

CS601: Software Development for Scientific Computing

Autumn 2023

Week14 : FEM and Program Representation
(Grids)

Program Representation – Structured Grids

- Grid requirements:
 - Grid dimension shall not be hardcoded
 - Consequence: implementations must define a compile-time constant
 - Grid step size shall not be hardcoded E.g. $h=1/3$, $h=1/5$ etc.
 - Consequence: can't define `int arr[m][n]; //m,n to be constant expr.`
 - A grid point shall be identified with cartesian coordinates / polar coordinates (e.g. with angle and radius from origin)
 - Shall be able to generate a structured grid given number of points, ξ , and η .
 - Shall allow access to any grid point
 - Shall allow for implementation of grid operators

Structured Grids - Representation

- Because of regular connectivity between cells
 - Cells can be identified with indices (x,y) or (x,y,z) and neighboring cell info can be obtained.
 - How about identifying a cell here?

Given:

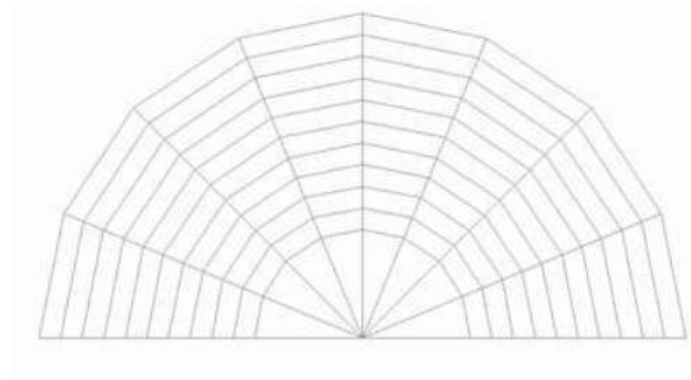
ξ = (“Xi”) radius

η = (“Eta”) angle

Compute:

$$x = \left(\frac{1}{2} + \xi \right) \cos(\pi\eta)$$

$$y = \left(\frac{1}{2} + \xi \right) \sin(\pi\eta)$$



class Domain

- We discretize the domain using a grid

```
class Domain{
    public:
        generate_grid(int m, int n);
        Domain(); // constructor
        //...
    private:
        //...
};
```

Method GenerateGrid

- What is the shortcoming of the following method?

```
void Domain::GenerateGrid(int m, int n) {
    if (m <=0 || n<=0)
        throw std::invalid_argument("ERR_generate_grid");
    else if( (xlen > 0) || (ylen > 0)) {
//there already exists a grid! Attempt to create a grid again
        delete [] x; delete [] y;
    }
    xlen=m;ylen=n; // initialize members
    x=new double[xlen*ylen]; y=new double[xlen*ylen];
}
```

- Assumes a 2D grid.

Grid Function

- We let a grid function to operate on the grid points
 - Example of an operator: numerical differentiation
 - Different operations possible
 - Note: grid function always operates on some grid.
 - Many functions may operate on the same grid.

```
class GridFn{
    public:
        //...
    private:
        Domain* d; //denotes aggregation relationship
        //...
};
```

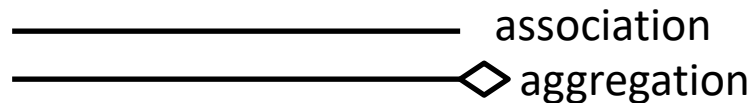
Detour: Relationships among Classes

- Dependencies (“uses”)

E.g. Customer uses a MS Word editor to produce MS Word document



- Association / Aggregation (“has a”)



E.g. Every course has a name, credits - aggregation
A student registers for course(s) – association
between student and course

- Generalization (“is a”)



E.g. Apple is a Fruit (*Apple and Fruit are modeled as classes, where Fruit is a super-class and Apple is a sub-class*)

Boundary conditions

- Multiple options: affect the accuracy of the solution

Name	Prescription	Interpretation
Dirichlet (essential)	u	Fixed temperature
Neumann (Natural)	$\partial u / \partial n$	Energy Flow
Robin (Mixed)	$\partial u / \partial n + f(u)$	Temperature dependent flow

- How to represent boundary conditions?
 - Create a separate Solution class

Solution

- pseudo-code

```
1 Domain dom; // create domain
2 GridFn g(dom); //create grid function to operate on a domain
3 Solution u(g) //prepare to compute a solution:
4 u.initcond() //1) set initial conditions
5 for(int step=0; step<maxsteps; step++) 2) iterate:
6 {
7     u.compute(); //2) compute solution repeatedly
8 }
```

`u.iterate()` or `u.solve()`

class Solution

- We discretize the domain using a grid

```
class Solution{
    public:
        Solution(GridFn* d): sol(d) {}
        initcond();
        boundarycond();
        //... other member functions?
    private:
        GridFn* sol;
};
```

What is missing?


- Data array?
 - We need to make provision for storing the results of algebraic equations (temperature, displacements, stress, strain etc.)
- Type of data as template parameter?
 - Does the application accept single-precision results?
Double-precision results?
- Operation on subgrids (Box)?
 - When a particular grid function is applied only in a certain region

Matrix Algebra and Efficient Computation

- Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)

<i>Motif</i>	Embed	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser	<i>Motif</i>	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser
	1 Finite State Mach.	Hot	Hot	Med	Hot	Med							Hot	9 N-Body	Med			Hot				
2 Combinational	Hot			Med	Med						Hot	10 MapReduce	Med		Hot	Hot	Hot			Med	Hot	Med
3 Graph Traversal	Hot	Hot	Hot	Hot	Hot		Hot		Hot		Med	11 Backtrack/B&B		Hot	Hot				Hot			Hot
4 Structured Grid	Hot	Hot	Hot	Hot	Hot	Hot	Hot		Hot	Hot		12 Graphical Models		Hot	Hot				Hot			
5 Dense Matrix	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot		13 Unstructured Grid	Hot	Hot	Hot	Hot	Hot	Hot				
6 Sparse Matrix	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	<i>Temperature Chart of Need</i>					DB = database					
7 Spectral (FFT)	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Warm	Warm	Med	Cool	ML = machine learning					
8 Dynamic Prog	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	HPC = High Perf. Comp.				

Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

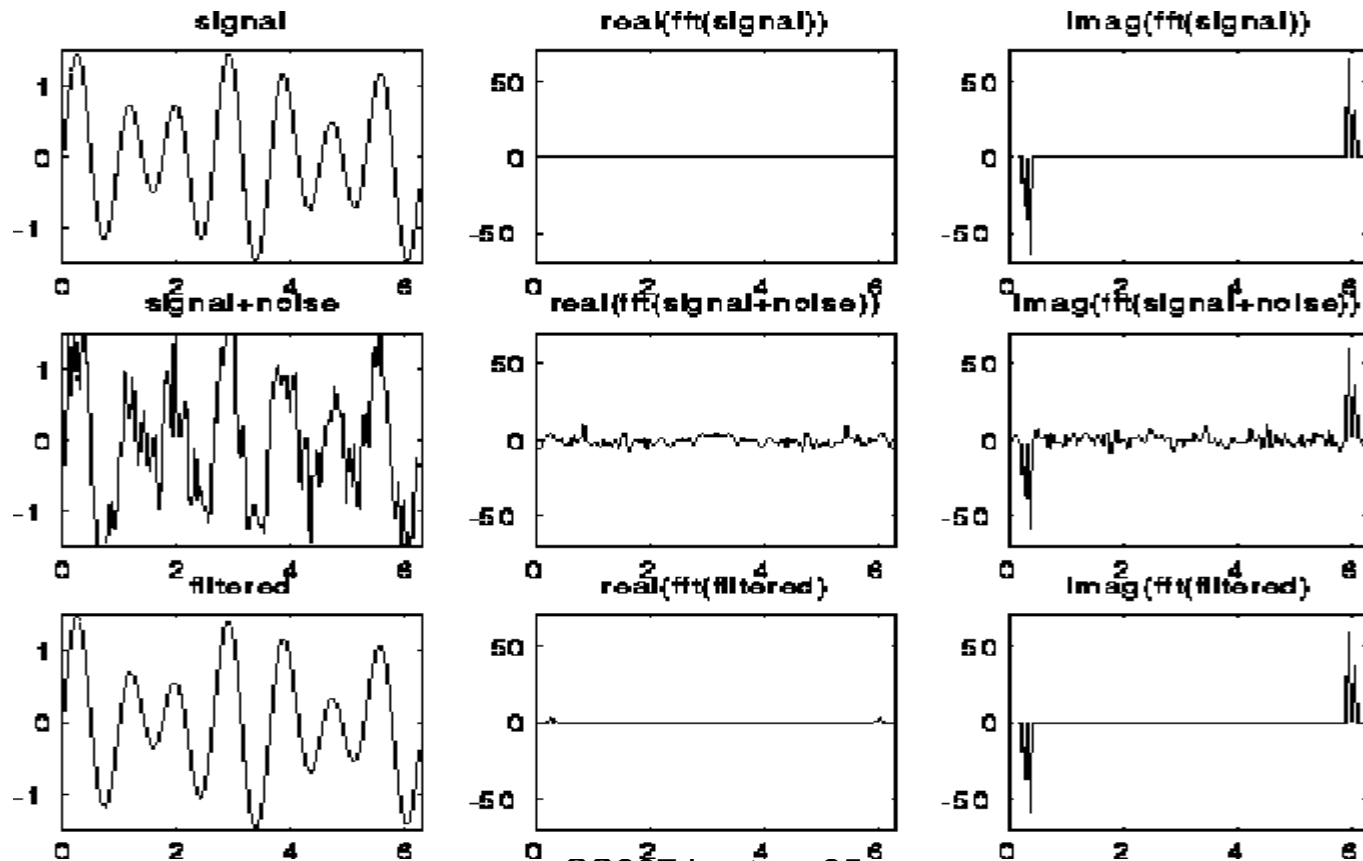
⇒ Seen earlier
 Next..

Faster $y=Ax$: Discrete Fourier Transforms (DFT)

- Very widely used
 - Image compression (jpeg)
 - Signal processing
 - Solving Poisson's Equation
- Represent A with F , a *Fourier Matrix* that has the following (remarkable) properties:
 - F^{-1} is easy to compute
 - Multiplications by F and F^{-1} is fast. (need to do $Fx=y$ and $x= F^{-1} y$ quickly)
- F has complex numbers in its entries.
 - Every entry is a power of a single number w such that $w^n=1$
 - Any entry of a Fourier matrix can be written using $f_{ij} = w^{ij}$ (row and col indices start from 0)

Using the 1D FFT for filtering

- Signal = $\sin(7t) + .5 \sin(5t)$ at 128 points
- Noise = random number bounded by .75
- Filter by zeroing out FFT components $< .25$



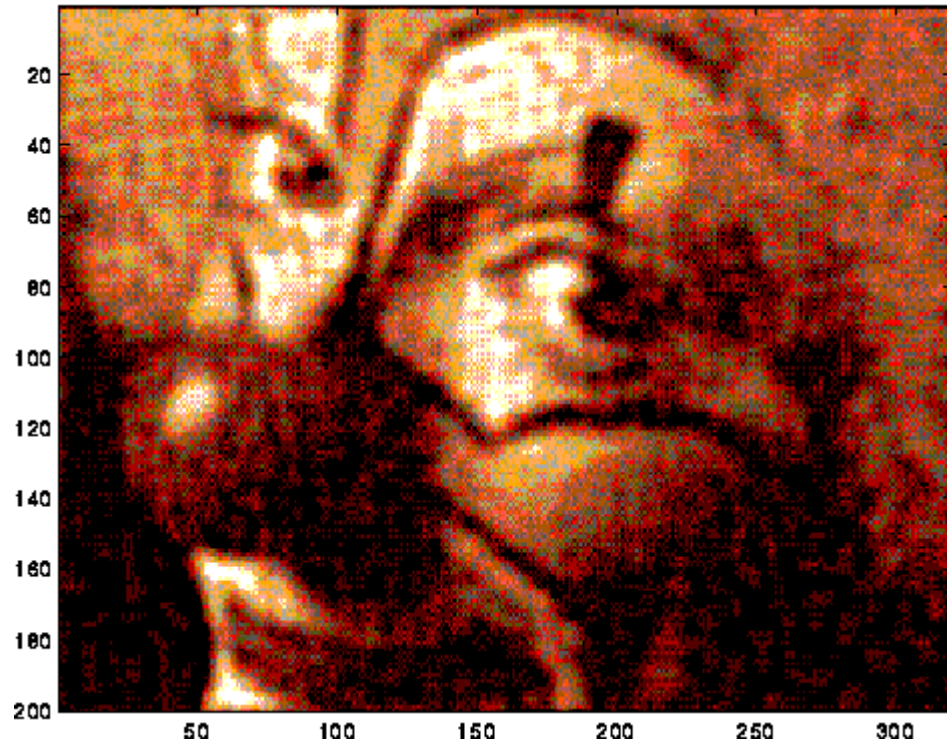
Using the 2D FFT for image compression

- Image = 200x320 matrix of values
- Compress by keeping largest 2.5% of FFT components
- Similar idea used by jpeg

Original Image



Keep only largest 2.5% of entries of 2DFFT



$Y = \text{fft}(X)$ in MATLAB

- X input vector
 - size=?
- Y output vector

$$Y(k) = \sum_{j=1}^n X(j) w_n^{(j-1)(k-1)}$$

where

$$w_n = e^{-\frac{2\pi i}{n}}$$

Examples: Fourier Matrix

- $$4 \times 4: F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & 1 & w^2 \\ 1 & w^3 & w^2 & w^1 \end{bmatrix}, i = \sqrt{-1}$$

– Here, $w = i$ (also denoted as $w_4 = i$). $w^4 = 1 \Rightarrow i$ is a root.

- $$8 \times 8: F_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} & w^{14} \\ 1 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} & w^{21} \\ 1 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} & w^{28} \\ 1 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} & w^{35} \\ 1 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} & w^{42} \\ 1 & w^7 & w^{14} & w^{21} & w^{28} & w^{35} & w^{42} & w^{49} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix}$$

Here, $w = \frac{1+i}{\sqrt{2}}$
(= sqrt of i)

Example: Faster $y=Fx$

Column: 1 2 3 4 5 6 7 8

$$\begin{bmatrix}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\
 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\
 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\
 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\
 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\
 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\
 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1
 \end{bmatrix}
 =
 \begin{bmatrix}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\
 1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\
 1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
 1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\
 1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\
 1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5
 \end{bmatrix}$$

(Writing columns 1,3,5,7 first and then columns 2,4,6,8)

Example: Faster $y=Fx$

$$\bullet \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5 \end{array} \right]$$



(Partitioning into 4 matrix blocks of size 4x4.)

Recall: $F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix}$, where $w = i = w_4$

Note: in F_8 , $w = \frac{1+i}{\sqrt{2}} = w_8$
 therefore, $w_8^2 = w_4$

Example: Faster $y=Fx$

- $$\begin{bmatrix}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\
 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\
 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\
 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\
 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\
 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\
 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1
 \end{bmatrix} = \begin{array}{c|c}
 F_4 & \Omega_4 F_4 \\
 \hline
 F_4 & -\Omega_4 F_4
 \end{array}$$

\uparrow
 (because $w^2 = w_4$)

$$\Omega_4 = \begin{bmatrix}
 1 & 0 & 0 & 0 \\
 0 & w & 0 & 0 \\
 0 & 0 & w^2 & 0 \\
 0 & 0 & 0 & w^3
 \end{bmatrix} \text{ (note: } w = \frac{1+i}{\sqrt{2}} = w_8)$$
- So, $F_8 = \begin{bmatrix} F_4 & \Omega_4 F_4 \\ F_4 & -\Omega_4 F_4 \end{bmatrix}$

FFT

We can obtain 8-point DFT from 4-point DFT. But how do we obtain the result of $y = F_8 x$, from $y_{\text{top}} = F_4 x_{\text{odd}}$ and $y_{\text{bottom}} = F_4 x_{\text{even}}$?

$$\begin{array}{c}
 \mathbf{F}_8 \\
 \left[\begin{array}{c|c}
 \mathbf{F}_4 & \Omega_4 \mathbf{F}_4 \\
 \hline
 \mathbf{F}_4 & -\Omega_4 \mathbf{F}_4
 \end{array} \right]
 \end{array}
 \begin{array}{c}
 \mathbf{x} \\
 \left[\begin{array}{c}
 x1 \\
 x3 \\
 x5 \\
 x7 \\
 \hline
 x2 \\
 x4 \\
 x6 \\
 x8
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 \left[\begin{array}{c|c}
 \mathbf{I}_4 & \Omega_4 \\
 \hline
 \mathbf{I}_4 & -\Omega_4
 \end{array} \right]
 \end{array}
 \begin{array}{c}
 \left[\begin{array}{c}
 \mathbf{F}_4 \\
 \hline
 \mathbf{F}_4
 \end{array} \right]
 \begin{array}{c}
 \left[\begin{array}{c}
 x1 \\
 x3 \\
 x5 \\
 x7 \\
 \hline
 x2 \\
 x4 \\
 x6 \\
 x8
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 \mathbf{y} \\
 \left[\begin{array}{c}
 y1 \\
 y3 \\
 y5 \\
 y7 \\
 \hline
 y2 \\
 y4 \\
 y6 \\
 y8
 \end{array} \right]
 \end{array}$$

Note: can be done with 4 multiplications

$$y1 \text{ to } y4 = y_{\text{top}} + \Omega_4 * y_{\text{bottom}}$$

$$y5 \text{ to } y8 = y_{\text{top}} - \Omega_4 * y_{\text{bottom}}$$

$$y_{\text{top}} = F_4 x_{\text{odd}}$$

$$y_{\text{bottom}} = F_4 x_{\text{even}}$$

(x_{odd} = elements at odd numbered indices of vector x)

(x_{even} = elements at even numbered indices of vector x)

Divide-and-Conquer FFT (D&C FFT)

FFT(v, ω , m) ... assume m is a power of 2

if m = 1 return v[0]

else

$$V_{\text{even}} = \text{FFT}(v[0:2:m-2], \omega^2, m/2)$$

$$V_{\text{odd}} = \text{FFT}(v[1:2:m-1], \omega^2, m/2)$$

$$\omega\text{-vec} = [\omega^0, \omega^1, \dots, \omega^{(m/2-1)}]$$

precomputed



$$\text{return } [V_{\text{even}} + (\omega\text{-vec} .* V_{\text{odd}}),$$

$$V_{\text{even}} - (\omega\text{-vec} .* V_{\text{odd}})]$$

° Matlab notation: “.*” means component-wise multiply.


Cost: $T(m) = 2T(m/2) + O(m) = O(m \log m)$ operations.

Popularized/published by Cooley-Tuckey in 1965.


FFT - Summary

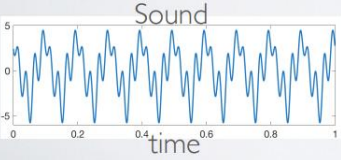
- We will revisit FFT when solving Poisson's equation
- 2-slide summary (**courtesy**: Alex Townsend, Cornell. [Source](#))

1965: THE FAST FOURIER TRANSFORM



"Mozart could listen to music just once and then write it down from memory without any mistakes" [Vernon, 1996]

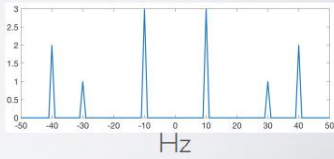
A simple example:  sound



Sound

time

FFT



|Frequencies|

Hz

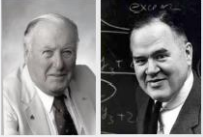
sound(t) = $3 \cos(2\pi 10t + 0.2) + \cos(2\pi 30t - 0.3) + 2 \cos(2\pi 40t + 2.4)$

HOW DOES IT WORK?

Given equally spaced samples $f(0/n), f(1/n), \dots, f((n-1)/n)$, find a_k so that

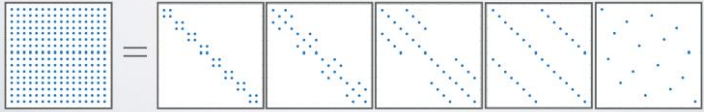
$$f(j/n) = \sum_{k=-n/2}^{n/2-1} a_k e^{2\pi i k(j/n)}, \quad 0 \leq j \leq n-1.$$

Fourier series

$$\begin{pmatrix} f(0/n) \\ \vdots \\ f((n-1)/n) \end{pmatrix} = F \begin{pmatrix} a_{-n/2} \\ \vdots \\ a_{n/2-1} \end{pmatrix}, \quad F_{jk} = e^{2\pi i k(j/n)}$$


Cooley Tukey

F has a sparse factorization. For $n = 16$ we have



- References:
 - Refer to Lecture 20 (Spring 2018) at <https://inst.eecs.berkeley.edu/~cs267/archives.html>
 - Section 1.4, Matrix Computations, 4th Ed, Golub and Van Loan
 - Section 3.5, Linear Algebra and Its Applications, 4th Ed, Gilbert Strang