CS 450: Numerical Anlaysis[1]

Fast Fourier Transform

University of Illinois at Urbana-Champaign

[1] *These slides have been drafted by Edgar Solomonik as lecture templates and supplementary material for the book "Scientific Computing: An Introductory Survey" by Michael T. Heath (slides).*

Sparse Linear Systems and Time-independent PDEs

- The Poisson equation serves as a model problem for numerical methods:
  - *the 2D Poisson problem and resulting Kronecker product linear system are a common benchmark,*
  - *this system has the form* $\boldsymbol{T} \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{T}$ *where* $\boldsymbol{T}$ *is tridiagonal.*
- Dense, sparse direct, iterative, FFT, and Multigrid methods provide increasingly good complexity for the problem:
  - *dense linear system solve costs* $O(n^3)$ *naively,*
  - *nested dissection with Cholesky has* $O(n^{3/2})$ *complexity and* $O(n \log n)$ *memory*
  - *Conjugate-Gradient gives* $O(n^{3/2})$ *complexity with* $O(n)$ *memory*
  - *FFT achieves* $O(n \log n)$ *cost and multigrid achieves* $O(n)$*.*

Multigrid

- Multigrid employs a hierarchy of grids to accelerate iterative methods:
  - *the residual equation $\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{r}$ on each fine grid, is approximately solved on the next coarser grid,*
  - *the equation is restricted by projection matrix $\boldsymbol{P}$, so that $\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T\boldsymbol{P}\hat{\boldsymbol{x}} = \boldsymbol{P}\boldsymbol{r}$*
  - *the interpolation operator (often given by $\boldsymbol{P}^T$) is used to obtain an approximate $\hat{\boldsymbol{x}}$ based on the coarse grid approximate solution,*
  - *at each level we perform some smoothing operations (e.g. Jacobi or Conjugate Gradient) before restriction and after interpolation,*
  - *at the coarsest level we typically solve directly.*

- The multigrid method works by resolving high-frequency error components on finer-grids and low-frequency error components on coarser grids:
  - *smoothers are usually effective at reducing local error, but slow at resolving global (low-frequency) components of the error,*
  - *on coarser grids, the low frequency error may be resolved more quickly.*

Multigrid

- Consider the Galerkin approximation with linear finite elements to the Poisson equation $u'' = f(t)$ with boundary conditions $u(a) = u(b) = 0$:

$$\phi_i^{(h)}(t) = \begin{cases} (t - t_{i-1})/h & : t \in [t_{i-1}, t_i] \\ (t_{i+1} - t)/h & : t \in [t_i, t_{i+1}] \\ 0 & : \text{otherwise} \end{cases}$$

where $t_0 = t_1 = a$ and $t_{n+1} = t_n = b$. *The weak form with grid spacing of h is*

$$\int_a^b f(t)\phi_i^{(h)}(t)dt = -\sum_{j=1}^{n} x_j \int_a^b \phi_j^{(h)\prime}(t)\phi_i^{(h)\prime}(t)dt.$$

in multigrid, we define a coarse grid basis of $(n-1)/2$ functions, which are hat functions of twice the width,

$$\phi_i^{(2h)}(t) = \frac{1}{2}\phi_{2i-2}^{(h)}(t) + \phi_{2i-1}^{(h)}(t) + \frac{1}{2}\phi_{2i}^{(h)}(t) = \begin{cases} (t - t_{i-2})/2h & : t \in [t_{i-2}, t_i] \\ (t_{i+2} - t)/2h & : t \in [t_i, t_{i+2}] \\ 0 & : \textit{otherwise} \end{cases}$$

Coarse Grid Matrix

- Multigrid restricts the residual equation on the fine grid $\boldsymbol{A}^{(h)}\boldsymbol{x} = \boldsymbol{r}^{(h)}$ to the coarse grid: *Let* $\boldsymbol{\phi}^{(2h)} = \begin{bmatrix}\phi_1^{(2h)} & \cdots & \phi_{(n-1)/2}^{(2h)}\end{bmatrix}$ *and* $\boldsymbol{\phi}^{(h)} = \begin{bmatrix}\phi_1^{(h)} & \cdots & \phi_n^{(h)}\end{bmatrix}$ *and define* *restriction matrix* $\boldsymbol{P}$ *so that* $\boldsymbol{\phi}^{(2h)} = \boldsymbol{P}\boldsymbol{\phi}^{(h)}$, *i.e.,*

$$\boldsymbol{P} = \frac{1}{2}\begin{bmatrix} 1 & 2 & 1 & & & \\ & 1 & 2 & 1 & & \\ & & \ddots & \ddots & \ddots \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}^{(1)} \\ \boldsymbol{p}^{(2)} \\ \vdots \end{bmatrix}.$$

The coarse grid stiffness matrix is given by

$$\begin{aligned} a_{ij}^{(2h)} &= -\int_a^b \phi_j^{(2h)\prime}(t)\phi_i^{(2h)\prime}(t)dt \\ &= -\boldsymbol{p}^{(i)} \underbrace{\left(\int_a^b \boldsymbol{\phi}^{(h)\prime}(t)\boldsymbol{\phi}^{(h)\prime T}(t)dt\right)}_{-\boldsymbol{A}^{(h)}} \boldsymbol{p}^{(j)T}, \end{aligned}$$

$$\boldsymbol{A}^{(2h)} = \boldsymbol{P}\boldsymbol{A}^{(h)}\boldsymbol{P}^T.$$

Restricting the Residual Equation

- Given the fine-grid residual $\boldsymbol{r}^{(h)}$, we seek to use the coarse grid to approximate $\boldsymbol{x}^{(h)}$ so that $\boldsymbol{A}\boldsymbol{x}^{(h)} \approx \boldsymbol{r}^{(h)}$
  - *Given a function in the coarse grid basis, $u^{(2h)} = {\boldsymbol{x}^{(2h)}}^T \boldsymbol{\phi}^{(2h)}$, we can express it in the fine-grid basis via*

$$u^{(2h)} = {\boldsymbol{x}^{(2h)}}^T \underbrace{\boldsymbol{P}\boldsymbol{\phi}^{(h)}}_{\boldsymbol{\phi}^{(2h)}} = \underbrace{{\boldsymbol{x}^{(2h)}}^T \boldsymbol{P}}_{{\boldsymbol{x}^{(h)}}^T} \boldsymbol{\phi}^{(h)}.$$

  - *Consequently, the solution to the restricted residual equation $\boldsymbol{A}^{(2h)}\boldsymbol{x}^{(2h)} = \boldsymbol{r}^{(2h)}$ will lead to an approximate residual equation solution on the fine grid with $\boldsymbol{x}^{(h)} = \boldsymbol{P}^T\boldsymbol{x}^{(2h)}$.*
  - *Noting this, we derive the form of the coarse grid residual,*

$$\begin{aligned} \boldsymbol{r}^{(2h)} &= \boldsymbol{A}^{(2h)}\boldsymbol{x}^{(2h)} \\ &= \boldsymbol{P}\boldsymbol{A}^{(h)}\boldsymbol{P}^T\boldsymbol{x}^{(2h)} = \boldsymbol{P}\boldsymbol{A}^{(h)}\boldsymbol{x}^{(h)} \\ &= \boldsymbol{P}\boldsymbol{r}^{(h)}. \end{aligned}$$

Discrete Fourier Transform

- The solutions to hyperbolic PDEs like Poisson are wave-like and take on simple representations in the frequency basis, both for continuous and discretized equations. We define the *discrete Fourier transform* using

$$\omega_{(n)} = \cos(2\pi/n) - i\sin(2\pi/n) = e^{-2\pi i/n}.$$

The DFT matrix $\boldsymbol{F} \in \mathbb{R}^{n\times n}$ *is given by* $f_{ij} = \omega_{(n)}^{ij}$*,*

$$\boldsymbol{F} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_{(4)}^1 & \omega_{(4)}^2 & \omega_{(4)}^3 \\ 1 & \omega_{(4)}^2 & \omega_{(4)}^4 & \omega_{(4)}^6 \\ 1 & \omega_{(4)}^3 & \omega_{(4)}^6 & \omega_{(4)}^9 \end{bmatrix}$$

  - *it is complex and symmetric (not Hermitian),*
  - *it is unitary modulo scaling* $\boldsymbol{F}^* = n\boldsymbol{F}^{-1}$*.*

The discrete Fourier transform of vector $\boldsymbol{v}$ *is* $\boldsymbol{F}\boldsymbol{v}$*.*

Fast Fourier Transform (FFT)

- Consider $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{a}$, we have

$$\forall j \in [0, n-1] \quad b_j = \sum_{k=0}^{n-1} \omega_{(n)}^{jk} a_k,$$

the FFT computes this recursively via 2 FFTs of dimension $n/2$, using $\omega_{(n/2)} = \omega_{(n)}^2$,

$$\begin{aligned} b_j &= \sum_{k=0}^{n/2-1} \omega_{(n)}^{j(2k)} a_{2k} + \sum_{k=0}^{n/2-1} \omega_{(n)}^{j(2k+1)} a_{2k+1} \\ &= \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k} + \omega_{(n)}^{j} \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k+1} \end{aligned}$$

Fast Fourier Transform Derivation

- The FFT leverages similarity between the first and second half of the output,

$$b_j = \underbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k}}_{u_j} + \omega_{(n)}^{j} \underbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k+1}}_{v_j}$$

corresponds closely to the entry shifted by $n/2$,

$$b_{j+n/2} = \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{(j+n/2)k} a_{2k} + \omega_{(n)}^{j+n/2} \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{(j+n/2)k} a_{2k+1}$$

Now $\omega_{(n/2)}^{(j+n/2)k} = \omega_{(n/2)}^{jk}$ *since* $(\omega_{(n/2)}^{n/2})^k = 1^k = 1$ *and using* $\omega_{(n)}^{n/2} = -1$,

$$b_{j+n/2} = \underbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k}}_{u_j} - \omega_{(n)}^{j} \underbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k+1}}_{v_j}$$

FFT Algorithm Summary

- Let vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ be two recursive FFTs, $\forall j \in [0, n/2-1]$

$$u_j = \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k}, \quad v_j = \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k+1}$$

  - *Given $\boldsymbol{u}$ and $\boldsymbol{v}$ scale using "twiddle factors"* $z_j = \omega_{(n)}^{j} \cdot v_j$
  - *Then it suffices to combine the vectors as follows* $\boldsymbol{b} = \begin{bmatrix} \boldsymbol{u} + \boldsymbol{z} \\ \boldsymbol{u} - \boldsymbol{z} \end{bmatrix}$
- The FFT has $O(n \log n)$ cost complexity:

  There are two recursive calls of dimension $n/2$ and $O(n)$ work for application to twiddle factors and final summation, thus

$$T(n) = 2T(n) + O(n) = O(n \log n).$$

Applications of the FFT

- We can rapidly multiply degree n polynomials by considering their values $\omega^i_{(2n-1)}$ for $i \in \{0, \dots, 2n-1\}$

$$p_c(\omega^i_{(2n-1)}) = p_a(\omega^i_{(2n-1)})p_b(\omega^i_{(2n-1)})$$

  - *The product of coefficients of p_a, p_b with Vandermonde matrix $v_{ij} = (\omega^i_{(2n-1)})^j$, which is the DFT matrix, gives values of polynomials at $2n-1$ nodes.*
  - *Interpolation to compute coefficients of p_c from the products of values of p_a and p_b at those nodes is multiplication by the inverted DFT matrix and is exact since p_c is degree $2n-2$.*
- More generally the DFT can be used to solve any Toeplitz linear system (convolution):
  - *A standard convolution has the form, $\forall k \in [0, n-1] \quad c_k = \sum_{j=0}^{k} a_j b_{k-j}$.*
  - *Convolution is equivalent to multiplications of polynomials with degree $n/2-1$ and coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$, where the convolution computes the coefficients $\boldsymbol{c}$ of the product of the two polynomials.*

Convolution via DFT

- The Fourier transform method for computing a convolution is given by

$$c_k = \frac{1}{n} \sum_s \omega_{(n)}^{-ks} \Big(\sum_j \omega_{(n)}^{sj} a_j \Big) \Big(\sum_t \omega_{(n)}^{st} b_t \Big)$$

  - *Rearrange the order of the summations to see what happens to every product of a and b*

$$c_k = \frac{1}{n} \sum_s \sum_j \sum_t \omega_{(n)}^{(j+t-k)s} a_j b_t$$

  - *For any $u = j + t - k \neq 0$, we observe $\sum_s (\omega_{(n)}^{u})^s = 0$*
  - *When $j + t - k = 0$ the products $\omega_{(n)}^{(s+t-j)k} = 1$, so there are n nonzero terms $a_j b_{k-j}$ in the summation*

Solving Numerical PDEs with the FFT

- 1D finite-difference schemes on a regular grid correspond to convolutions:
  1D model problem is simply convolution with vector $[1, -2, 1]$.
- For the 1D Poisson model problem, the eigenvectors of $\boldsymbol{T}$ corresponds to the imaginary part of a minor of a $2(n+1)$-dimensional DFT matrix:
  - *In particular,* $\boldsymbol{T} = \boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^{-1}$ *where* x_{ij} *is the imaginary part of* $f_{i+1,j+1}$ *with* $\boldsymbol{X} \in \mathbb{R}^{n \times n}$ *and* $\boldsymbol{F} \in \mathbb{R}^{2(n+1) \times 2(n+1)}$.
  - *Consequently,* $\boldsymbol{T}$ *can be diagonalized and the overall system solved by FFT with* $O(n \log n)$ *cost.*
- Multidimensional Poisson can be handled with multidimensional FFT:
  For example 2D FFT (1D FFT of each row then 1D FFT of each column) suffices to solve the 2D Poisson problem.