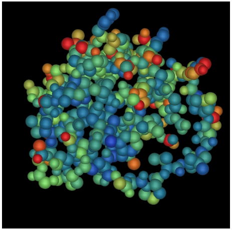
Part II – Orthogonal Polynomials
Jens, Tuesday, 12:15 – 14:00, Room U141 (Lecture Hall F)
Practice Session: 14:30 – 16:00, Room Y339b (Basics and Matlab Hands-On)

Part III – Fast Polynomial Transforms and Applications
Jens, Wednesday, 12:15 – 14:00, Room U345
Practice Session: 14:30 – 16:00, Room Y338c (C Library Hands-On)

Part IV – Fourier Transforms on the Rotation Group
Antje, Thursday, 14:15 – 16:00, Room U322

Part V – High Dimensions and Reconstruction
Stefan, Friday, 10:15 – 12:00, Room U322

Part IV – Fourier Transforms on the Rotation Group



What this talk is about...

1. The Rotation Group
2. Fourier Analysis on the Rotation Group
3. Algorithms for $\mathrm{SO}(3)$ Fourier Transforms
4. Applications

...and why do we care?

Considering

1. the Rotation Group

may lead to

2. a template for Fourier analysis on other groups
   - locally compact groups in general
   - $SU(2)$
   - $SE(3)$ (the motion group)

whereas fast

3. Algorithms for Fourier transforms on the Rotation Group

are useful in various

4. applications
   - motion estimation
   - texture analysis
   - Protein-Protein-Docking

1 The Rotation Group

2 Fourier Analysis on the Rotation Group

3 Algorithms for $\mathrm{SO}(3)$ Fourier Transforms

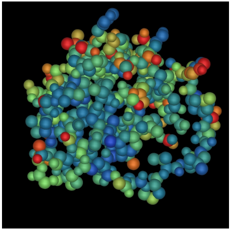
4 Applications

- based on:
  Gregory S. Chirikjian, Alexander B. Kyatkin,
  Engineering Applications of Noncommutative Harmonic Analysis with Emphasis on Rotation and Motion Groups

What is a rotation?

- ... a movement of a rigid body that keeps any given point of that body at a constant distance from a fixed line.





- ... a linear transformation that preserves angles, lengths and orientations of vectors.

Representations of Rotations:

A rotation is a linear transformation

$\downarrow$

consider $\mathbf{R} \in \mathbb{R}^{3\times 3}$ as a rotation

Representations of Rotations:

A rotation is a linear transformation that preserves angles,

consider inner product: $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$ and $\mathbf{R} \in \mathbb{R}^{3\times 3}$:

$$\begin{aligned} \mathbf{v} \cdot \mathbf{w} &= \mathbf{R}\mathbf{v} \cdot \mathbf{R}\mathbf{w} \\ \Leftrightarrow \mathbf{v}^T\mathbf{w} &= (\mathbf{R}\mathbf{v})^T\mathbf{R}\mathbf{w} \\ \Leftrightarrow \mathbf{v}^T\mathbf{w} &= \mathbf{v}^T \underbrace{\mathbf{R}^T\mathbf{R}}_{\mathbf{I_3}} \mathbf{w} \end{aligned}$$

Representations of Rotations:

A rotation is a linear transformation that preserves angles,

$\Big\downarrow$ $\mathbf{R} \in \mathbb{R}^{3\times 3}$ is a rotation

consider inner product: $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$ and $\mathbf{R} \in \mathbb{R}^{3\times 3}$:

$\Big\downarrow$ angle preservation

$\mathbf{R} \in \mathbb{R}^{3\times 3}$ is orthogonal

$$
\begin{aligned}
\mathbf{v} \cdot \mathbf{w} &= \mathbf{R}\mathbf{v} \cdot \mathbf{R}\mathbf{w} \\
\Leftrightarrow \mathbf{v}^T\mathbf{w} &= (\mathbf{R}\mathbf{v})^T\mathbf{R}\mathbf{w} \\
\Leftrightarrow \mathbf{v}^T\mathbf{w} &= \mathbf{v}^T \underbrace{\mathbf{R}^T\mathbf{R}}_{\mathbf{I_3}} \mathbf{w}
\end{aligned}
$$

A rotation is a linear transformation that preserves angles, lengths

$\Big\downarrow$ already done

$$\boxed{\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}\cdot\mathbf{v}}}$$

Representations of Rotations:

A rotation is a linear transformation that preserves angles, lengths and orientations of vectors.

↓

orientation preserving if det$(\mathbf{R}) > 0$

↓

orthogonal matrices have determinant either -1 or 1

Representations of Rotations: Rotation matrices

A rotation is a linear transformation that preserves angles, lengths and orientations of vectors.

$$\Downarrow$$

Rotations can be represented by orthogonal 3×3 - matrices having determinant 1.

What's in a name: $\mathrm{SO}(3)$

The set of all rotation matrices

$$\{\mathbf{R} \in \mathbb{R}^{3\times 3} \,|\, \mathsf{det}(\mathbf{R}) = 1 \,\wedge\, \mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{I_3}\}$$

constitutes the group of **S**pecial **O**rthogonal transformations in $\mathbb{R}^{\mathbf{3}}$:

$\mathrm{SO}(3)$, the rotation group

What's in a name: $\mathrm{SO}(3)$

The set of all rotation matrices

$$\{\mathbf{R} \in \mathbb{R}^{3\times 3} \,|\, \mathsf{det}(\mathbf{R}) = 1 \,\wedge\, \mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{I_3}\}$$

constitutes the group of **S**pecial **O**rthogonal transformations in $\mathbb{R}^{\mathbf{3}}$:

$\mathrm{SO}(3)$, the rotation group

- group operation is composition
- $\mathbf{I_3}$ is the identity element
- $\mathbf{R}^T$ is the inverse element (or back rotation) of $\mathbf{R}$
- non-abelian

Representations: Axis - Angle

We also defined a rotation to be:

> ... a movement of a rigid body that keeps any given point of that body at a constant distance from a fixed line.

A more natural describtion of a rotation:

1. Where does the rotation take place? (Where is the fixed line?)
   $\longrightarrow$ rotation axis $\mathbf{r} \in \mathbb{R}^3$
2. How much do we rotate?
   $\longrightarrow$ rotation angle $\omega \in [0, \pi]$ (absolute value of a rotation)

$$\mathbf{R} \longrightarrow \mathbf{R_r}(\omega)$$

- The eigenvalues of all rotation matrices are given by $\lambda_1 = 1$ and $\lambda_{2,3} = e^{\pm i\omega}$ where $0 \leq \omega \leq \pi$.

Connection between Rotation Matrices and Axis & Angle

$$\mathbf{R} \longrightarrow \mathbf{R}_{\mathbf{r}}(\omega)$$

- The eigenvalues of all rotation matrices are given by $\lambda_1 = 1$ and $\lambda_{2,3} = e^{\pm i\omega}$ where $0 \leq \omega \leq \pi$.
- ω defines the angle of rotation ($=$ the absolute value of rotation)
- it is uniquely determined by $\cos\omega = \frac{1}{2}\mathrm{trace}(\mathbf{R}) - 1$.

Connection between Rotation Matrices and Axis & Angle

$$\mathbf{R} \longrightarrow \mathbf{R}_{\mathbf{r}}(\omega)$$

- the rotation axis $\mathbf{r}$ is defined to be the normalized eigenvector corresponding to the eigenvalue $\lambda = 1$ of the rotation matrix.

Connection between Rotation Matrices and Axis & Angle

$$\mathbf{R} \longrightarrow \mathbf{R}_{\mathbf{r}}(\omega)$$

- the rotation axis $\mathbf{r}$ is defined to be the normalized eigenvector corresponding to the eigenvalue $\lambda = 1$ of the rotation matrix.

Note: If we have $\omega = 0$ then $\mathbf{R}_{\mathbf{r}}(0) = \mathbf{I}_3$. In that case $\mathbf{R}$ has a threefold eigenvalue $\lambda = 1$. Therefore we can not determine the axis of rotation. It can be any vector $\mathbf{r} \in \mathbb{R}^3$.

Two important examples

A rotation about the z-axis reads as

$$\mathbf{R}_z(\omega) = \begin{pmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Analogously the matrices for rotations about the y-axis is

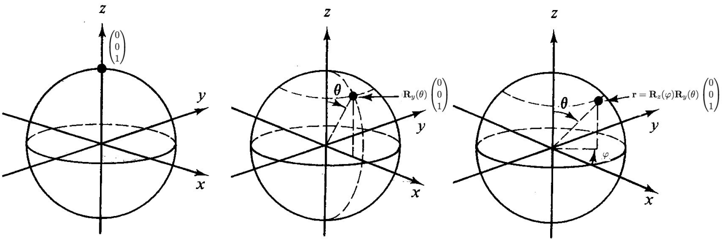
$$\mathbf{R}_y(\omega) = \begin{pmatrix} \cos(\omega) & 0 & \sin(\omega) \\ 0 & 1 & 0 \\ -\sin(\omega) & 0 & \cos(\omega) \end{pmatrix}.$$

Deducing the rotation matrix from axis & angle

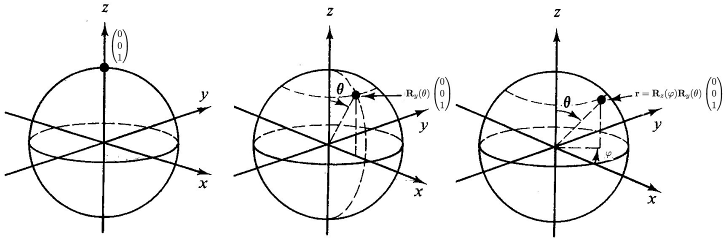
- Any point $\mathbf{r} = (\varphi, \theta)$ on the sphere can be reached by rotating the unit vector of the z-axis

$$\mathbf{r} = \mathbf{R}_z(\varphi)\mathbf{R}_y(\theta) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

- Consequently the back rotation $\mathbf{R}_y^T(\theta)\mathbf{R}_z^T(\varphi)$ rotates $\mathbf{r}$ onto the z-axis.



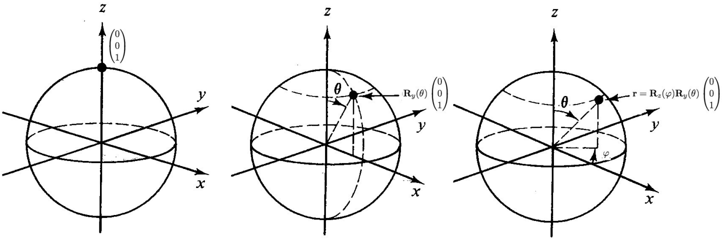
Deducing the rotation matrix from axis & angle



Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R_r}(\omega) \in \mathrm{SO}(3)$,

$$\mathbf{R_r}(\omega) \quad =$$

Deducing the rotation matrix from axis & angle

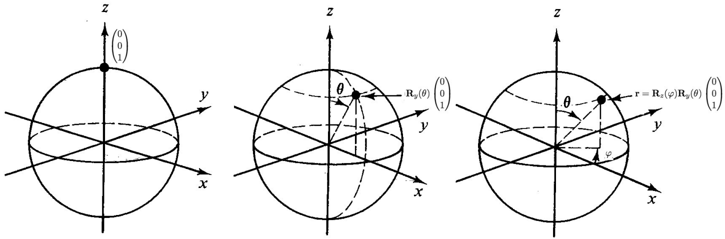


Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R_r}(\omega) \in \mathrm{SO}(3)$,

$$\mathbf{R_r}(\omega) \quad = \quad \underbrace{\mathbf{R}_y^T(\theta)\mathbf{R}_z^T(\varphi)}$$

- rotate axis $\mathbf{r}$ on z-axis

Deducing the rotation matrix from axis & angle

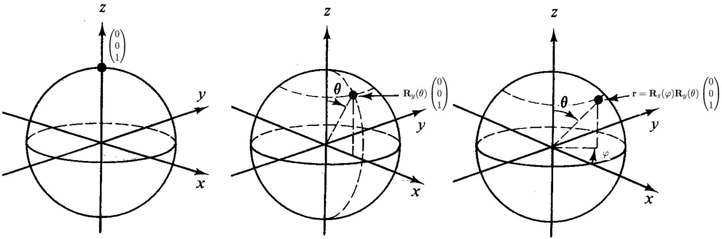


Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R}_\mathbf{r}(\omega) \in \mathrm{SO}(3)$,

$$\mathbf{R}_\mathbf{r}(\omega) \quad = \qquad\qquad \mathbf{R}_z(\omega)\,\underbrace{\mathbf{R}_y^T(\theta)\mathbf{R}_z^T(\varphi)}$$

- does a rotation about the actual rotation angle

Deducing the rotation matrix from axis & angle

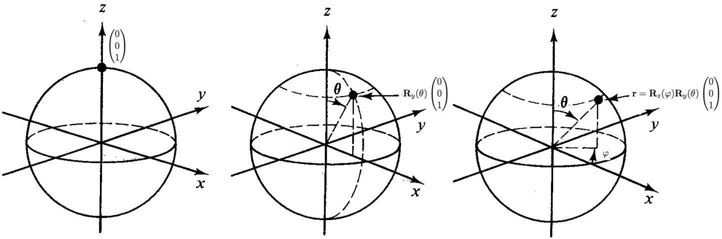


Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R}_\mathbf{r}(\omega) \in \mathrm{SO}(3)$,

$$\mathbf{R}_\mathbf{r}(\omega) \quad = \quad \underbrace{\mathbf{R}_z(\varphi)\mathbf{R}_y(\theta)}\,\mathbf{R}_z(\omega)\,\underbrace{\mathbf{R}_y^T(\theta)\mathbf{R}_z^T(\varphi)}$$

- rotate z-axis back onto the actual rotation axis

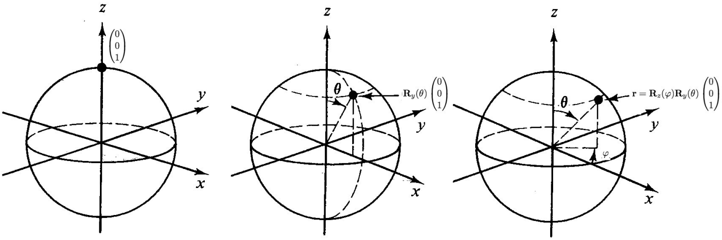
Deducing the rotation matrix from axis & angle



Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R_r}(\omega) \in \mathrm{SO}(3)$,

$$\begin{aligned}\mathbf{R_r}(\omega) &= \underbrace{\mathbf{R}_z(\varphi)\mathbf{R}_y(\theta)}\,\mathbf{R}_z(\omega)\,\underbrace{\mathbf{R}_y^T(\theta)\mathbf{R}_z^T(\varphi)} \\ &= \mathbf{F}\mathbf{R}_z(\omega)\mathbf{F}^{-1} \qquad \text{similarity transform}\end{aligned}$$

Deducing the rotation matrix from axis & angle



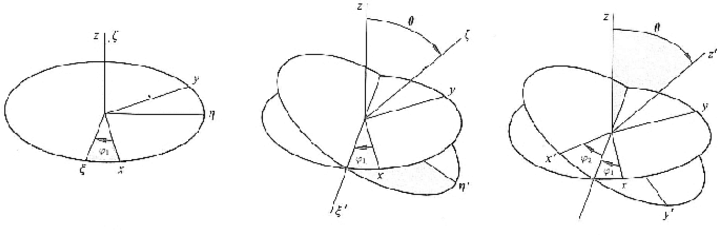
Let $\mathbf{r} = (\varphi, \theta)$ be the rotation axis and ω the rotation angle of $\mathbf{R}_{\mathbf{r}}(\omega) \in \mathrm{SO}(3)$,

$$\mathbf{R}_{\mathbf{r}}(\omega) =$$

$$\mathbf{R}_{\mathbf{r}}(\omega) = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$$

Euler Angle Representation

We find that arbitrary rotations can be described by three angles α, β and γ, describing three consecutive rotations about orthogonal axes in $\mathbb{R}^3$



- composition of three rotations with fixed axes
- axes of two consecutive rotations are orthogonal

A common convention for Euler Angles

Given three angles $\alpha, \gamma \in [0, 2\pi)$ and $\beta \in [0, \pi]$, the corresponding rotation matrix $\mathbf{R}$ is given by

$$\mathbf{R} = \mathbf{R}_{zyz}(\alpha, \beta, \gamma) \quad = \quad \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$$

or

$$\mathbf{R} = \begin{pmatrix} \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\gamma\sin\alpha - \cos\alpha\cos\beta\sin\gamma & \cos\alpha\sin\beta \\ \cos\beta\cos\gamma\sin\alpha + \cos\alpha\sin\gamma & \cos\alpha\cos\gamma - \cos\beta\sin\alpha\sin\gamma & \sin\alpha\sin\beta \\ -\cos\gamma\sin\beta & \sin\beta\sin\gamma & \cos\beta \end{pmatrix}$$

Linking rotations and the sphere

There is a connection between the sphere $\mathbb{S}^2$ and the rotation group $\mathrm{SO}(3)$.

We already learned that any element on the sphere can be represented as

$$\mathbb{S}^2 = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Looking at the Euler angle representation we find the same pair of rotations

Linking rotations and the sphere

There is a connection between the sphere $\mathbb{S}^2$ and the rotation group $\mathrm{SO}(3)$.
We already learned that any element on the sphere can be represented as

$$\mathbb{S}^2 = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Looking at the Euler angle representation we find the same pair of rotations

$$\mathbf{R} \quad = \quad \underbrace{\mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)}_{\mathbb{S}^2-\text{part}}\mathbf{R}_z(\gamma)$$

Linking rotations and the sphere

There is a connection between the sphere $\mathbb{S}^2$ and the rotation group $\mathrm{SO}(3)$.
We already learned that any element on the sphere can be represented as

$$\mathbb{S}^2 = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Looking at the Euler angle representation we find the same pair of rotations

$$\mathbf{R} \quad = \quad \underbrace{\mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)}_{\mathbb{S}^2-\text{part}} \underbrace{\mathbf{R}_z(\gamma)}_{\mathbb{S}^1-\text{part}}$$

The rotation $\mathbf{R}_z(\gamma)$ can be though of a one-dimensional rotation, i.e, we can represent every point on the unit circle $\mathbb{S}^1$ with it.

Linking rotations and the sphere

Indeed, we have:

$$\boxed{\mathrm{SO}(3): \quad \mathbb{S}^2 \times \mathbb{S}^1}$$

$$\downarrow$$

$$\boxed{\mathrm{SO}(3) \text{ generalizes the sphere } \mathbb{S}^2}$$

$$\downarrow$$

Can we generalize the concept of $\mathbb{S}^2$ Fourier transforms to the rotation group as well?

1 The Rotation Group

2 Fourier Analysis on the Rotation Group

3 Algorithms for $\mathrm{SO}(3)$ Fourier Transforms

4 Applications

based on:

- Gregory S. Chirikjian, Alexander B. Kyatkin,
  Engineering Applications of Noncommutative Harmonic Analysis with Emphasis on Rotation and Motion Groups
- Naum J. Vilenkin,
  Special Functions and the Theory of Group Representations

Fourier Analysis on Groups : the ingredients

Given a function $f \in L^2(G)$ where G is a locally compact group and $g \in G$

1. We need to know an integration invariant measure μ

$$\int_G |f(g)|^2 \, \mathrm{d}\mu(f(g)) < \infty$$

Fourier Analysis on Groups : the ingredients

Given a function $f \in L^2(G)$ where G is a locally compact group and $g \in G$

1. We need to know an integration invariant measure μ
2. We define the Fourier transform

$$\hat{f}(l) = \int_G f(g) U(g^{-1}, l) \mathrm{d}g$$

where $U(\cdot, l)$ is some unitary matrix function with index l

- unitary matrix: $\overline{\mathbf{U}}^T \mathbf{U} = \mathbf{I}$
- satisfies $\mathbf{U}(gh) = \mathbf{U}(g)\mathbf{U}(h)$ where $g, h \in G$

Fourier Analysis on Groups : the ingredients

Given a function $f \in L^2(G)$ where G is a locally compact group and $g \in G$

1. We need to know an integration invariant measure μ
2. We define the Fourier transform
3. and the inverse Fourier transform

$$f(g) = \int_{\hat{G}} \mathrm{trace}(\hat{f}(l)U(g,l))\mathrm{d}\nu(l)$$

where $\hat{G}$ is the space of all values l and ν an appropriate integration measure on $\hat{G}$

Fourier Analysis on $\mathrm{SO}(3)$

We consider an element $\mathbf{R} \in \mathrm{SO}(3)$ to be parameterized in Euler angles and functions $f \in L^2(\mathrm{SO}(3))$:

$$f(\mathbf{R}) = f(\mathbf{R}(\alpha, \beta, \gamma)) = f(\alpha, \beta, \gamma)$$

Fourier Analysis on $\mathrm{SO}(3)$

We consider an element $\mathbf{R} \in \mathrm{SO}(3)$ to be parameterized in Euler angles and functions $f \in L^2(\mathrm{SO}(3))$:

$$f(\mathbf{R}) = f(\mathbf{R}(\alpha, \beta, \gamma)) = f(\alpha, \beta, \gamma)$$

1. an invariant integration measure:

$$\mathrm{d}\mathbf{R} = \sin\beta \, \mathrm{d}\alpha \, \mathrm{d}\beta \, \mathrm{d}\gamma$$

   arises from the coordinate transform to Euler angles as

$$\int_{\mathrm{SO}(3)} \mathrm{d}\mathbf{R} = \int_0^{2\pi} \int_0^{\pi} \int_0^{2\pi} \sin\beta \, \mathrm{d}\alpha \, \mathrm{d}\beta \, \mathrm{d}\gamma$$

2. We define the Fourier transform :

$$\hat{f}(l) = \int_G f(g) U(g^{-1}, l) \mathrm{d}g$$

where $U(\cdot, l)$ is some unitary matrix function with index l

Fourier Analysis on $\mathrm{SO}(3)$

2. We define the Fourier transform componentwise:

$$\hat{f}_{m,n}(l) = \int_{\mathrm{SO}(3)} f(\mathbf{R}) U_{m,n}(\mathbf{R}^{-1}, l) \mathrm{d}\mathbf{R}$$

where $U_{m,n}(\cdot, l)$ is the (m, n)th element of the unitary matrix $U(\cdot, l)$ and $l \in \mathbb{N}_0$

Fourier Analysis on $\mathrm{SO}(3)$

2. We define the Fourier transform componentwise:

$$\hat{f}_{m,n}(l) = \int_{\mathrm{SO}(3)} f(\mathbf{R}) U_{m,n}(\mathbf{R}^{-1}, l) \mathrm{d}\mathbf{R}$$

where $U_{m,n}(\cdot, l)$ is the (m,n)th element of the unitary matrix $U(\cdot, l)$ and $l \in \mathbb{N}_0$

Theorem (Peter-Weyl-Theorem for $\mathrm{SO}(3)$)

Let $U_{m,n}(\cdot, l)$ be defined as above. Then

- the collection of functions $\{U_{m,n}(\cdot, l)\}$ for all $l \in \mathbb{N}_0$ forms a *complete orthogonal basis for* $L^2(\mathrm{SO}(3))$.
- $L^2(\mathrm{SO}(3))$ can be decomposed into orthogonal subspaces: $L^2(\mathrm{SO}(3)) = \bigoplus_{l \in \mathbb{N}_0} \mathrm{Harm}_l(\mathrm{SO}(3))$
- For each fixed l the functions $U_{m,n}(\cdot, l)$ form a basis of the subspace $\mathrm{Harm}_l(\mathrm{SO}(3))$

Theorem (Peter-Weyl-Theorem for $\mathrm{SO}(3)$)

- The collection of functions $\{U_{m,n}(\cdot, l)\}$ for all $l \in \mathbb{N}_0$ forms a *complete orthogonal basis for* $L^2(\mathrm{SO}(3))$.

$\downarrow$

Every function $f \in L^2(\mathrm{SO}(3))$ has a unique representation in terms of the basis functions $U_{m,n}(\cdot, l)$

$\downarrow$

3. we get the inverse Fourier transform:

$$f(g) = \int_{\hat{G}} \mathrm{trace}(\hat{f}(l) U(g, l)) \mathrm{d}\nu(l)$$

Theorem (Peter-Weyl-Theorem for $\mathrm{SO}(3)$)

- The collection of functions $\{U_{m,n}(\cdot, l)\}$ for all $l \in \mathbb{N}_0$ forms a *complete orthogonal basis for* $L^2(\mathrm{SO}(3))$.

$\downarrow$

Every function $f \in L^2(\mathrm{SO}(3))$ has a unique representation in terms of the basis functions $U_{m,n}(\cdot, l)$

$\downarrow$

3. we get the inverse Fourier transform:

$$f(g) = \sum_{l \in \mathbb{N}_0} \mathrm{trace}(\hat{f}(l) U(g, l))$$

How do the functions $U_{m,n}(\cdot, l)$ arise?

...they arise as eigenfunctions of the Laplacian

$$\text{PDE: } \Delta u = \lambda u$$

separation of variables

$$u(\alpha, \beta, \gamma) = u_1(\alpha) u_2(\gamma) u_3(\beta)$$

three ODEs:

$$\begin{array}{lll} u_1'' + m^2 u_1 = 0 & u_1(0) = u_1(2\pi) & u_1'(0) = u_1'(2\pi) \\ u_2'' + n^2 u_1 = 0 & u_2(0) = u_2(2\pi) & u_2'(0) = u_2'(2\pi) \end{array}$$

$$(\sin\beta u_3')' + \left(l(l+1)\sin\beta - \frac{n^2 - 2mn\cos\beta + m^2}{\sin\beta} \right) u_3 = 0$$

$$u_3(0) = u_3(\pi) \qquad u_3'(0) = u_3'(\pi)$$

How do the functions $U_{m,n}(\cdot, l)$ arise?

$$u_1'' + m^2 u_1 = 0 \quad u_1(0) = u_1(2\pi) \quad u_1'(0) = u_1'(2\pi)$$
$$u_2'' + n^2 u_1 = 0 \quad u_2(0) = u_2(2\pi) \quad u_2'(0) = u_2'(2\pi)$$

$$\Big\downarrow$$

$$\begin{aligned} u_1(\alpha) &= \mathrm{e}^{-\mathrm{i}m\alpha} \\ u_2(\gamma) &= \mathrm{e}^{-\mathrm{i}n\gamma} \end{aligned}$$

How do the functions $U_{m,n}(\cdot, l)$ arise?

$$(\sin\beta u_3')' + \left(l(l+1)\sin\beta - \frac{n^2 - 2mn\cos\beta + m^2}{\sin\beta}\right)u_3 \quad = \quad 0$$

set $x = \cos\beta$

$$\left(\frac{\mathrm{d}}{\mathrm{d}x}((1-x^2)\frac{\mathrm{d}}{\mathrm{d}x}) - \left(l(l+1) - \frac{n^2 - 2mnx + m^2}{1-x^2}\right)\right)u_3 \quad = \quad 0$$

- the solution to this ODE is an associated function
- from self-adjoint form $(\sigma w u_3')' + \lambda_l w u_3 = 0$ we get

$$\begin{aligned} \sigma(x) &= (1-x^2) \qquad w(x) = (1-x)^{|n-m|}(1+x)^{|n+m|} \\ \lambda_l &= l(l+1+|n-m|+|n+m|) \end{aligned}$$

A suitable polynomial for u_3

The solution of the ODE for u_3 is the so-called Wigner-d function.

$$d_l^{mn}(x) = \frac{(-1)^{l-m}}{2^l}\sqrt{\frac{(l+m)!}{(l-n)!(l+n)!(l-m)!}}\sqrt{\frac{(1-x)^{n-m}}{(1+x)^{m+n}}}\frac{\mathrm{d}^{l-m}}{\mathrm{d}x^{l-m}}\frac{(1+x)^{n+l}}{(1-x)^{n-l}}$$

where $l \in \mathbb{N}_0$, $|m|, |n| \leq l$.

Wigner-d functions d_l^{mn} are related to:

- Jacobi polynomials:

$$d_\ell^{m,n}(x) := v(m,n)(1-x)^{\frac{|n-m|}{2}}(1+x)^{\frac{|n+m|}{2}}P_{\ell-\max(|m|,|n|)}^{(|n-m|,|n+m|)}(x)$$

for some factor $v(m,n)$

Wigner-d functions as Classical Orthogonal Polynomials

Wigner-d functions d_l^{mn}

- are orthogonal on $[-1, 1]$
- $\sigma(x) = (1 - x^2)$,

  $w(x) = (1 - x)^{|n-m|}(1 + x)^{|n+m|}$,
  $\lambda_l = l(l + 1 + |n - m| + |n + m|)$, $-l \leq m, n \leq l$
- Differential equation

$$\left(\frac{\mathrm{d}}{\mathrm{d}x}((1 - x^2)\frac{\mathrm{d}}{\mathrm{d}x}) - \left(l(l + 1) - \frac{n^2 - 2mnx + m^2}{1 - x^2}\right)\right) d_l^{mn} = 0$$

- Rodrigues formula

$$d_l^{mn}(x) = \frac{(-1)^{l-m}}{2^l}\sqrt{\frac{(l + m)!}{(l - n)!(l + n)!(l - m)!}}\sqrt{\frac{(1 - x)^{n-m}}{(1 + x)^{m+n}}}\frac{\mathrm{d}^{l-m}}{\mathrm{d}x^{l-m}}\frac{(1 + x)^{n+l}}{(1 - x)^{n-l}}$$

- Three-term recurrence ugly

Putting the pieces together: Wigner-D functions

- We considered the eigenfunctions u of the Laplace operator on $\mathrm{SO}(3)$ by solving $\Delta u = \lambda u$.
- We found a separation of u in Euler angles $u = u_1(\alpha)u_2(\gamma)u_3(\beta)$ and gave explicit expression for each u_i

Putting the pieces together: Wigner-D functions

- We considered the eigenfunctions u of the Laplace operator on $\mathrm{SO}(3)$ by solving $\Delta u = \lambda u$.
- We found a separation of u in Euler angles $u = u_1(\alpha)u_2(\gamma)u_3(\beta)$ and gave explicit expression for each u_i
- From now on we will denote the functions u by $D_l^{mn} := u$ a Wigner-D function
- It is given for $|m|, |n| \leq l \in \mathbb{N}_0$ by

$$D_l^{mn}(\alpha, \beta, \gamma) = \underbrace{\mathrm{e}^{-\mathrm{i}m\alpha}}_{u_1} \underbrace{\mathrm{e}^{-\mathrm{i}n\gamma}}_{u_2} \underbrace{d_l^{mn}(\cos\beta)}_{u_3}$$

where d_l^{mn} is a so-called Wigner-d function.

The sphere $\mathbb{S}^2$ as a special case of $\mathrm{SO}(3)$

- associated Legendre functions P_l^n:

$$d_l^{0,-n}(x) = P_l^n(x) = \frac{1}{2^l l!}\sqrt{\frac{(l-n)!}{(l+n)!}}\sqrt{(1-x^2)^n}\frac{\mathrm{d}^{l+n}}{\mathrm{d}x^{l+n}}(x^2-1)^l$$

Wigner-d function $\longrightarrow$ generalized associated Legendre fct.

- spherical harmonics Y_l^m:

$$\begin{aligned} Y_l^m(\beta,\alpha) &= \sqrt{\frac{2l-1}{4\pi}}\mathrm{e}^{\mathrm{i}m\alpha}d_l^{0m}(\cos\beta) \\ &= (-1)^{\delta_{m|m|}}\sqrt{\frac{2l-1}{4\pi}}D_l^{0,-m}(\alpha,\beta,\cdot) \end{aligned}$$

Wigner-D function $\longrightarrow$ generalized spherical harmonic

Finally: An orthogonal basis for $L^2(\mathrm{SO}(3))$

By means of the Peter-Weyl-Theorem the spaces

$$\mathsf{Harm}_l(\mathrm{SO}(3)) = \mathrm{span}\,\{D_l^{mn} : \, m, n = -l, \ldots, l\}$$

spanned by the Wigner-D functions satisfy

$$L^2(\mathrm{SO}(3)) = \bigoplus_{l=0}^{\infty} \mathsf{Harm}_l(\mathrm{SO}(3)).$$

The collection of Wigner-D functions

$$\{D_l^{mn}(\mathbf{R}) : \, l \in \mathbb{N}_0, m, n = -l, \ldots, l\}$$

forms an orthogonal basis system in $\mathrm{L}^2(\mathrm{SO}(3))$.

Now we have all ingredients for Fourier transforms on $\mathrm{SO}(3)$

The $\mathrm{SO}(3)$ Fourier Transform

$$\boxed{\{D_l^{m,n} \mid l \in \mathbb{N}_0, m, n = -l, \dots, l\} \text{ forms a basis in } L^2(\mathrm{SO}(3))}$$

As a consequence a function $f \in L^2(\mathrm{SO}(3))$ has a unique series expansion in terms of the Wigner-D functions

$$f = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \widehat{f}_l^{m,n} D_l^{m,n}$$

with Fourier coefficients $\hat{f}_l^{m,n}$ given by the inner product

$$\begin{aligned} \widehat{f}_l^{m,n} &= \frac{2l+1}{8\pi^2} \int_0^{2\pi} \int_0^{\pi} \int_0^{2\pi} f(\alpha,\beta,\gamma) D_l^{m,n}(\alpha,\beta,\gamma) \sin\beta \,\mathrm{d}\alpha \mathrm{d}\beta \mathrm{d}\gamma \\ &= \frac{2l+1}{8\pi^2} \langle f, D_l^{m,n} \rangle_{L^2(\mathrm{SO}(3))}. \end{aligned}$$

1 The Rotation Group

2 Fourier Analysis on the Rotation Group

3 Algorithms for $\mathrm{SO}(3)$ Fourier Transforms

4 Applications

A space for the Discrete $\mathrm{SO}(3)$ Fourier Transform

Consider the space of polynomials of maximum degree $L \in \mathbb{N}_0$:

$$\mathbb{D}_L = \bigoplus_{l=0}^{L} \operatorname{span}\left\{ D_l^{m,n} \,|\, m,n = -l,\ldots,l \right\} \subset L^2(\mathrm{SO}(3)).$$

with

- the index set consisting of all admissible indices corresponding to polynomials in $\mathbb{D}_L$

$$\mathcal{J}_L = \{(l,m,n) \mid l = 0,\ldots,L; m,n = -l,\ldots,l\}$$

- and dimension

$$\dim(\mathbb{D}_L) = |\mathcal{J}_L| = \sum_{l=0}^{L} (2l+1)^2 = \frac{1}{6}(2L+1)(2L+2)(2L+3).$$

The Discrete $\mathrm{SO}(3)$ Fourier Transform

Input:

- sampling set on $\mathrm{SO}(3)$: $\mathcal{R}_Q = (\mathbf{R}_1, \ldots, \mathbf{R}_Q)$ with $\mathbf{R}_q \in \mathrm{SO}(3)$, is a finite sequence of **arbitrary** rotations
- Fourier coefficients $\hat{\mathbf{f}} = (\hat{f}_l^{m,n})_{(l,m,n)\in\mathcal{J}_L}$

Evaluation:

$$f(\mathbf{R}_q) = \sum_{l=0}^{L} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{m,n} D_l^{m,n}(\mathbf{R}_q), \quad q = 1, \ldots, Q,$$

$\boxed{\text{nonequispaced discrete } \mathrm{SO}(3) \text{ Fourier transform (NDSOFT)}}$

Output:

- a function $f \in \mathbb{D}_L$ evaluated at rotations $\mathbf{R}_1, \ldots, \mathbf{R}_Q$

The Discrete SO(3) Fourier Transform

$$f(\mathbf{R}_q) = \sum_{l=0}^{L} \sum_{k=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{m,n} D_l^{m,n}(\mathbf{R}_q), \quad q = 1, \dots, Q,$$

In matrix-vector notation, the NDSOFT reads

$$\boxed{\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q} \hat{\mathbf{f}}}$$

with

- $\mathbf{f} = (f(\mathbf{R}_q))_{q=1,\dots,Q}$, the function samples
- $\hat{\mathbf{f}} = (\hat{f}_l^{m,n})_{(l,m,n)\in\mathcal{J}_L}$, the $\mathrm{SO}(3)$ Fourier coefficients
- $\mathbf{D}_{\mathcal{R}_Q} = \left(D_l^{m,n}(\mathbf{R}_q)\right)_{q=1,\dots,Q;\,(l,m,n)\in\mathcal{J}_L}$ the nonequispaced $\mathrm{SO}(3)$ Fourier matrix

The Discrete $\mathrm{SO}(3)$ Fourier Transform reversed

- The NDSOFT reads

$$\boxed{\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q} \hat{\mathbf{f}}}$$

- in general $\mathbf{D}_{\mathcal{R}_Q}$ is not a square-matrix $\Rightarrow$ not invertible
- Instead: applying the **adjoint** transform

$$\mathbf{D}_{\mathcal{R}_Q}^{\mathrm{H}} : \mathbb{C}^{Q} \to \mathbb{C}^{\mathcal{J}_L}$$

  is called adjoint NDSOFT:

$$\boxed{\tilde{\mathbf{f}} = \mathbf{D}_{\mathcal{R}_Q}^{\mathrm{H}} \mathbf{f}}$$

- For $\mathbf{W}_{\mathcal{R}_Q} = \mathrm{diag}(w_q)_{q=1,\ldots,Q}$ with suitable weights w_q,

$$\boxed{\hat{\mathbf{f}} = \mathbf{D}_{\mathcal{R}_Q}^{\mathrm{H}} \mathbf{W}_{\mathcal{R}_Q} \mathbf{f}}$$

  can be interpreted as a quadrature rule for the calculation of the Fourier coefficients of f, yields the **(pseudo) inverse** NDSOFT

A note on complexity

$$\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q} \hat{\mathbf{f}}$$

- lower bound: $\mathcal{O}(L^3)$ Fourier coefficients and $\mathcal{O}(Q)$ rotations as input values
  $\Rightarrow \mathcal{O}(L^3 + Q)$ flops
- naive approach: matrix-multiplication with $\mathbf{D}_{\mathcal{R}_Q} \in \mathbb{C}^{Q \times |\mathcal{J}_L|}$
  $\Rightarrow \mathcal{O}(L^3 Q)$ flops
- our approach on nonequispaced grids on the $\mathrm{SO}(3)$:

A note on complexity

$$\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q}\hat{\mathbf{f}}$$

- lower bound: $\mathcal{O}(L^3)$ Fourier coefficients and $\mathcal{O}(Q)$ rotations as input values
  $\Rightarrow \mathcal{O}(L^3 + Q)$ flops
- naive approach: matrix-multiplication with $\mathbf{D}_{\mathcal{R}_Q} \in \mathbb{C}^{Q \times |\mathcal{J}_L|}$
  $\Rightarrow \mathcal{O}(L^3 Q)$ flops
- our approach on nonequispaced grids on the $\mathrm{SO}(3)$:
  - generalizing the algorithm for the Fourier transform of scattered data on the sphere $\mathbb{S}^2$ (Jens' talk yesterday)

A note on complexity

$$\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q} \hat{\mathbf{f}}$$

- lower bound: $\mathcal{O}(L^3)$ Fourier coefficients and $\mathcal{O}(Q)$ rotations as input values
  $\Rightarrow \mathcal{O}(L^3 + Q)$ flops
- naive approach: matrix-multiplication with $\mathbf{D}_{\mathcal{R}_Q} \in \mathbb{C}^{Q \times |\mathcal{J}_L|}$
  $\Rightarrow \mathcal{O}(L^3 Q)$ flops
- our approach on nonequispaced grids on the $\mathrm{SO}(3)$:
  - an approximate algorithm called the *nonequispaced fast* $\mathrm{SO}(3)$ *Fourier transform (NFSOFT)* using the Fast Polynomial transform (Potts/Prestin/V.)
    $\Rightarrow \mathcal{O}(L^3 \log^2 L + Q)$

A note on complexity

$$\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q} \hat{\mathbf{f}}$$

- lower bound: $\mathcal{O}(L^3)$ Fourier coefficients and $\mathcal{O}(Q)$ rotations as input values
  $\Rightarrow \mathcal{O}(L^3 + Q)$ flops
- naive approach: matrix-multiplication with $\mathbf{D}_{\mathcal{R}_Q} \in \mathbb{C}^{Q \times |\mathcal{J}_L|}$
  $\Rightarrow \mathcal{O}(L^3 Q)$ flops
- our approach on nonequispaced grids on the $\mathrm{SO}(3)$:
  - an approximate algorithm called the *nonequispaced fast* $\mathrm{SO}(3)$ *Fourier transform (NFSOFT)* using the Fast Polynomial transform (Potts/Prestin/V.)
    $\Rightarrow \mathcal{O}(L^3 \log^2 L + Q)$
  - a variation of the NFSOFT using a fast algorithm based on semiseparable matrices (Keiner/V., in progress)
    $\Rightarrow \mathcal{O}(L^3 \log L + Q)$

We want to compute: $\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q}\hat{\mathbf{f}}$

Direct multiplication is slow.

Can we decompose $\mathbf{D}_{\mathcal{R}_Q}$ into a product of matrices that can be computed with faster?

We surely do.

The NFSOFT computes $\mathbf{f} = \mathbf{F}_{\mathcal{R}_Q}\mathbf{A}\mathbf{P}\,\hat{\mathbf{f}}$

A closer look at the NFSOFT

We want to compute: $\mathbf{f} = \mathbf{D}_{\mathcal{R}_Q}\hat{\mathbf{f}}$

$\Big\downarrow$ Direct multiplication is slow.

Can we decompose $\mathbf{D}_{\mathcal{R}_Q}$ into a product of matrices that can be computed with faster?

$\Big\downarrow$ We surely do.

The NFSOFT computes $\mathbf{f} = \mathbf{F}_{\mathcal{R}_Q}\mathbf{A}\mathbf{P}\,\hat{\mathbf{f}}$

$\longrightarrow$ only matrix $\mathbf{F}_{\mathcal{R}_Q}$ depends on input rotations

The NFSOFT, Step 1: Rearranging sums

Basic Idea: Turning the NFSOFT into a three-dimensional NFFT

We split up the Wigner-D functions according to the Euler angles of $f(\mathbf{R}_q) = f(\alpha_q, \beta_q, \gamma_q) \in \mathbb{D}_L$ for $q = 1, \ldots, Q$:

$$\begin{aligned} f(\alpha_q, \beta_q, \gamma_q) &= \sum_{l=0}^{L} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{m,n} D_l^{m,n}(\alpha_q, \beta_q, \gamma_q) \\ &= \sum_{l=0}^{L} \sum_{m=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{m,n} \mathrm{e}^{-\mathrm{i}m\alpha_q} \mathrm{e}^{-\mathrm{i}n\gamma_q} d_l^{m,n}(\cos\beta_q) \end{aligned}$$

and rearrange these sums:

$$f(\alpha_q, \beta_q, \gamma_q) = \sum_{m=-L}^{L} \mathrm{e}^{-\mathrm{i}m\alpha_q} \sum_{n=-L}^{L} \mathrm{e}^{-\mathrm{i}n\gamma_q} \sum_{l=\max(|m|,|n|)}^{L} \hat{f}_l^{m,n} d_l^{m,n}(\cos\beta_q).$$

The NFSOFT, Step 1: Rearranging sums

We got

$$f(\alpha_q, \beta_q, \gamma_q) \quad = \quad \underbrace{\sum_{m=-L}^{L} \mathrm{e}^{-\mathrm{i}m\alpha_q} \sum_{n=-L}^{L} \mathrm{e}^{-\mathrm{i}n\gamma_q}} \underbrace{\sum_{l=\mathsf{max}(|m|,|n|)}^{L} \widehat{f}_l^{m,n} d_l^{m,n}(\cos\beta_q)} \,.$$

The NFSOFT, Step 1: Rearranging sums

We got

$$f(\alpha_q, \beta_q, \gamma_q) = \underbrace{\sum_{m=-L}^{L} \mathrm{e}^{-\mathrm{i}m\alpha_q} \sum_{n=-L}^{L} \mathrm{e}^{-\mathrm{i}n\gamma_q}}_{} \underbrace{\sum_{l=\max(|m|,|n|)}^{L} \widehat{f}_l^{m,n} d_l^{m,n}(\cos\beta_q)}_{}\,.$$

- A closer look at the Wigner-d functions:

$$d_l^{m,n}(x) = \mathrm{const} \sqrt{\frac{(1-x)^{n-m}}{(1+x)^{m+n}}} \frac{\mathrm{d}^{l-m}}{\mathrm{d}x^{l-m}} \frac{(1+x)^{n+l}}{(1-x)^{n-l}}$$

- for $m+n$ even: $d_l^{m,n}(x)$ are polynomials of degree at most l
- for $m+n$ odd: $(1-x^2)^{-1/2} d_l^{m,n}(x)$ are polynomials of degree $l-1$

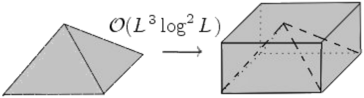
1. For a fixed pair of m, n we want to transform the $\mathrm{SO}(3)$-Fourier coefficients into Chebyshev coefficients

$$\sum_{l=\max(|m|,|n|)}^{L} \widehat{f}_l^{m,n} d_l^{m,n}(\cos\beta) = \begin{cases} \sum_{l=0}^{L} t_l^{m,n} T_l(\cos\beta) & \text{for } m+n \text{ even,} \\ \sin\beta \sum_{l=0}^{L-1} t_l^{m,n} T_l(\cos\beta) & \text{for } m+n \text{ odd} \end{cases}$$

2. and these Chebyshev coefficients into 'standard' Fourier coefficients using $T_l(\cos\beta) = \cos l\beta = \frac{1}{2}(\mathrm{e}^{\mathrm{i}l\beta} + \mathrm{e}^{-\mathrm{i}l\beta})$ and $\sin\beta = \frac{i}{2}(\mathrm{e}^{-\mathrm{i}\beta} - \mathrm{e}^{\mathrm{i}\beta})$

$$\sum_{l=0}^{L} t_l^{m,n} T_l(\cos\beta_q)(\sin\beta_q)^{\mathrm{mod}(m+n,2)} = \sum_{l=-L}^{L} \widehat{h}_l^{m,n} e^{-\mathrm{i}l\beta_q}.$$

- only coefficients are transformed ($\Rightarrow$ node-independent)



- compute

$$\mathbf{t}^{m,n} = \mathbf{P}^{m,n}\widehat{\mathbf{f}}^{m,n}$$

with $\widehat{\mathbf{f}}^{m,n} = \left(\widehat{f}^{m,n}_{\max(|m|,|n|)}, \dots, \widehat{f}^{m,n}_{L}\right)^T$, $\mathbf{t}^{m,n} = (t_0^{m,n}, \dots, t_L^{m,n})^T$

- The multiplication with $\mathbf{P}^{m,n} \in \mathbb{R}^{L-\max(|m|,|n|)\times L+1}$ is realised via

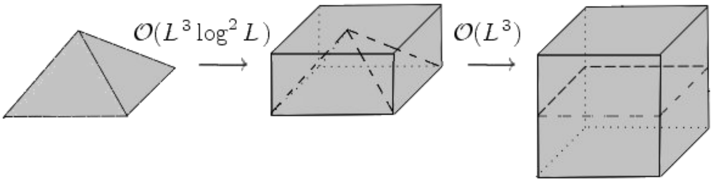
Fast Polynomial Transform

in $\mathcal{O}(L\log^2 L)$ flops per set of orders m, n for $(2L+1)^2$ many vectors $\mathbf{t}^{m,n}$

The NFSOFT, Step 2: Transforming the coefficients

2. Chebyshev coefficients can be transformed easily into 'standard' Fourier coefficients

$$\sum_{l=0}^{L} t_l^{m,n} T_l(\cos\beta_q)(\sin\beta_q)^{\mathrm{mod}(m+n,2)} = \sum_{l=-L}^{L} \widehat{h}_l^{m,n} e^{-\mathrm{i}l\beta_q}.$$



After that we get:

$$\begin{aligned} f(\alpha_q,\beta_q,\gamma_q) &= \sum_{m=-L}^{L} \mathrm{e}^{-\mathrm{i}m\alpha_q} \sum_{n=-L}^{L} \mathrm{e}^{-\mathrm{i}n\gamma_q} \sum_{l=\max(|m|,|n|)}^{L} \widehat{f}_l^{m,n} d_l^{m,n}(\cos\beta) \\ &= \sum_{m=-L}^{L} \sum_{n=-L}^{L} \sum_{l=-L}^{L} h_l^{m,n} \mathrm{e}^{-\mathrm{i}(m\alpha_q+n\gamma_q+l\beta_q)}. \end{aligned}$$

The NFSOFT, Step 3: The 3D-NFFT

Now, we are left with the computation of the three dimensional NFFT

$$f(\alpha_q, \beta_q, \gamma_q) \quad = \quad \sum_{m=-L}^{L} \sum_{n=-L}^{L} \sum_{l=-L}^{L} h_l^{m,n} \mathrm{e}^{-\mathrm{i}(m\alpha_q + n\gamma_q + l\beta_q)}$$

with complexity $\mathcal{O}(L^3 \log L + Q)$.

The NFSOFT

The NFSOFT computes

$$\boxed{\mathbf{f} = \mathbf{F}_{\mathcal{R}_Q} \mathbf{A} \mathbf{P} \, \widehat{\mathbf{f}}}$$

- a block diagonal matrix consisting of the matrices $\mathbf{P}^{m,n}$ representing the Fast Polynomial Transforms:

$$\mathbf{P} = \operatorname{diag}\left(\mathbf{P}^{m,n}\right)_{m,n=-L,\ldots,L}$$

The NFSOFT

The NFSOFT computes

$$\boxed{\mathbf{f} = \mathbf{F}_{\mathcal{R}_Q} \mathbf{A} \mathbf{P} \,\widehat{\mathbf{f}}}$$

- a block diagonal matrix consisting of the matrices $\mathbf{P}^{m,n}$ representing the Fast Polynomial Transforms:

$$\mathbf{P} = \operatorname{diag}\left(\mathbf{P}^{m,n}\right)_{m,n=-L,\ldots,L}$$

- a block diagonal matrix composed of blocks $\mathbf{A}^{m,n}$ that performs the change from Chebyshev to standard Fourier coefficients for each pair of orders m,n:

$$\mathbf{A} = \operatorname{diag}\left(\mathbf{A}^{m,n}\right)_{m,n=-L,\ldots,L}$$

The NFSOFT

The NFSOFT computes

$$\boxed{\mathbf{f} = \mathbf{F}_{\mathcal{R}_Q} \mathbf{A} \mathbf{P} \,\widehat{\mathbf{f}}}$$

- a block diagonal matrix consisting of the matrices $\mathbf{P}^{m,n}$ representing the Fast Polynomial Transforms:

$$\mathbf{P} = \operatorname{diag}\left(\mathbf{P}^{m,n}\right)_{m,n=-L,\ldots,L}$$

- a block diagonal matrix composed of blocks $\mathbf{A}^{m,n}$ that performs the change from Chebyshev to standard Fourier coefficients for each pair of orders m, n:

$$\mathbf{A} = \operatorname{diag}\left(\mathbf{A}^{m,n}\right)_{m,n=-L,\ldots,L}$$

- a three-dimensional Fourier matrix $\mathbf{F}_{\mathcal{R}_Q}$:

$$\mathbf{F}_{\mathcal{R}_Q} = \left(\mathrm{e}^{-\mathrm{i}(m\alpha_q + l\beta_q + n\gamma_q)}\right)_{q=1,\ldots,Q;(l,m,n)\in\mathcal{J}_L}.$$

Summary

Let $\mathcal{R}_Q = (\mathbf{R}_1, \ldots, \mathbf{R}_Q)$ be a finite sequence of **arbitrary** rotations $\mathbf{R}_q \in \mathrm{SO}(3)$. Then

$$f(\mathbf{R}_q) = \sum_{l=0}^{L} \sum_{k=-l}^{l} \sum_{n=-l}^{l} \hat{f}_l^{m,n} D_l^{m,n}(\mathbf{R}_q), \quad q = 1, \ldots, Q,$$

evaluates a polynomial $f \in \mathbb{D}_L$ at rotations $\mathbf{R}_1, \ldots, \mathbf{R}_Q$ given its Fourier coefficients $\hat{\mathbf{f}} = (\hat{f}_l^{m,n})_{(l,m,n)\in\mathcal{J}_L}$; and is called

nonequispaced discrete $\mathrm{SO}(3)$ Fourier transform (NDSOFT).

It can be computed for Q arbitrary rotations in $\mathcal{O}(L^3 \log^2 L + Q)$ flops via the

nonequispaced fast $\mathrm{SO}(3)$ Fourier transform (NFSOFT)

Outline

1. The Rotation Group
2. Fourier Analysis on the Rotation Group
3. Algorithms for $\mathrm{SO}(3)$ Fourier Transforms
4. Applications

Fast summation of radial functions on $\mathrm{SO}(3)$

- A radial function $f \in L^2(\mathrm{SO}(3))$ with center $\mathbf{R}_0 \in \mathrm{SO}(3)$ is a function that depends only on the distance to $\mathbf{R}_0$
- What is the distance between two rotations?
  - from axis-angle representation: the rotation angle is the absolute value of a rotation
  - We use the absolute value of a rotation $\mathbf{R} = \mathbf{R}_0\mathbf{R}_1^{-1}$ that turns $\mathbf{R}_1$ onto $\mathbf{R}_0$ as a measure for the distance between $\mathbf{R}_1$ and $\mathbf{R}_0$

$$d(\mathbf{R}_0, \mathbf{R}_1) = \arccos \frac{1}{2}(\mathrm{trace}(\mathbf{R}_0\mathbf{R}_1^{-1}) - 1)$$

- compare: summation of Gaussians on the sphere (Jens' talk yesterday)

Radial Functions on $\mathrm{SO}(3)$:

$\psi \in L^2(\mathrm{SO}(3))$ is a radial function with center $\mathbf{R}_0 \in \mathrm{SO}(3)$ if and only if there is a sequence of coefficients $\hat{\psi}_l$ such that

$$\hat{\psi}_l^{m,n} = \hat{\psi}_l D_l^{m,n}(\mathbf{R}_0), \quad l \in \mathbb{N}_0,\, m,n = -l,\dots,l.$$

In particular,

$$\psi(\mathbf{R}) \sim \sum_{l \in \mathbb{N}_0} \hat{\psi}_l \sum_{m,n=-l}^{l} D_l^{m,n}(\mathbf{R}_0) D_l^{m,n}(\mathbf{R}) \sim \sum_{l \in \mathbb{N}_0} \hat{\psi}_l \, U_{2l}\Big(\cos \frac{d(\mathbf{R}_0, \mathbf{R})}{2}\Big),$$

where

$$U_l(\cos\omega) = \frac{\sin(l+1)\omega}{\sin\omega}, \quad l \in \mathbb{N}_0,\, \omega \in (0,\pi)$$

denotes the Chebyshev polynomials of second kind with $U_l(1) = l+1$.

Fast Summation

- Input:
  - $\mathcal{R}_Q = (\mathbf{R}_1, \ldots, \mathbf{R}_Q)$ $\mathbf{R}_q \in \mathrm{SO}(3)$ a list of source rotations
  - $\mathcal{T}_S = (\mathbf{T}_1, \ldots, \mathbf{T}_S)$, $\mathbf{T}_s \in \mathrm{SO}(3)$ a list of target rotations,
  - $\psi\colon \mathrm{SO}(3) \to \mathbb{C}$ a pointwise given radial function,
  - $\mathbf{c} = (c_1, \ldots, c_Q) \in \mathbb{C}^Q$ a coefficient vector.

Fast Summation

- Input:
  - $\mathcal{R}_Q = (\mathbf{R}_1, \ldots, \mathbf{R}_Q)$ $\mathbf{R}_q \in \mathrm{SO}(3)$ a list of source rotations
  - $\mathcal{T}_S = (\mathbf{T}_1, \ldots, \mathbf{T}_S)$, $\mathbf{T}_s \in \mathrm{SO}(3)$ a list of target rotations,
  - $\psi \colon \mathrm{SO}(3) \to \mathbb{C}$ a pointwise given radial function,
  - $\mathbf{c} = (c_1, \ldots, c_Q) \in \mathbb{C}^Q$ a coefficient vector.
- We are concerned with evaluating the sum

$$f(\mathbf{T}_s) = \sum_{q=1}^{Q} c_q \psi(\mathbf{T}_s \mathbf{R}_q^{-1}), \quad s = 1, \ldots, S,$$

Fast Summation

- Input:
  - $\mathcal{R}_Q = (\mathbf{R}_1, \ldots, \mathbf{R}_Q)$ $\mathbf{R}_q \in \mathrm{SO}(3)$ a list of source rotations
  - $\mathcal{T}_S = (\mathbf{T}_1, \ldots, \mathbf{T}_S)$, $\mathbf{T}_s \in \mathrm{SO}(3)$ a list of target rotations,
  - $\psi \colon \mathrm{SO}(3) \to \mathbb{C}$ a pointwise given radial function,
  - $\mathbf{c} = (c_1, \ldots, c_Q) \in \mathbb{C}^Q$ a coefficient vector.
- We are concerned with evaluating the sum

$$f(\mathbf{T}_s) = \sum_{q=1}^{Q} c_q \psi(\mathbf{T}_s \mathbf{R}_q^{-1}), \quad s = 1, \ldots, S,$$

- Approximate ψ by its truncated Fourier series expansion for $L \in \mathbb{N}_0$

$$\begin{aligned} f(\mathbf{T}_s) &\approx \sum_{q=1}^{Q} c_q \sum_{l=0}^{L} \sum_{m,n=-l}^{l} \hat{\psi}(l) \overline{D_l^{m,n}(\mathbf{R}_q)} D_l^{m,n}(\mathbf{T}_s) \\ &= \sum_{l=0}^{L} \sum_{m,n=-l}^{l} \hat{\psi}(l) \left(\sum_{q=1}^{Q} c_q \overline{D_l^{m,n}(\mathbf{R}_q)} \right) D_l^{m,n}(\mathbf{T}_s) \end{aligned}$$

Fast Summation Algorithm

$$f(\mathbf{T}_s) = \underbrace{\sum_{l=0}^{L} \sum_{m,n=-l}^{l} \hat{\psi}(l) \underbrace{\left(\sum_{q=1}^{Q} c_q \overline{D_l^{m,n}(\mathbf{R}_q)} \right)}_{\text{adjoint NFSOFT}} D_l^{m,n}(\mathbf{T}_s)}_{\text{NFSOFT}}$$

What to compute:

1. an adjoint NDSOFT for the Q source rotations $\mathbf{R}_q$
   $\longrightarrow$ NFSOFT algorithm with $\mathcal{O}(L^3 \log^2 L + Q)$ flops
2. multiply with $|\mathcal{J}_L|$-many $\hat{\psi}_l$
   $\longrightarrow$ $\mathcal{O}(L^3)$ flops
3. another NDSOFT for the S target rotations $\mathbf{T}_s$
   $\longrightarrow$ NFSOFT algorithm with $\mathcal{O}(L^3 \log^2 L + S)$ flops

Total: $\mathcal{O}(L^3 \log^2 L + Q + S)$ flops instead of $\mathcal{O}(QS)$

Correlating functions on the sphere

Given a function or pattern f on the sphere we want to identify its orientation and position, i.e. longitude and latitude.

$\longrightarrow$

Correlating functions on the sphere

Given a function or pattern f on the sphere we want to identify its orientation and position, i.e. longitude and latitude.

The Task

Find a rotation $\mathbf{R} \in \mathrm{SO}(3)$ which turns a function $f \in \mathrm{L}^2(\mathbb{S}^2)$ into the function $\tilde{f}$ with

$$\tilde{f}(\mathbf{x}) = f(\mathbf{R}^{-1}\mathbf{x}).$$

$\longrightarrow$

Correlating functions on the sphere

Finding this rotation $\mathbf{R} \in \mathrm{SO}(3)$ leads us to correlating the functions f and $\tilde{f}$ by

$$\max_{\mathbf{R}} C(\mathbf{R}) = \int_{\mathbb{S}^2} f(\mathbf{x}) \overline{\tilde{f}(\mathbf{R}^{-1}\mathbf{x})} \mathrm{d}\mathbf{x}.$$

Correlating functions on the sphere

Finding this rotation $\mathbf{R} \in \mathrm{SO}(3)$ leads us to correlating the functions f and $\tilde{f}$ by

$$\max_{\mathbf{R}} C(\mathbf{R}) = \int_{\mathbb{S}^2} f(\mathbf{x}) \overline{\tilde{f}(\mathbf{R}^{-1}\mathbf{x})} \mathrm{d}\mathbf{x}.$$

Correlating functions on the sphere

Finding this rotation $\mathbf{R} \in \mathrm{SO}(3)$ leads us to correlating the functions f and $\tilde{f}$ by

$$\max_{\mathbf{R}} C(\mathbf{R}) = \int_{\mathbb{S}^2} f(\mathbf{x}) \overline{\tilde{f}(\mathbf{R}^{-1}\mathbf{x})} \mathrm{d}\mathbf{x}.$$

Naive attempt

Evaluation of $C(\mathbf{R})$ for a set of functions needs $\mathcal{O}(R^3 L^2)$ operations:

- $\mathcal{O}(R^3)$ different rotations $\mathbf{R}$
- $\mathcal{O}(L^2)$ flops to compute the inner product in $\mathrm{L}^2(\mathbb{S}^2)$.

Correlating functions on the sphere

Finding this rotation $\mathbf{R} \in \mathrm{SO}(3)$ leads us to correlating the functions f and $\tilde{f}$ by

$$\max_{\mathbf{R}} C(\mathbf{R}) = \int_{\mathbb{S}^2} f(\mathbf{x}) \overline{\tilde{f}(\mathbf{R}^{-1}\mathbf{x})} \mathrm{d}\mathbf{x}.$$

Naive attempt

Evaluation of $C(\mathbf{R})$ for a set of functions needs $\mathcal{O}(R^3 L^2)$ operations:

- $\mathcal{O}(R^3)$ different rotations $\mathbf{R}$
- $\mathcal{O}(L^2)$ flops to compute the inner product in $\mathrm{L}^2(\mathbb{S}^2)$.

Instead:

NFSOFT

Evaluation of $C(\mathbf{R})$ for $\mathcal{O}(R^3)$ different rotations in $\mathcal{O}(L^3 \log^2 L + R^3)$ operations

Spherical Fourier expansion

- The Fourier expansions of the two L-band-limited functions $f, \tilde{f} \in \mathrm{L}^2(\mathbb{S}^2)$ are

$$\tilde{f}(\mathbf{x}) = \sum_{l=0}^{L-1} \sum_{m=-l}^{l} a_l^m Y_l^m(\mathbf{x}) \text{ and } f(\mathbf{x}) = \sum_{l=0}^{L-1} \sum_{m=-l}^{l} b_l^m Y_l^m(\mathbf{x}).$$

  Their spherical Fourier coefficients can be computed using the NFSFT in $\mathcal{O}(L^2 \log^2 L)$ flops.
- use rotation invariance of spherical harmonics

$$Y_l^n(\mathbf{R}^{-1}\mathbf{x}) = \sum_{m=-l}^{l} D_l^{mn}(\mathbf{R}) Y_l^m(\mathbf{x})$$

Correlation via NFSOFT

Inserting these expansions into $C(\mathbf{R})$ yields

$$
\begin{aligned}
C(\mathbf{R}) &= \int_{\mathbb{S}^2} \tilde{f}(\mathbf{x})\overline{f(\mathbf{R}^{-1}\mathbf{x})}\mathrm{d}\mathbf{x} \\
&= \int_{\mathbb{S}^2} \left(\sum_{l=0}^{L-1}\sum_{m=-l}^{l} a_l^m Y_l^m(\mathbf{R}^{-1}\mathbf{x})\right) \overline{\left(\sum_{l'=0}^{L-1}\sum_{n=-l'}^{l'} b_{l'}^n Y_{l'}^n(\mathbf{x})\right)}\mathrm{d}\mathbf{x} \\
&= \sum_{l=0}^{B-1}\sum_{m=-l}^{l}\sum_{n=-l}^{l}\sum_{k=-l}^{l} \overline{D_l^{mk}(\mathbf{R}^{-1})}a_l^m\overline{b_l^n}\int_{\mathbb{S}^2} Y_l^m(\mathbf{x})\overline{Y_l^k(\mathbf{x})}\mathrm{d}\mathbf{x}. \\
&= \sum_{l=0}^{L-1}\sum_{m=-l}^{l}\sum_{n=-l}^{l} \underbrace{(-1)^{m-n}a_l^{-m}\overline{b_l^{-n}}}\, D_l^{mn}(\mathbf{R})
\end{aligned}
$$

It remains to compute this actual NFSOFT for $\mathcal{O}(R^3)$ rotations in $\mathcal{O}(L^3\log^2 L + R^3)$ flops.

Program

Part I – Fourier Analysis and the FFT
Stefan, Monday, 14:15 – 16:00, Room U322

Part II – Orthogonal Polynomials
Jens, Tuesday, 12:15 – 14:00, Room U141 (Lecture Hall F)

Practice Session: 14:30 – 16:00, Room Y339b (Basics and Matlab Hands-On)

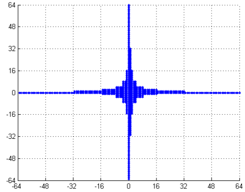
Part III – Fast Polynomial Transforms and Applications
Jens, Wednesday, 12:15 – 14:00, Room U345

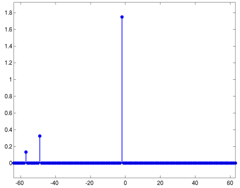
Practice Session: 14:30 – 16:00, Room Y338c (C Library Hands-On)

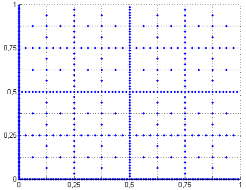
Part IV – Fourier Transforms on the Rotation Group
Antje, Thursday, 14:15 – 16:00, Room U322

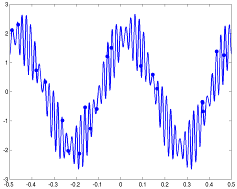
Part V – High Dimensions and Reconstruction
Stefan, Friday, 10:15 – 12:00, Room U322

Part V – High Dimensions and Reconstruction









Outline

1. NFFT on the Hyperbolic Cross
2. Compressed Sensing
3. Least Squares
4. Optimal Interpolation

NFFT on the Hyperbolic Cross

- $\mathbb{T} \simeq [0,1)$, $f \in C(\mathbb{T})$, $n \in \mathbb{N}$, $x_j = j/2^n$

$$
\begin{aligned}
\hat{f}_k &= \int_{\mathbb{T}} f(x)\,\mathrm{e}^{-2\pi \mathrm{i} k x} \mathrm{d}x \\
&\approx \frac{1}{2^n} \sum_{j=0}^{2^n-1} f(x_j)\,\mathrm{e}^{-2\pi \mathrm{i} k x_j}, \quad k = -2^{n-1}+1, \dots, 2^{n-1},
\end{aligned}
$$

- discrete Fourier transform (DFT)

$$
f(x_j) = \sum_{k=-2^{n-1}+1}^{2^{n-1}} \hat{f}_k \mathrm{e}^{2\pi \mathrm{i} k x_j}, \quad j = 0, \dots, 2^n - 1
$$

- Complexity
  - DFT: $\mathcal{O}\left(2^{2n}\right)$
  - FFT: $\mathcal{O}\left(n 2^n\right)$

NFFT on the Hyperbolic Cross

- discrete Fourier transformation

$$
\begin{aligned}
f(\mathbf{x}) &= \sum_{\mathbf{k}\in {G'_n}^d} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi\mathrm{i}\mathbf{k}\mathbf{x}}, \\
{G'_n}^d &= \left\{-2^{n-1}+1,\ldots,2^{n-1}\right\}^d \subset \mathbb{Z}^d \\
\mathbf{x} &= \left(\frac{j_1}{2^n},\ldots,\frac{j_d}{2^n}\right)^T \in \mathbb{T}^d, \quad j_1,\ldots,j_d \in \{0,\ldots,2^n-1\}
\end{aligned}
$$

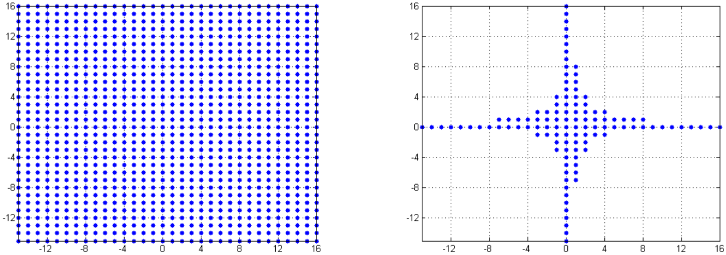
- Complexity, problem size 2^{nd}
  - DFT: $\mathcal{O}\left(2^{2nd}\right)$ or $\mathcal{O}\left(2^{n(d+1)}\right)$
  - FFT: $\mathcal{O}\left(nd2^{nd}\right)$
- Limitation of the FFT
  - complexity increases fast with d
  - equidistant grid

NFFT on the Hyperbolic Cross

- Goal: fast algorithm to evaluate

$$f(\mathbf{x}) = \sum_{\mathbf{k}\in H_n^d} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi \mathrm{i}\mathbf{k}\mathbf{x}}$$

at arbitrary nodes $\mathbf{x} \in \mathbb{T}^d = [0,1)^d$



(a) full grid (b) hyperbolic cross

NFFT on the Hyperbolic Cross

- one dimensional grid in frequency domain

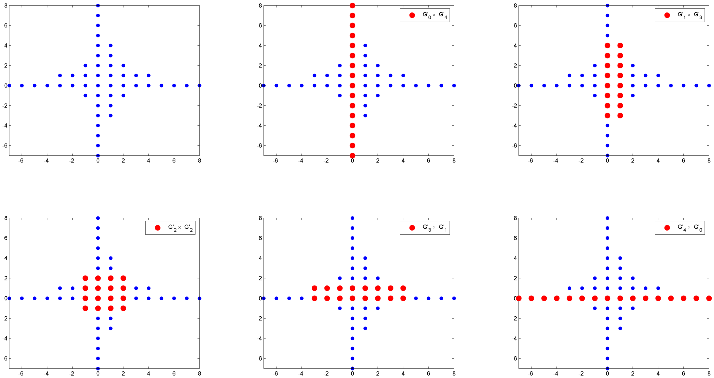
$$G'_n = \{-2^{n-1}+1, \ldots, 2^{n-1}\}, \quad G'_0 = \{0\}$$

- hyperbolic cross, dimension $d \in \mathbb{N}$, refinement $n \in \mathbb{N}_0$

$$H_n^d = \bigcup_{\substack{\|\mathbf{q}\|_1 = n \\ \mathbf{q} \in \mathbb{N}_0^d}} G'_{q_1} \times \ldots \times G'_{q_d}$$

- $\mathbf{k} \in H_n^d \;\Rightarrow\; |k_1 \cdots k_d| \le 2^{n-d}$

Hyperbolic cross H_4^2



NFFT on the Hyperbolic Cross

- Evaluation of

$$f(\mathbf{x}) = \sum_{\mathbf{k}\in H_n^d} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi \mathrm{i}\mathbf{k}\mathbf{x}}$$

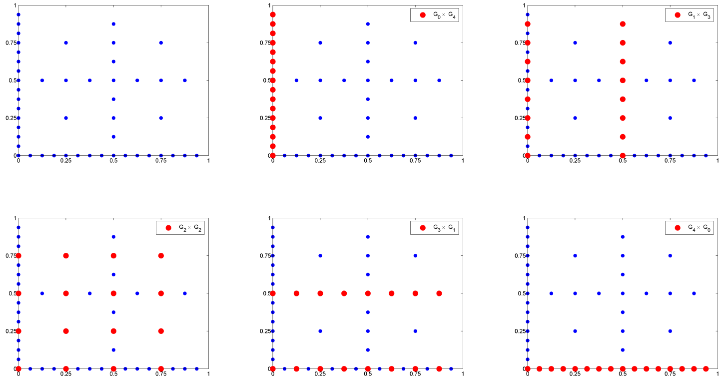
- one dimensional grid G_n in spatial domain

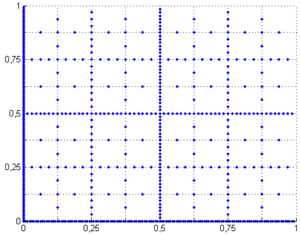
$$G_n = \left\{ \frac{j}{2^n} : j = 0, \ldots, 2^n - 1 \right\}$$

- sparse grid

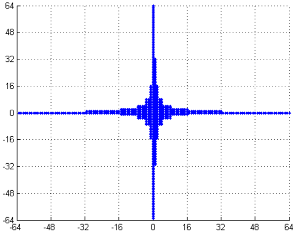
$$S_n^d = \bigcup_{\substack{\|\mathbf{q}\|_1 = n \\ \mathbf{q}\in\mathbb{N}_0^d}} G_{q_1} \times \ldots \times G_{q_d}$$

Sparse grid S_4^2

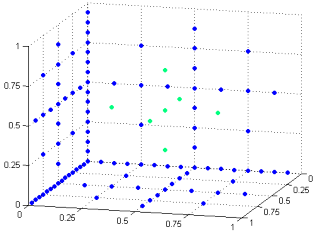




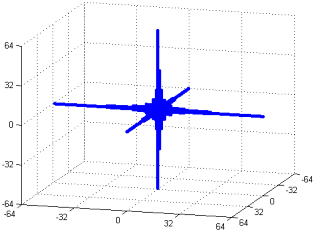
(a) Sparse grid S_7^2



(b) Hyperbolic cross H_7^2



(c) Sparse grid S_4^3



(d) Hyperbolic cross H_7^3

NFFT on the Hyperbolic Cross

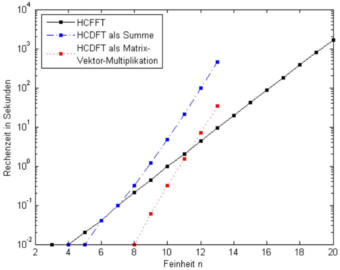
- discrete Fourier transform on the hyperbolic cross

$$f(\mathbf{x}) = \sum_{\mathbf{k}\in H_n^d} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi \mathrm{i}\mathbf{k}\mathbf{x}}, \quad \mathbf{x} \in S_n^d$$

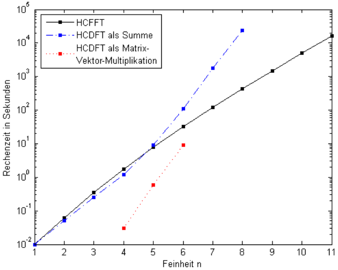
- $\left|H_n^d\right| = \left|S_n^d\right| = \mathcal{O}\left(n^{d-1}2^n\right)$

- HCDFT: $\mathcal{O}\left(n^{2d-2}2^{2n}\right)$ flops
- HCFFT: $\mathcal{O}\left(n^{d}2^{n}\right)$ flops [Baszenski, Delvos 1989; Hallatschek 1992; Gradinaru 2007]

Computational times HCDFT / HCFFT



(a) $d = 2$



(b) $d = 10$

- cardinal B-Spline

$$N_m = \begin{cases} \chi_{[0,1)}, & m = 1 \\ N_{m-1} * N_1 = \int_0^1 N_{m-1}(\cdot - t)dt, & m \geq 2 \end{cases}$$

- dilated periodisation $\phi_n : \mathbb{T} \to \mathbb{R}$,

$$\phi_n = \sum_{j \in \mathbb{Z}} N_m \left(2^n(\cdot + j)\right)$$

- translations $\phi_{n,k} : \mathbb{T} \to \mathbb{R}$,

$$\phi_{n,k} = \phi_n \left(\cdot - \frac{k}{2^n} \right), \quad k = 0, \ldots, 2^n - 1$$

NFFT on the Hyperbolic Cross

- Spline space $V_n = \mathrm{span}\{\phi_{n,k} : k = 0, \dots, 2^n - 1\}$, $m \in 2\mathbb{N}$
- Interpolation operator $\mathcal{L}_n : C(\mathbb{T}) \to V_n$ fulfils

$$\mathcal{L}_n f\left(\frac{l}{2^n}\right) = f\left(\frac{l}{2^n}\right), \quad l = 0, \dots, 2^n - 1$$

- Spline coefficients $a_{n,k} \in \mathbb{C}$ in

$$\mathcal{L}_n f = \sum_{k=0}^{2^n-1} a_{n,k}\phi_{n,k}$$

can be computed in $\mathcal{O}\left(2^n\right)$ flops [Berger, Strömberg 1995; Bittner 1999]

NFFT on the Hyperbolic Cross

- Spline space

$$V_n^{(2)} = \text{span}\left\{\phi_{j,k} \otimes \phi_{n-j,l} : j = 0, \ldots, n, \right.$$
$$\left. k = 0, \ldots, 2^j - 1,\ l = 0, \ldots, 2^{n-j} - 1\right\}$$

- $\mathcal{L}_n^{(2)} : C\left(\mathbb{T}^2\right) \to V_n^{(2)}$, $\mathcal{L}_n^{(2)} = \sum_{j=0}^{n} \mathcal{L}_j \otimes \left(\mathcal{L}_{n-j} - \mathcal{L}_{n-j-1}\right)$

$$\mathcal{L}_n^{(2)} f(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in S_n^2$$

- Spline coefficients $a_{j,(k,l)} \in \mathbb{C}$ in

$$\mathcal{L}_n^{(2)} f = \sum_{j=0}^{n} \sum_{k=0}^{2^j-1} \sum_{l=0}^{2^{n-j}-1} a_{j,(k,l)} \phi_{j,k} \otimes \phi_{n-j,l}$$

can be computed in $\mathcal{O}\left(n2^n\right)$ flops

- Evaluation of $\mathcal{L}_n^{(2)} f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{T}^2$ takes $\mathcal{O}\left(n\right)$ flops

Algorithm NHCFFT Input: $m \in 2\mathbb{N}$, $\alpha \geq 2$, $\mathcal{X} \subset \mathbb{T}^2$, $\hat{\mathbf{f}} \in \mathbb{C}^{|H_n^2|}$

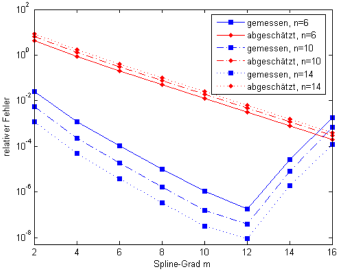
1. HCFFT: Compute

$$f(\mathbf{x}) = \sum_{\mathbf{k} \in H_{n+\alpha}^2} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi \mathrm{i} \mathbf{k} \mathbf{x}}$$

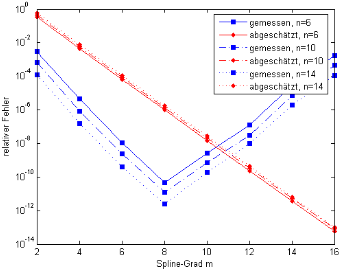
   for $\mathbf{x} \in S_{n+\alpha}^2$, where $\hat{f}_{\mathbf{k}} = 0$ for $\mathbf{k} \in H_{n+\alpha}^2 \backslash H_n^2$

2. Spline interpolation $s = \mathcal{L}_{n+\alpha}^{(2)} f$
3. Evaluation of $s(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$

Interpolation error $\|f - s\|_\infty / \|\hat{\mathbf{f}}\|_1$ with respect to m

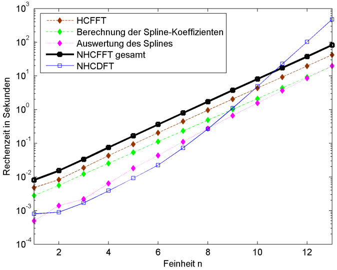


(a) $\alpha = 2$

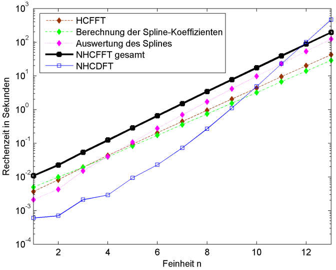


(b) $\alpha = 4$

Computational times



(a) $m = 4,\ \alpha = 2$



(b) $m = 10,\ \alpha = 2$

HCFFT	$\mathcal{O}\left(n^2 2^n\right)$
Spline interpolation	$\mathcal{O}\left(n 2^n\right)$
Spline evaluation	$\mathcal{O}\left(n^2 2^n\right)$
NHCFFT total	$\mathcal{O}\left(n^2 2^n\right)$
NHCDFT	$\mathcal{O}\left(n^2 2^{2n}\right)$

NFFT on the Hyperbolic Cross

- Spline space

$$V_n^{(d)} = \text{span}\{\bigotimes_{l=1}^{d} \phi_{j_l,k_l} : \mathbf{j}, \mathbf{k} \in \mathbb{N}_0^d,\ \|\mathbf{j}\|_1 = n, k_l = 0, \ldots, 2^{j_l}-1\}$$

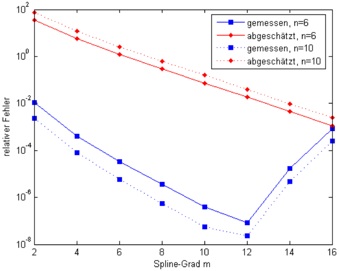
- Interpolation operator $\mathcal{L}_n^{(d)} : C(\mathbb{T}^d) \to V_n^{(d)}$,

$$\mathcal{L}_n^{(d)} = \bigoplus_{\substack{\|\mathbf{j}\|_1 = n \\ \mathbf{k} \in \mathbb{N}_0^d}} \mathcal{L}_{j_1} \otimes \ldots \otimes \mathcal{L}_{j_d}$$

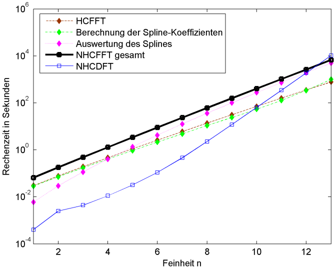
Theorem: Let $n \in \mathbb{N}_0$, $m \in 2\mathbb{N}$, $\alpha \in \mathbb{N}$, $f(\mathbf{x}) = \sum_{\mathbf{k} \in H_n^d} \hat{f}_{\mathbf{k}} \mathrm{e}^{2\pi \mathrm{i}\mathbf{k}\mathbf{x}}$ and $s = \mathcal{L}_{n+\alpha}^{(d)} f$, then

$$\|f - s\|_\infty \le \frac{(2(n+\alpha)+2)^{d-1} F_m^d}{2^{(\alpha-d+1)m}} \|\hat{\mathbf{f}}\|_1, \quad \frac{\pi^2}{8} \le F_m < \frac{4}{\pi}.$$

NFFT on the Hyperbolic Cross



(a) $d = 3$, $m = 4$, $\alpha = 3$



(b) $d = 3$, $m = 4$, $\alpha = 3$

HCFFT	$\mathcal{O}\left(n^d 2^n\right)$
Spline interpolation	$\mathcal{O}\left(n^{d-1} 2^n\right)$
Spline evaluation	$\mathcal{O}\left(n^{2d-2} 2^n\right)$
NHCFFT total	$\mathcal{O}\left(n^{2d-2} 2^n\right)$
NHCDFT	$\mathcal{O}\left(n^{2d-2} 2^{2n}\right)$

So far: $\hat{\mathbf{f}} \in \mathbb{C}^N$ given, fast and approximate algorithms for

$$\mathbf{f} = \mathbf{A}\hat{\mathbf{f}}$$

Now: $\mathbf{y} \in \mathbb{C}^M$ given, solve

$$\mathbf{A}\hat{\mathbf{f}} \approx \mathbf{y}$$

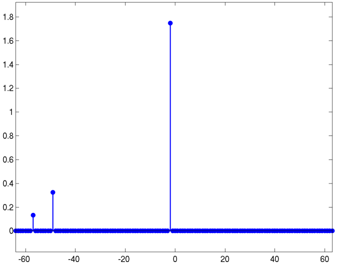
$M = N$ equispaced nodes $x_j = j/N$ yield

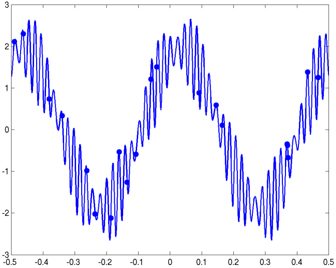
$$\mathbf{A}^{\mathsf{H}}\mathbf{A} = \mathbf{A}\mathbf{A}^{\mathsf{H}} = N\mathbf{I}$$

Considered problems:

$$\begin{aligned} \|\mathbf{A}\hat{\mathbf{f}} - \mathbf{y}\|_{\mathbf{W}} &\to \min \\ \|\hat{\mathbf{f}}\|_{\hat{\mathbf{W}}^{-1}} &\to \min \quad \text{s.t.} \quad \mathbf{A}\hat{\mathbf{f}} = \mathbf{y} \\ \|\hat{\mathbf{f}}\|_0 &\to \min \quad \text{s.t.} \quad \mathbf{A}\hat{\mathbf{f}} = \mathbf{y} \end{aligned}$$

Random sampling of sparse trigonometric polynomials [Rauhut, K.]





Compressed Sensing

Support $T \subset I_N = \{-\frac{N}{2}, \ldots, \frac{N}{2} - 1\}$, $S = |T| \ll N$, sparse trigonometric polynomials

$$f : \mathbb{T} \to \mathbb{C}, \quad f(x) = \sum_{k \in T} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} k x}$$

Nonlinear spaces of trigonometric polynomials

$$\Pi_T \subset \Pi_{I_N}, \quad \text{vs.} \quad \Pi_{I_N}(S) = \bigcup_{T \subset I_N : |T| \leq S} \Pi_T$$

Reconstruct $f \in \Pi_{I_N}(S)$ from samples y_j at nodes $x_j \in \mathbb{T}$, i.e.,

$$y_j = f(x_j) = \sum_{k \in I_N} \hat{f}_k \mathrm{e}^{-2\pi \mathrm{i} k x_j}, \quad j = 0, \ldots, M-1$$

Dimension N, Fourier coefficients $\hat{\mathbf{f}} \in \mathbb{C}^N$
Sparsity $S = |T|$, support $T = \text{supp}(\hat{\mathbf{f}})$
Number of samples M, samples $(x_j, y_j) \in \mathbb{T} \times \mathbb{C}$

Interesting case

$$S \sim M \ll N$$

Nonequispaced Fourier matrix and its T_s-restriction

$$\mathbf{A} = (\mathrm{e}^{-2\pi\mathrm{i}kx_j})_{j=0,\ldots,M-1;k\in I_N} = (\ldots\phi_k|\phi_{k+1}\ldots) \in \mathbb{C}^{M\times N}$$
$$\mathbf{A}_{T_s} = (\phi_k)_{k\in T_s} \in \mathbb{C}^{M\times|T_s|}$$

Sampling a trigonometric polynomial

$$\mathbf{y} = \mathbf{A}\hat{\mathbf{f}}$$

Compressed Sensing - Thresholding

Input: $\mathbf{y} \in \mathbb{C}^M$, maximum sparsity $S \in \mathbb{N}$

1: find $T \subset I_N$ to the S largest inner products $\{|\langle \mathbf{y}, \phi_l \rangle|\}_{l \in I_N}$
2: solve $\|\mathbf{A}_T \mathbf{c} - \mathbf{y}\|_2 \xrightarrow{\mathbf{c}} \min$
3: $(\hat{f}_k)_{k \in T} = \mathbf{c}$

Output: $\hat{\mathbf{f}} \in \mathbb{C}^N$, $T \subset I_N$

Remark:

- we might hope that $M^{-1}\langle \mathbf{y}, \phi_l \rangle = M^{-1} \sum_{j=0}^{M-1} f(x_j) \mathrm{e}^{2\pi \mathrm{i} l x_j} \approx \int_{\mathbb{T}} f(x) \mathrm{e}^{2\pi \mathrm{i} l x} \mathrm{d}x = \hat{f}_l$
- computation of the inner products by $(\langle \mathbf{y}, \phi_l \rangle)_{l \in I_N} = \mathbf{A}^{\mathsf{H}} \mathbf{y}$ in $\mathcal{O}(N \log N + M)$ floating point operations

Compressed Sensing - Matching Pursuit

Input: $\mathbf{y} \in \mathbb{C}^M$, $\varepsilon > 0$, maximum number of iterations $L \in \mathbb{N}$

$s = 0$, $\mathbf{r}_0 = \mathbf{y}$, $T_0 = \emptyset$, $\hat{\mathbf{f}} = 0$
repeat
  $s = s + 1$
  $k_s = \arg\max_{k \in I_N} |\langle \mathbf{r}_{s-1}, \phi_k \rangle|$
  $\hat{f}_{k_s} = \hat{f}_{k_s} + \langle \mathbf{r}_{s-1}, \phi_{k_s} \rangle$
  $\mathbf{r}_s = \mathbf{r}_{s-1} - \langle \mathbf{r}_{s-1}, \phi_{k_s} \rangle \phi_{k_s}$
  $T_s = T_{s-1} \cup \{k_s\}$
until $s = L$ or $\|\mathbf{r}_s\| \leq \varepsilon$
$T = T_s$

Output: $\hat{\mathbf{f}} \in \mathbb{C}^N$, $T \subset I_N$

Compressed Sensing - Orthogonal Matching Pursuit

Input: $\mathbf{y} \in \mathbb{C}^M$, $\varepsilon > 0$

1: $s = 0$, $\mathbf{r}_0 = y$, $T_0 = \emptyset$
2: **repeat**
3: $\quad s = s + 1$
4: $\quad T_s = T_{s-1} \cup \{\arg\max_{k \in I_N} |\langle \mathbf{r}_{s-1}, \phi_k \rangle|\}$
5: $\quad$ solve $\|\mathbf{A}_{T_s}\mathbf{d}_s - \mathbf{y}\|_2 \overset{\mathbf{d}_s}{\to} \min$
6: $\quad \mathbf{r}_s = \mathbf{y} - \mathbf{A}_{T_s}\mathbf{d}_s$
7: **until** $s = M$ or $\|\mathbf{r}_s\| \le \varepsilon$
8: $T = T_s$, $(\hat{f}_k)_{k \in T} = \mathbf{d}_s$

Output: $\hat{\mathbf{f}} \in \mathbb{C}^N$, $T \subset I_N$

Compressed Sensing - Thresholding

Theorem:

- fix $f \in \Pi_{I_N}(S)$
- define its dynamic range by

$$R = \frac{\max_{k \in T} |\hat{f}_k|}{\min_{k \in T} |\hat{f}_k|}$$

- choose sampling nodes $x_0, \dots, x_{M-1}$ independently and uniformly at random on $\mathbb{T}$ or on the grid $\frac{1}{N} I_N$
- if for some $\epsilon > 0$

$$M \geq C R^2 \cdot S \cdot \log(N/\epsilon)$$

- then with probability at least $1 - \epsilon$ thresholding recovers T and $\hat{\mathbf{f}}$

Theorem:

- fix $f \in \Pi_{I_N}(S)$, choose sampling nodes as before
- if for some $\epsilon > 0$

$$M \geq C \cdot S \cdot \log(N/\epsilon),$$

then with probability at least $1 - \epsilon$ orthogonal matching pursuit selects $k_1 \in \mathrm{supp}(\hat{\mathbf{f}})$

Theorem:

- choose sampling nodes as before
- if for some $\epsilon > 0$

$$M \geq C \cdot S^2 \cdot \log(N/\epsilon)$$

then with probability at least $1 - \epsilon$ OMP recovers every $f \in \Pi_{I_N}(S)$

Sketch of proof

- fix $T \subset I_N$, $\mathbf{c} \in \mathbb{C}^S$, and choose M sampling nodes independently and uniformly at random on $\mathbb{T}$ or on $\frac{1}{N} I_N$
- for $k \notin T$ and $\delta > 0$ holds

$$\mathbb{P}\left(|\langle \mathbf{A}_T \mathbf{c}, \phi_k \rangle| \geq M\delta\right) \leq 4 \exp\left(-\frac{M\delta^2}{4\|\mathbf{c}\|_2^2 + \frac{4}{3\sqrt{2}}\|\mathbf{c}\|_1 \delta}\right)$$

- Remark: this quantifies the "quadrature rule"

$$\begin{aligned} \langle \mathbf{A}_T \mathbf{c}, \phi_k \rangle = \langle \mathbf{y}, \phi_k \rangle &= \sum_{j=0}^{M-1} f(x_j) \mathrm{e}^{2\pi \mathrm{i} k x_j} \\ &\approx M \cdot \int_{\mathbb{T}} f(x) \mathrm{e}^{2\pi \mathrm{i} k x} \mathrm{d}x = 0 \end{aligned}$$

Compressed Sensing - Thresholding

...

- thresholding recovers the correct support if

$$\min_{j \in T} |\langle \phi_j, \mathbf{A}_T \mathbf{c} \rangle| > \max_{k \notin T} |\langle \phi_k, \mathbf{A}_T \mathbf{c} \rangle|$$

- for $l \in T$, the triangle inequality yields

$$\begin{aligned} |M^{-1} \langle \phi_l, \mathbf{A}_T \mathbf{c} \rangle| &= |c_l + M^{-1} \langle \phi_l, \mathbf{A}_{T \setminus \{l\}} \mathbf{c}_{T \setminus \{l\}} \rangle| \\ &\geq \min_{j \in T} |c_j| - \max_{j \in T} |M^{-1} \langle \phi_j, \mathbf{A}_{T \setminus \{j\}} \mathbf{c}_{T \setminus \{j\}} \rangle| \end{aligned}$$

- hence, thresholding succeeds if

$$\begin{aligned} \max_{j \in T} |M^{-1} \langle \phi_j, \mathbf{A}_{T \setminus \{j\}} \mathbf{c}_{T \setminus \{j\}} \rangle| &< \min_{j \in T} |c_j|/2 \\ \max_{k \notin T} |M^{-1} \langle \phi_k, \mathbf{A}_T \mathbf{c} \rangle| &< \min_{j \in T} |c_j|/2 \end{aligned}$$

Compressed Sensing - Further results

Theorem: [Rauhut] If

$$M \leq C \cdot S^2/\sigma,$$

then with probability exceeding $1 - c_1/S - c_2/\sigma^2$ there exists an $f \in \Pi_{I_N}(S)$ on which tresholding fails. Similar result for OMP with S iterations.

Theorem: [Needell, Vershynin] If

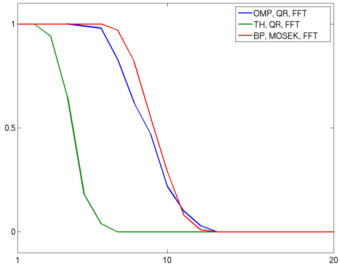
$$M \geq C \cdot S \cdot \log^4(N) \log(1/\epsilon),$$

then with probability at least $1 - \epsilon$ regularised OMP recovers every $f \in \Pi_{I_N}(S)$.

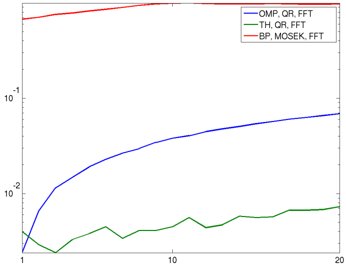
Greedy methods for ℓ^1 minimisation [Donoho, Tsaig]

Compressed Sensing - Numerics

fixed dimension $N = 1000$, fixed
number of samples $M = 40$, normalised Fourier coefficients $|\hat{f}_k| = 1$

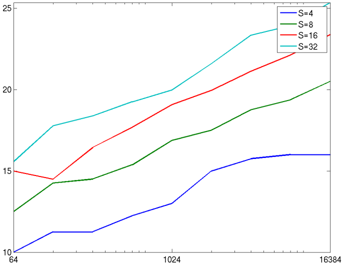


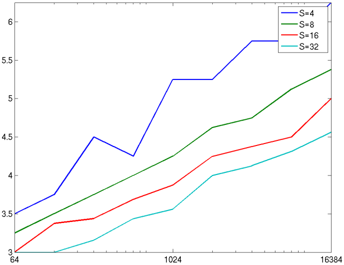
reconstruction rate vs. sparsity S



computation time vs. sparsity S

fixed reconstruction rate 90%, fixed sparsity S





Thresholding ($|\hat{f}_k| = 1$) OMP

generalised oversampling factor M/S vs. dimension N

OMP numerically: $M \approx S \log_2 N$ samples are sufficient

Reconstruction on the Sphere

NFSFT: $\hat{\mathbf{f}} \in \mathbb{C}^{(N+1)^2}$ given, compute

$$\mathbf{f} = \mathbf{Y}\hat{\mathbf{f}}$$

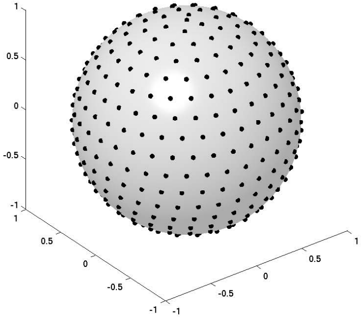
Now: $\mathbf{y} \in \mathbb{C}^M$ given, solve

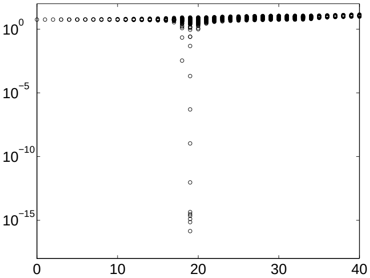
$$\mathbf{Y}\hat{\mathbf{f}} \approx \mathbf{y}$$

Mairhuber-Curtis: $M \geq (N+1)^2$ nodes do not guarantee $\mathrm{rank}\mathbf{Y} = (N+1)^2$

Reconstruction on the Sphere

Singular values of $\mathbf{Y} \in \mathbb{C}^{M\times(N+1)^2}$ $(N = 0, \ldots, 40,\ M = 400)$





Reconstruction on the Sphere

Geodesic distance

$$\operatorname{dist}(\mathbf{x}, \mathbf{y}) = \arccos(\mathbf{y} \cdot \mathbf{x})$$

Mesh norm

$$\delta_{\mathcal{X}} = 2 \max_{\mathbf{x} \in \mathbb{S}^2} \min_{j=0,\ldots,M-1} \operatorname{dist}(\mathbf{x}_j, \mathbf{x})$$

Separation distance

$$q_{\mathcal{X}} = \min_{0 \le j < l < M} \operatorname{dist}(\mathbf{x}_j, \mathbf{x}_l)$$

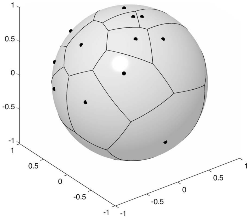
Overdetermined $(N+1)^2 \leq M$, $\mathbf{W} = \operatorname{diag}(w_j) \in \mathbb{R}^{M \times M}$

$$\|\mathbf{Y}\hat{\mathbf{f}} - \mathbf{y}\|_{\mathbf{W}} \to \min$$

$$\Leftrightarrow$$

$$\mathbf{Y}^{\mathsf{H}}\mathbf{W}\mathbf{Y}\hat{\mathbf{f}} = \mathbf{Y}^{\mathsf{H}}\mathbf{W}\mathbf{y}$$

Voronoi region Ω_j and weight $w_j = \mu(\Omega_j)$ to node $\mathbf{x}_j$



Least Squares

Conjugate gradients for least squares (CGNR)

$$\left\| \mathbf{Y}\hat{\mathbf{f}}_l - \mathbf{y} \right\|_{\mathbf{W}} \leq 2 \left(\frac{\sqrt{\Lambda} - \sqrt{\lambda}}{\sqrt{\Lambda} + \sqrt{\lambda}} \right)^l \|\mathbf{y}\|_{\mathbf{W}}$$

Eigenvalues and Marcinkiewicz-Zygmund inequalities

$$[\lambda, \Lambda] = \left\{ \frac{\hat{\mathbf{f}}^{\mathsf{H}}\mathbf{Y}^{\mathsf{H}}\mathbf{W}\mathbf{Y}\hat{\mathbf{f}}}{\hat{\mathbf{f}}^{\mathsf{H}}\hat{\mathbf{f}}} : \hat{\mathbf{f}} \in \mathbb{C}^{(N+1)^2} \right\}$$

Since $\mathbf{f} = \mathbf{Y}\hat{\mathbf{f}}$ and $\|f\|_{L^2} = \|\hat{\mathbf{f}}\|_2$, it remains to prove

$$c_{\mathsf{MZ}} \|f\|_{L^2}^2 \leq \|\mathbf{f}\|_{\mathbf{W}}^2 \leq C_{\mathsf{MZ}} \|f\|_{L^2}^2$$

Lemma: [Jetter, Stöckler, Ward]

$$\left(1-\frac{N\delta}{2}\right)\|f\|_{L^\infty} \leq \max_{0\leq j<M}\left|f\left(\mathbf{x}_j\right)\right| \leq \left(1+\frac{N\delta}{2}\right)\|f\|_{L^\infty}.$$

Proof: Markov inequality (consider the great arc through $\mathbf{x},\mathbf{y}$)

$$|f(\mathbf{x})-f(\mathbf{y})| \leq \operatorname{dist}(\mathbf{x},\mathbf{y})\cdot N\|f\|_{L^\infty}, \qquad \mathbf{x},\mathbf{y}\in\mathbb{S}^2.$$

For arbitrary $\mathbf{x}\in\mathbb{S}^2$ and its closest sampling node with index $j=\arg\min_{0\leq l<M}\operatorname{dist}(\mathbf{x},\mathbf{x}_l)$

$$|f(\mathbf{x})| \leq |f(\mathbf{x})-f(\mathbf{x}_j)|+|f(\mathbf{x}_j)| \leq \frac{N\delta}{2}\|f\|_{L^\infty}+\max_{0\leq j<M}|f(\mathbf{x}_j)|.$$

Taking the maximum over $\mathbf{x}\in\mathbb{S}^2$ yields the assertion.

Lemma: [Mhaskar, Narcowich, Ward; Filbir, Themistoclakis]

$$(1 - 153N\delta)\,\|f\|_{L^1} \leq \sum_{j=0}^{M-1} w_j \left|f\left(\mathbf{x}_j\right)\right| \leq (1 + 153N\delta)\,\|f\|_{L^1}\,.$$

Proof: use a "nice" reproducing kernel v_N for $\Pi_N(\mathbb{S}^2)$

$$\left|\|f\|_{L^1} - \sum_{j=0}^{M-1} w_j \left|f\left(\mathbf{x}_j\right)\right|\right| = \sum_{j=0}^{M-1} \int_{\Omega_j} \left|f\left(\mathbf{x}\right) - f\left(\mathbf{x}_j\right)\right| \mathrm{d}\mu\left(\mathbf{x}\right)$$

$$= \sum_{j=0}^{M-1} \int_{\Omega_j} \left|\int_{\mathbb{S}^2} \left(v_N\left(\mathbf{x}\cdot\mathbf{y}\right) - v_N\left(\mathbf{x}_j\cdot\mathbf{y}\right)\right) f\left(\mathbf{y}\right) \mathrm{d}\mu\left(\mathbf{y}\right)\right| \mathrm{d}\mu\left(\mathbf{x}\right)$$

$$\leq \sum_{j=0}^{M-1} \int_{\Omega_j} \int_{\mathbb{S}^2} \left|v_N\left(\mathbf{x}\cdot\mathbf{y}\right) - v_N\left(\mathbf{x}_j\cdot\mathbf{y}\right)\right| \left|f\left(\mathbf{y}\right)\right| \mathrm{d}\mu\left(\mathbf{y}\right) \mathrm{d}\mu\left(\mathbf{x}\right)$$

$$\leq \|f\|_{L^1} \sup_{\mathbf{y}\in\mathbb{S}^2} \sum_{j=0}^{M-1} \int_{\Omega_j} \left|v_N\left(\mathbf{x}\cdot\mathbf{y}\right) - v_N\left(\mathbf{x}_j\cdot\mathbf{y}\right)\right| \mathrm{d}\mu\left(\mathbf{x}\right).$$

Least Squares

Theorem: [Mhaskar, Narcowich, Ward; Filbir, Themistoclakis; Keiner, Potts, K.]
Let the sampling set $\mathcal{X} \subset \mathbb{S}^2$ of cardinality $M \in \mathbb{N}$ with mesh norm δ and Voronoi weights w_j, $\mathbf{W} = \mathrm{diag}(w_j)_{j=0,\ldots,M-1}$ be given. Polynomial degree $N \in \mathbb{N}$ with

$$N < \frac{1}{154\delta}.$$

$f \in \Pi_N(\mathbb{S}^2)$ and its samples $\mathbf{f} = \left(f(\mathbf{x}_j)\right)_{j=0,\ldots,M-1}$ fulfil the norm estimate

$$(1 - 154N\delta)\,\|f\|_{L^2}^2 \leq \|\mathbf{f}\|_{\mathbf{W}}^2 \leq (1 + 154N\delta)\,\|f\|_{L^2}^2\,.$$

Least Squares

Proof: Riesz-Thorin interpolation theorem

$$
\begin{aligned}
&\sup_{f \in \Pi_N(\mathbb{S}^2),\, f \neq 0} \sum_{j=0}^{M-1} w_j \left| f\left(\mathbf{x}_j\right) \right|^2 \Big/ \|f\|_{L^2}^2 \\
\leq &\sup_{f \in \Pi_N(\mathbb{S}^2),\, f \neq 0} \sum_{j=0}^{M-1} w_j \left| f\left(\mathbf{x}_j\right) \right| \Big/ \|f\|_{L^1} \\
\times &\sup_{f \in \Pi_N(\mathbb{S}^2),\, f \neq 0} \max_{0 \leq j < M} \left| f\left(\mathbf{x}_j\right) \right| \Big/ \|f\|_{L^\infty}.
\end{aligned}
$$

Remark:

- the condition $N < \frac{c}{\delta}$ is optimal
- for $\mathbb{T}$ the sharp condition $N < 1/\delta$ suffices, uses a Wirtinger and a Bernstein inequality [Gröchenig]

Optimal Interpolation

Underdet. $(N+1)^2 \geq M$, $\hat{\mathbf{W}} = \operatorname{diag}(\tilde{w}_{\mathbf{k}}) \in \mathbb{R}^{(N+1)^2 \times (N+1)^2}$

$$\|\hat{\mathbf{f}}\|^2_{\hat{\mathbf{W}}^{-1}} = \sum_{\mathbf{k} \in J_N} \frac{|\hat{f}_{\mathbf{k}}|^2}{\tilde{w}_{\mathbf{k}}} \to \min \quad s.t. \quad \mathbf{Y}\hat{\mathbf{f}} = \mathbf{y}$$

$$\Leftrightarrow$$

$$\mathbf{Y}\hat{\mathbf{W}}\mathbf{Y}^{\mathsf{H}}\tilde{\mathbf{f}} = \mathbf{y}, \qquad \hat{\mathbf{f}} = \hat{\mathbf{W}}\mathbf{Y}^{\mathsf{H}}\tilde{\mathbf{f}}$$

Polynomial kernel $K_N : [-1,1] \to \mathbb{R}$

$$K_N(t) = \sum_{k=0}^{N} \frac{2k+1}{4\pi} \hat{w}_k P_k(t)$$

and its associated matrix, $\tilde{w}_{\mathbf{k}} = \tilde{w}_k^n = \hat{w}_k$

$$\mathbf{Y}\hat{\mathbf{W}}\mathbf{Y}^{\mathsf{H}} = \mathbf{K} = \left(K_N\left(\mathbf{x}_j \cdot \mathbf{x}_l\right)\right)_{j,l=0,\ldots,M-1}$$

Conjugate gradients for optimal interpolation (CGNE)

$$\left\| \hat{\mathbf{f}}_l - \hat{\mathbf{W}} \mathbf{Y}^{\mathsf{H}} \mathbf{K}^{-1} \mathbf{y} \right\|_{\hat{\mathbf{W}}^{-1}} \leq \frac{2}{\sqrt{\lambda}} \left(\frac{\sqrt{\Lambda} - \sqrt{\lambda}}{\sqrt{\Lambda} + \sqrt{\lambda}} \right)^l \|\mathbf{y}\|_2$$

Gershgorin circle theorem (off diagonal decay of $\mathbf{K}$)

$$\left|\lambda_\star - K_N\left(\mathbf{x}_{j_\star} \cdot \mathbf{x}_{j_\star}\right)\right| \leq \sum_{l=0, l \neq j_\star}^{M-1} \left|K_N\left(\mathbf{x}_{j_\star} \cdot \mathbf{x}_l\right)\right|$$

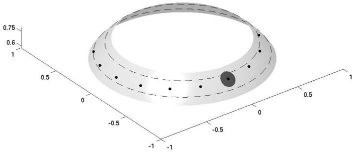
We need: packing argument on $\mathbb{S}^2$ and localisation of K_N.

Packing argument

- for separation distance $q \leq \pi$ and $0 \leq m < \lfloor \pi q^{-1} \rfloor$ define

$$S_{q,m} = \left\{ \mathbf{x} \in \mathbb{S}^2 : mq \leq \arccos\left(\mathbf{x}_0 \cdot \mathbf{x}\right) < (m+1)\, q \right\},$$
$$S_{q,\lfloor \pi q^{-1} \rfloor} = \left\{ \mathbf{x} \in \mathbb{S}^2 : \left\lfloor \pi q^{-1} \right\rfloor q \leq \arccos\left(\mathbf{x}_0 \cdot \mathbf{x}\right) \leq \pi \right\}$$

- restrictions to the set of sampling nodes $S_{\mathcal{X},q,m} = S_{q,m} \cap \mathcal{X}$
- the set $S_{\mathcal{X},q,m}$, the ring $S_{q,m}$ (dashed), the larger ring $\tilde{S}_{q,m}$, and a spherical cap of colatitude $q/2$ centred at one node



Lemma: If $\mathcal{X} \subset \mathbb{S}^2$ is q-separated, then

$$|S_{\mathcal{X},q,m}| \le 25m, \qquad m = 1, \ldots, \lfloor \pi q^{-1} \rfloor.$$

Proof: [Narcowich, Sivakumar, Ward; Prestin, Selig]
for $m = 1, \ldots, \lfloor \pi q^{-1} \rfloor - 2$

$$\begin{aligned} |S_{\mathcal{X},q,m}| &\le \frac{\int_{\left(m-\frac{1}{2}\right)q}^{\left(m+\frac{3}{2}\right)q} \sin\theta \mathrm{d}\theta}{\int_0^{\frac{q}{2}} \sin\theta \mathrm{d}\theta} \\ &= \frac{\sin\left((2m+1)\frac{q}{2}\right)\sin\left(2\frac{q}{2}\right)}{\sin^2\frac{q}{4}} \\ &\le 8\left(2m+1\right). \end{aligned}$$

Localised kernels

- normalised B-spline of order $\beta \in \mathbb{N}$, $g_\beta : [-\frac{1}{2}, \frac{1}{2}] \to \mathbb{R}$

$$g_\beta(z) = \beta N_\beta(\beta z + \frac{\beta}{2})$$

$$N_{\beta+1}(z) = \int_{z-1}^{z} N_\beta(\tau)\,\mathrm{d}\tau, \qquad N_1(z) = \begin{cases} 1 & 0 < z < 1, \\ 0 & \text{otherwise} \end{cases}$$

- B-spline kernel $B_{\beta,N} : [-1, 1] \to \mathbb{R}$, $N \in \mathbb{N}$

$$B_{\beta,N}(t) = \frac{1}{\|g_\beta\|_{1,N}} \sum_{l=0}^{N} (2 - \delta_{l,0})\, g_\beta\left(\frac{l}{2(N+1)}\right) T_l(t),$$

$$\|g_\beta\|_{1,N} = \sum_{l=-N}^{N} g_\beta\left(\frac{l}{2(N+1)}\right)$$

Lemma: The B-spline kernel $B_{\beta,N}$ obeys $B_{\beta,N}(1) = 1$ and for $N \geq \beta - 1$, $t \in [-1,1)$ the localisation

$$|B_{\beta,N}(t)| \leq \underbrace{\frac{\left(2^{\beta}-1\right)\zeta(\beta)\,\beta^{\beta}}{2^{\beta-1}-\zeta(\beta)\,\pi^{-\beta}}}_{=c_\beta} \cdot |(N+1)\arccos(t)|^{-\beta}$$

Moreover, the following representation with **positive** Fourier-Legendre coefficients $\hat{w}_k$ holds

$$B_{\beta,N}(t) = \sum_{k=0}^{N} \frac{2k+1}{4\pi} \hat{w}_k P_k(t)\,.$$

Proof: Poisson summation formula, integration by parts; explicit calculation of connection coefficients.

Theorem: Sampling set $\mathcal{X} \subset \mathbb{S}^2$ of cardinality $M \in \mathbb{N}$ with separation distance q, weights $\hat{w}_k$ set to the Fourier-Legendre coefficients of the β-th B-spline kernel, polynomial degree $N \in \mathbb{N}$, $N \geq \beta - 1 \geq 2$ and

$$N + 1 > \frac{\left(25 c_\beta \zeta\left(\beta - 1\right)\right)^{1/\beta}}{q}.$$

Then the kernel matrix obeys

$$|\lambda_\star - 1| \leq \frac{25 c_\beta \zeta\left(\beta - 1\right)}{\left(\left(N+1\right) q\right)^\beta}.$$

Optimal Interpolation

Proof:

$$
\begin{aligned}
|\lambda_\star - 1| &\leq \sum_{l=0, l\neq j_\star}^{M-1} \left| K_N\left(\mathbf{x}_{j_\star} \cdot \mathbf{x}_l\right)\right| \\
&\leq \sum_{m=1}^{\lfloor \pi q^{-1} \rfloor} |S_{\mathcal{X},q,m}| \max_{\mathbf{x}\in S_{q,m}} \cdot \left| B_{\beta,N}\left(\mathbf{x}_{j_\star} \cdot \mathbf{x}\right)\right| \\
&\leq \sum_{m=1}^{\lfloor \pi q^{-1} \rfloor} 25m \cdot c_\beta \left|(N+1)mq\right|^{-\beta}.
\end{aligned}
$$

Remark:

- the condition $N > \frac{C}{q}$ is optimal
- for $\mathbb{T}$ the condition $N > \frac{\pi}{\sqrt{3}q}$ suffices $(\beta = 2)$
- generalised for $\mathbb{S}^{d-1}$, where $N + 1 > \frac{5\pi d}{2q}$ suffices $(\beta = d)$
- similar results for $\mathbb{T}^d$ and $SO(3)$

Reconstruction on the Sphere

Least squares approximation stable for

$$N < \frac{c}{\delta}$$

[$\mathbb{T}^d$ Gröchenig; $\mathbb{S}^d$ Mhaskar, Narcowich, Ward; Filbir, Themistoclakis; $\mathbb{S}^2$ Keiner, K., Potts; $SO(3)$ Schmid]

Optimal interpolation stable for

$$N > \frac{C}{q}$$

[$\mathbb{T}^d$ K., Potts; $\mathbb{S}^2$ Keiner, K., Potts; $\mathbb{S}^d$ K., $SO(3)$ Gräf, K.]

"Curse of dimensions": with spatial dimension $d \to \infty$

$$c \to 0, \qquad C \to \infty.$$

Summary

Fast approximative algorithms

- Fourier transforms
- polynomial transforms
- convolutions
- iterative reconstruction
- numerical harmonic analysis on $\mathbb{T}^d$, $\mathbb{S}^2$, $SO(3)$

for n degrees of freedom spend

$$\mathcal{O}(n \log^\alpha n |\log \varepsilon|^\beta)$$

flops

Software and papers on

http://www-user.tu-chemnitz.de/~potts/nfft