Solving the Poisson Equation

Michael Kazhdan

(600.657)

Outline

Direct Methods
- The Fast Fourier Transform

Preliminaries

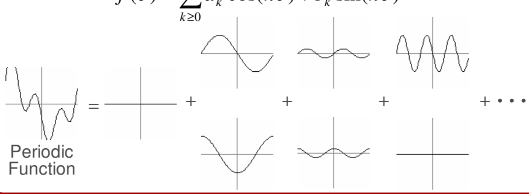
Iterative / Relaxation Methods
- Jacobi
- Steepest Descent

Direct Methods

Fourier Decomposition (1D):

Given a real-valued periodic function $f(\theta)$, with period 2π, we can express $f(\theta)$ in terms of its cosine / sine decomposition:

$$f(\theta)=\sum_{k\geq 0} a_k\cos(k\theta)+b_k\sin(k\theta)$$



Direct Methods

Fourier Decomposition (1D):

More generally, given a complex-valued periodic function $f(\theta)$, with period 2π, we can express $f(\theta)$ in terms of its Fourier decomposition:

$$f(\theta)=\sum_k \hat{f}[k]e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Both the cosine/sine and Fourier decompositions have the property that the Laplacian of the constituent functions is:

$$\Delta\cos(k\theta)=$$

$$\Delta\sin(k\theta)=$$

$$\Delta e^{ik\theta}=$$

Direct Methods

Fourier Decomposition (1D):

Both the cosine/sine and Fourier decompositions have the property that the Laplacian of the constituent functions is:

$$\Delta\cos(k\theta)=-k^2\cos(k\theta)$$

$$\Delta\sin(k\theta)=$$

$$\Delta e^{ik\theta}=$$

Direct Methods

Fourier Decomposition (1D):

Both the cosine/sine and Fourier decompositions have the property that the Laplacian of the constituent functions is:

$$\Delta\cos(k\theta) = -k^2\cos(k\theta)$$

$$\Delta\sin(k\theta) = -k^2\sin(k\theta)$$

$$\Delta e^{ik\theta} =$$

Direct Methods

Fourier Decomposition (1D):

Both the cosine/sine and Fourier decompositions have the property that the Laplacian of the constituent functions is:

$$\Delta\cos(k\theta) = -k^2\cos(k\theta)$$

$$\Delta\sin(k\theta) = -k^2\sin(k\theta)$$

$$\Delta e^{ik\theta} = -k^2 e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Both the cosine/sine and Fourier decompositions have the property that the Laplacian of the constituent functions is:

$$\Delta\cos(k\theta) = -k^2\cos(k\theta)$$

$$\Delta\sin(k\theta) = -k^2\sin(k\theta)$$

$$\Delta e^{ik\theta} = -k^2 e^{ik\theta}$$

These functions are the eigenvectors of the Laplacian operator!

Direct Methods

Fourier Decomposition (1D):

Suppose we are given a known function $g(\theta)$ and we would like to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

Direct Methods

Fourier Decomposition (1D):

Expressing *f* and *g* in terms of their Fourier decomposition:

$$f(\theta) = \sum_k \hat{f}[k]e^{ik\theta} \qquad g(\theta) = \sum_k \hat{g}[k]e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Expressing *f* and *g* in terms of their Fourier decomposition:

$$f(\theta) = \sum_k \hat{f}[k]e^{ik\theta} \qquad g(\theta) = \sum_k \hat{g}[k]e^{ik\theta}$$

… the Poisson equation becomes:

$$\Delta f(\theta) = g(\theta)$$

$$\Updownarrow$$

$$\Delta\left(\sum_k \hat{f}[k]e^{ik\theta}\right) = \sum_k \hat{g}[k]e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Applying the Laplacian operator to the complex exponentials $e^{ik\theta}$, this gives:

$$\Delta\left(\sum_k \hat{f}[k]e^{ik\theta}\right) = \sum_k \hat{g}[k]e^{ik\theta}$$

$$\Updownarrow$$

$$\sum_k -k^2 \hat{f}[k]e^{ik\theta} = \sum_k \hat{g}[k]e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Applying the Laplacian operator to the complex exponentials $e^{ik\theta}$, this gives:

$$\Delta\left(\sum_k \hat{f}[k]e^{ik\theta}\right) = \sum_k \hat{g}[k]e^{ik\theta}$$

$$\Updownarrow$$

$$\sum_k -k^2 \hat{f}[k]e^{ik\theta} = \sum_k \hat{g}[k]e^{ik\theta}$$

$$\Updownarrow$$

$$-k^2 \hat{f}[k] = \hat{g}[k]$$

Direct Methods

Fourier Decomposition (1D):

Given a function $g(\theta)$, to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

Direct Methods

Fourier Decomposition (1D):

Given a function $g(\theta)$, to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

1. We compute the Fourier decomposition of $g(\theta)$:

$$g(\theta) = \sum_k \hat{g}[k]e^{ik\theta}$$

Direct Methods

Fourier Decomposition (1D):

Given a function $g(\theta)$, to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

1. We compute the Fourier decomposition of $g(\theta)$:

$$g(\theta) = \sum_k \hat{g}[k]e^{ik\theta}$$

2. We scale the k-th coefficient of g by $-1/k^2$:

$$\hat{f}[k] = -\frac{1}{k^2}\hat{g}[k]$$

Direct Methods

Fourier Decomposition (1D):

Given a function $g(\theta)$, to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

1. We compute the Fourier decomposition of $g(\theta)$:

$$g(\theta) = \sum_k \hat{g}[k]e^{ik\theta}$$

2. We scale the k-th coefficient of g by $-1/k^2$:

$$\hat{f}[k] = -\frac{1}{k^2}\hat{g}[k]$$

3. We compute the inverse Fourier decomposition:

$$\sum_k \hat{f}[k]e^{ik\theta} = f(\theta)$$

Direct Methods

N-Dimensional Array: *n*=*N*

Given a function $g(\theta)$, to solve for the function $f(\theta)$ satisfying the Poisson equation:

$$\Delta f(\theta) = g(\theta)$$

1. We compute the Fourier decomposition of $g(\theta)$:
$$g(\theta) = \sum_k \hat{g}[k] e^{ik\theta}$$
O(*n* log *n*)
2. We scale the *k*-th coefficient of *g* by $-1/k^2$:
$$\hat{f}[k] = -\frac{1}{k^2}\hat{g}[k]$$
O(*n*)
3. We compute the inverse Fourier decomposition:
$$\sum_k \hat{f}[k] e^{ik\theta} = f(\theta)$$
O(*n* log *n*)

Direct Methods

Fourier Decomposition (2D):

For 2D functions, the Fourier decomposition gives an expression for the periodic function $f(\theta,\phi)$ as:

$$f(\theta,\varphi) = \sum_{k,l} \hat{f}[k][l] e^{ik\theta} e^{il\phi}$$

Direct Methods

Fourier Decomposition (2D):

Applying the Laplacian to the 2D complex exponential we get:

$$\Delta\left(e^{ik\theta}e^{il\phi}\right) = \frac{\partial^2}{\partial\theta^2}\left(e^{ik\theta}e^{il\phi}\right) + \frac{\partial^2}{\partial\phi^2}\left(e^{ik\theta}e^{il\phi}\right)$$

Direct Methods

Fourier Decomposition (2D):

Applying the Laplacian to the 2D complex exponential we get:

$$\begin{aligned}\Delta\left(e^{ik\theta}e^{il\phi}\right) &= \frac{\partial^2}{\partial\theta^2}\left(e^{ik\theta}e^{il\phi}\right) + \frac{\partial^2}{\partial\phi^2}\left(e^{ik\theta}e^{il\phi}\right)\\ &= -k^2\left(e^{ik\theta}e^{il\phi}\right) - l^2\left(e^{ik\theta}e^{il\phi}\right)\end{aligned}$$

Direct Methods

Fourier Decomposition (2D):

Applying the Laplacian to the 2D complex exponential we get:

$$\begin{aligned}\Delta\left(e^{ik\theta}e^{il\phi}\right) &= \frac{\partial^2}{\partial\theta^2}\left(e^{ik\theta}e^{il\phi}\right) + \frac{\partial^2}{\partial\phi^2}\left(e^{ik\theta}e^{il\phi}\right)\\ &= -k^2\left(e^{ik\theta}e^{il\phi}\right) - l^2\left(e^{ik\theta}e^{il\phi}\right)\\ &= \left(-k^2 - l^2\right)\left(e^{ik\theta}e^{il\phi}\right)\end{aligned}$$

Direct Methods

Fourier Decomposition (2D):

Applying the Laplacian to the 2D complex exponential we get:

$$\begin{aligned}\Delta\left(e^{ik\theta}e^{il\phi}\right) &= \frac{\partial^2}{\partial\theta^2}\left(e^{ik\theta}e^{il\phi}\right) + \frac{\partial^2}{\partial\phi^2}\left(e^{ik\theta}e^{il\phi}\right)\\ &= -k^2\left(e^{ik\theta}e^{il\phi}\right) - l^2\left(e^{ik\theta}e^{il\phi}\right)\\ &= \left(-k^2 - l^2\right)\left(e^{ik\theta}e^{il\phi}\right)\end{aligned}$$

As in the 1D case, the complex exponentials are the eigenvectors of the Laplacian operator.

Direct Methods

Fourier Decomposition (2D):

Given a function $g(\theta,\phi)$, to solve for the function $f(\theta,\phi)$ satisfying the Poisson equation:

$$\Delta f(\theta,\phi) = g(\theta,\phi)$$

Direct Methods

Fourier Decomposition (2D):

Given a function $g(\theta,\phi)$, to solve for the function $f(\theta,\phi)$ satisfying the Poisson equation:

$$\Delta f(\theta,\phi) = g(\theta,\phi)$$

1. We compute the Fourier decomposition of $g(\theta,\phi)$:
$$g(\theta,\phi) = \sum_k \hat{g}[k][l] e^{ik\theta} e^{il\phi}$$

Direct Methods

Fourier Decomposition (2D):

Given a function $g(\theta,\phi)$, to solve for the function $f(\theta,\phi)$ satisfying the Poisson equation:

$$\Delta f(\theta,\phi) = g(\theta,\phi)$$

1. We compute the Fourier decomposition of $g(\theta,\phi)$:
$$g(\theta,\phi) = \sum_k \hat{g}[k][l] e^{ik\theta} e^{il\phi}$$
2. We scale the (k,l)-th coefficient of g by $-1/(k^2+l^2)$:
$$\hat{f}[k][l] = -\frac{1}{k^2+l^2}\hat{g}[k][l]$$

Direct Methods

Fourier Decomposition (2D):

Given a function $g(\theta,\phi)$, to solve for the function $f(\theta,\phi)$ satisfying the Poisson equation:

$$\Delta f(\theta,\phi) = g(\theta,\phi)$$

1. We compute the Fourier decomposition of $g(\theta,\phi)$:
$$g(\theta,\phi) = \sum_k \hat{g}[k][l] e^{ik\theta} e^{il\phi}$$
2. We scale the (k,l)-th coefficient of g by $-1/(k^2+l^2)$:
$$\hat{f}[k][l] = -\frac{1}{k^2+l^2}\hat{g}[k][l]$$
3. We compute the inverse Fourier decomposition:
$$\sum_k \hat{f}[k][l] e^{ik\theta} e^{il\phi} = f(\theta,\phi)$$

Direct Methods

NxN-Dimensional Grid: $n=N^2$

Given a function $g(\theta,\phi)$, to solve for the function $f(\theta,\phi)$ satisfying the Poisson equation:

$$\Delta f(\theta,\phi) = g(\theta,\phi)$$

1. We compute the Fourier decomposition of $g(\theta,\phi)$:
$$g(\theta,\phi) = \sum_k \hat{g}[k][l] e^{ik\theta} e^{il\phi}$$
O(n log n)
2. We scale the (k,l)-th coefficient of g by $-1/(k^2+l^2)$:
$$\hat{f}[k][l] = -\frac{1}{k^2+l^2}\hat{g}[k][l]$$
O(n)
3. We compute the inverse Fourier decomposition:
$$\sum_k \hat{f}[k][l] e^{ik\theta} e^{il\phi} = f(\theta,\phi)$$
O(n log n)

Outline

Direct Methods

Preliminaries

Iterative / Relaxation Methods

Preliminaries (Dense *n*x*n* Matrices)

Matrix-vector multiplication

$O(n^2)$ time

Matrix inversion takes

$O(n^3)$ time: Gaussian Elimination

$O(n^{2.807})$: Strassen Inversion

$O(n^{2.376})$: Coppersmith-Winograd

Preliminaries (Sparse *n*x*n* Matrices)

Matrix-vector multiplication

$O(n)$ time

Matrix inversion takes

$\geq O(n^2)$ time

In general, the inverse of a sparse matrix will be a dense matrix.

Matrix Convergence

Given an *n*x*n* matrix *M*, then for any vector *v*, the sequence:

$$\{v, Mv, M^2v, \ldots, M^kv, \ldots\}$$

converges to zero if the matrix *M* is guaranteed to "shrink" the size of vectors.

Matrix Convergence

If we define the notion of "size" as the magnitude of the largest coefficient (L_∞-norm) of the vector:

$$\|v\|_\infty = \max_{0\leq k<n} |v[k]|$$

what condition should *M* satisfy to guarantee that it "shrinks" vectors?

Matrix Convergence

Claim:

A sufficient condition for the convergence is that the sum of the absolute values of the coefficients in each row is less than one:

$$\max_{0\leq i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right) < 1$$

Matrix Convergence

$$\max_{0\leq i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right) < 1$$

Proof:

The *i*-th coefficient of *Mv* is:

$$(Mv)[i] = \sum_{i=0}^{n} M[i][j] \cdot v[j]$$

Matrix Convergence

Proof: $\max_{0\le i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right)<1$

The *i*-th coefficient of *Mv* is:

$$(Mv)[i]=\sum_{i=0}^{n}M[i][j]\cdot v[j]$$

$$|(Mv)[i]|=\left|\sum_{i=0}^{n}M[i][j]\cdot v[j]\right|$$

Matrix Convergence

Proof: $\max_{0\le i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right)<1$

The *i*-th coefficient of *Mv* is:

$$(Mv)[i]=\sum_{i=0}^{n}M[i][j]\cdot v[j]$$

$$|(Mv)[i]|=\left|\sum_{i=0}^{n}M[i][j]\cdot v[j]\right|$$

$$|(Mv)[i]|\le\sum_{i=0}^{n}|M[i][j]|\cdot|v[j]|$$

Matrix Convergence

Proof: $\max_{0\le i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right)<1$

The *i*-th coefficient of *Mv* is:

$$(Mv)[i]=\sum_{i=0}^{n}M[i][j]\cdot v[j]$$

$$|(Mv)[i]|=\left|\sum_{i=0}^{n}M[i][j]\cdot v[j]\right|$$

$$|(Mv)[i]|\le\sum_{i=0}^{n}|M[i][j]|\cdot|v[j]|$$

$$|(Mv)[i]|\le\left(\sum_{i=0}^{n}|M[i][j]|\right)\|v\|_\infty$$

Matrix Convergence

Proof: $\max_{0\le i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right)<1$

Since we have:

$$|(Mv)[i]|\le\left(\sum_{i=0}^{n}|M[i][j]|\right)\|v\|_\infty$$

for every coefficient *i*, it follows that:

$$\|Mv\|_\infty\le\left(\sum_{i=0}^{n}|M[i][j]|\right)\|v\|_\infty$$

Matrix Convergence

Proof: $\max_{0\le i<n}\left(\sum_{j=0}^{n-1}|M[i][j]|\right)<1$

Since we have:

$$|(Mv)[i]|\le\left(\sum_{i=0}^{n}|M[i][j]|\right)\|v\|_\infty$$

for every coefficient *i*, it follows that:

$$\|Mv\|_\infty\le\left(\sum_{i=0}^{n}|M[i][j]|\right)\|v\|_\infty$$

Combining this with the condition on the magnitude of the row coefficients, we get:

$$\|Mv\|_\infty<\|v\|_\infty$$

Outline

Direct Methods

Preliminaries

Iterative / Relaxation Methods
- Jacobi
- Steepest Descent

Iterative / Relaxation Methods: *Ax=b*

General Problem:

Given an *n*x*n* matrix *A* and given an *n*-dim. vector *b*, solve for the *n*-dim. vector *x* such that:

$$Ax = b$$

Iterative / Relaxation Methods: *Ax=b*

Direct Approach (Inversion):

Compute A^{-1} and apply to the vector *b*:

$$x = A^{-1}b$$

Iterative / Relaxation Methods: *Ax=b*

Direct Approach (Inversion):

Compute A^{-1} and apply to the vector *b*:

$$x = A^{-1}b$$

Limitations:

Temporal Complexity:

1. $O(n^3)$: Gaussian Elimination
2. $O(n^{2.807})$: Strassen Inversion
3. $O(n^{2.376})$: Coppersmith-Winograd

Iterative / Relaxation Methods: *Ax=b*

Direct Approach (Inversion):

Compute A^{-1} and apply to the vector *b*:

$$x = A^{-1}b$$

Limitations:

Temporal Complexity:

1. $O(n^3)$: Gaussian Elimination
2. $O(n^{2.807})$: Strassen Inversion
3. $O(n^{2.376})$: Coppersmith-Winograd

Spatial Complexity: $O(n^2)$

Note: In general, even if the matrix *A* is sparse, the matrix A^{-1} will not be.

Iterative / Relaxation Methods: *Ax=b*

Motivation:

Solving for the inverse matrix A^{-1} is overkill.

The inverse matrix allows us to solve the Poisson equation for any vector *b*.

We are only interested in the solution for a specific vector.

Iterative / Relaxation Methods: *Ax=b*

Approach:

Define an iterative process that takes as its input a preliminary guess for the solution and returns an improved guess.

Applying the process to some initial guess x^0, we obtain a sequence of vectors:

$$\{x^0, x^1, \dots, x^i, \dots\}$$

converging to the solution:

$$\lim_{i \to \infty} \left\| Ax^i - b \right\| \to 0$$

Iterative / Relaxation Methods: *Ax=b*

Key Idea:

Rather than trying to solve for all the coefficients of *x* simultaneously, we will try to iteratively perform 1D optimizations.

Iterative / Relaxation Methods: *Ax=b*

Direct Approach:

- For some number of iterations:
  - For each $0 \le k < n$:
    - Update $x[k]$ by fixing all but the k-th coefficient of $x[\,]$ and solving for the optimal value of $x[k]$.

Iterative / Relaxation Methods: *Ax=b*

Direct Approach:

To update the value of $x[k]$, we treat the matrix A as a set of column vectors A_k:

$$\begin{pmatrix} A_0 & \cdots & A_{n-1} \end{pmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

Iterative / Relaxation Methods: *Ax=b*

Direct Approach:

To update the value of $x[k]$, we treat the matrix A as a set of column vectors A_k:

$$\begin{pmatrix} A_0 & \cdots & A_{n-1} \end{pmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

Then the equation for $x[k]$ becomes:

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i \neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i \neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

Direct Approach:

Setting v to be the constant component gives:

$$x[k] = \arg\min_t \left\| tA_k + v \right\|^2$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i \neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

Direct Approach:

Setting v to be the constant component gives:

$$x[k] = \arg\min_t \left\| tA_k + v \right\|^2 = -\frac{\langle A_k, v \rangle}{\langle A_k, A_k \rangle}$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

$$x[k] = \arg\min_t \|tA_k + v\|^2 = -\frac{\langle A_k, v\rangle}{\langle A_k, A_k\rangle}$$

Direct Approach:

Q: What is the complexity of updating *x*[*k*]?

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

$$x[k] = \arg\min_t \|tA_k + v\|^2 = -\frac{\langle A_k, v\rangle}{\langle A_k, A_k\rangle}$$

Direct Approach:

Q: What is the complexity of updating *x*[*k*]?

A: Computing *v* amounts to a matrix-vector multiplication so O(*n*) for sparse matrices.

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

$$x[k] = \arg\min_t \|tA_k + v\|^2 = -\frac{\langle A_k, v\rangle}{\langle A_k, A_k\rangle}$$

Direct Approach:

Q: What is the complexity of updating *x*[*k*]?

One iteration of the solver to update all the coefficients of x[ ] will taker order O(n^2).

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b \right\|^2$$

$$x[k] = \arg\min_t \|tA_k + v\|^2 = -\frac{\langle A_k, v\rangle}{\langle A_k, A_k\rangle}$$

Direct Approach:

Q: What is the complexity of updating *x*[*k*]?

One iteration of the solver to update all the

If we are solving the Poisson equation on an *N*x*N* grid ($n=N^2$) this would result in an O(N^4) algorithm!

Iterative / Relaxation Methods: *Ax=b*

To address the limitations of the direct approach, we consider two different types of approaches:

1. Jacobi:
   Minimize the amount of work required to update the coefficient *x*[*k*].
2. Steepest Descent:
   Choose the "coefficients" more carefully so that less work is required to get to the correct solution.

Iterative / Relaxation Methods: *Ax=b*

Jacobi:

- For some number of iterations:
  - For each $0 \le k < n$:
    - Update *x*[*k*] by fixing all but the *k*-th coefficient of *x*[ ] and solving for the optimal value of x[*k*].

Iterative / Relaxation Methods: *Ax=b*

Jacobi:

When updating $x[k]$, assume that the k-th column vector A_k only has a non-zero entry in the k-th coefficient ($A_k[j]=0$ for all $j\neq k$).

$$\begin{pmatrix} A_0 \cdots \begin{matrix}0\\0\\A_k[k]\\0\\0\end{matrix} \cdots A_{n-1}\end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} b \end{pmatrix}$$

Iterative / Relaxation Methods: *Ax=b*

Jacobi:

When updating $x[k]$, assume that the k-th column vector A_k only has a non-zero entry in the k-th coefficient ($A_k[j]=0$ for all $j\neq k$).

$$\begin{pmatrix} A_0 \cdots \begin{matrix}0\\0\\A_k[k]\\0\\0\end{matrix} \cdots A_{n-1}\end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} b \end{pmatrix}$$

Then the values of $(Ax)[j]$ is independent of $x[k]$ whenever $j\neq k$.

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0\\i\neq k}}^{n-1} x[i]A_i\right) - b}_{v} \right\|^2$$

Jacobi:

In this case, the optimization:

$$x[k] = \arg\min_t \|tA_k + v\|^2$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0\\i\neq k}}^{n-1} x[i]A_i\right) - b}_{v} \right\|^2$$

Jacobi:

In this case, the optimization:

$$x[k] = \arg\min_t \|tA_k + v\|^2$$

$$= \arg\min_t \sum_{j=0}^{n-1} (tA_k[j] + v[j])^2$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0\\i\neq k}}^{n-1} x[i]A_i\right) - b}_{v} \right\|^2$$

Jacobi:

In this case, the optimization:

$$x[k] = \arg\min_t \|tA_k + v\|^2$$

$$= \arg\min_t \sum_{j=0}^{n-1} (tA_k[j] + v[j])^2$$

becomes:

$$x[k] = \arg\min_t (tA_k[k] + v[k])^2$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0\\i\neq k}}^{n-1} x[i]A_i\right) - b}_{v} \right\|^2$$

$$x[k] = \arg\min_t (tA_k[k] + v[k])^2$$

Jacobi:

Or in other words:

$$x[k] = -\frac{v[k]}{A[k][k]} = -\frac{\left(\sum_{\substack{i=0\\i\neq k}}^{n-1} x[i]A_i[k]\right) - b[k]}{A[k][k]}$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b}_{v} \right\|^2$$

Jacobi:

What is really going on?

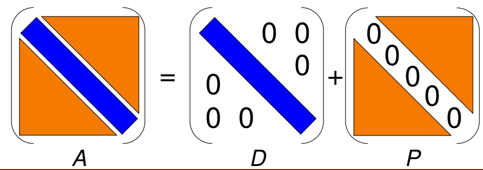
Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b}_{v} \right\|^2$$

Jacobi:

We decompose the matrix *A* as the sum *A*=*D*+*P*:

- *D* is the diagonal part of *A*.
- *P* is everything else.



Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b}_{v} \right\|^2$$

Jacobi:

We decompose the matrix *A* as the sum *A*=*D*+*P*:

- *D* is the diagonal part of *A*.
- *P* is everything else.

With respect to this decomposition, we get:

$$\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i[k] = Px$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b}_{v} \right\|^2$$

Jacobi:

We decompose the matrix *A* as the sum *A*=*D*+*P*:

- *D* is the diagonal part of *A*.
- *P* is everything else.

With respect to this decomposition, we get:

$$\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i[k] = Px$$

and the update becomes:

$$x[k] = -\frac{(Px)[k] - b[k]}{A[k][k]}$$

Iterative / Relaxation Methods: *Ax=b*

$$x[k] = \arg\min_t \left\| tA_k + \underbrace{\left(\sum_{\substack{i=0 \\ i\neq k}}^{n-1} x[i]A_i \right) - b}_{v} \right\|^2$$

Jacobi:

We decompose the matrix *A* as the sum *A*=*D*+*P*:

- *D* is the diagonal part of *A*.
- *P* is everything else.

If we update all of the coefficients of *x* at once, then an iteration of the Jacobi solver becomes:

$$x \leftarrow D^{-1}\left(b - (Px)\right)$$

Iterative / Relaxation Methods: *Ax=b*

$$x \leftarrow D^{-1}\left(b - (Px)\right)$$

Jacobi:

We can think of the Jacobi solver as an iterative solver that takes a guess for the solution x^l and returns the improved guess x^{l+1}:

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Iterative / Relaxation Methods: *Ax=b*

$$x \leftarrow D^{-1}\left(b-(Px)\right)$$

Jacobi:

We can think of the Jacobi solver as an iterative solver that takes a guess for the solution x^l and returns the improved guess x^{l+1}:

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

We need to show:

1. The true solution is a fixed point of the update.
2. The process converges.

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

Jacobi (Fixed Point):

If x^l is the true solution, $Ax^l-b=0$, then:

$$(P+D)x^l - b = 0$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

Jacobi (Fixed Point):

If x^l is the true solution, $Ax^l-b=0$, then:

$$(P+D)x^l - b = 0$$

$$\Updownarrow$$

$$D^{-1}(P+D)x^l - D^{-1}b = 0$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

Jacobi (Fixed Point):

If x^l is the true solution, $Ax^l-b=0$, then:

$$(P+D)x^l - b = 0$$

$$\Updownarrow$$

$$D^{-1}(P+D)x^l - D^{-1}b = 0$$

$$\Updownarrow$$

$$x^l = D^{-1}(b - Px^l)$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

Jacobi (Fixed Point):

If x^l is the true solution, $Ax^l-b=0$, then:

$$(P+D)x^l - b = 0$$

$$\Updownarrow$$

$$D^{-1}(P+D)x^l - D^{-1}b = 0$$

$$\Updownarrow$$

$$x^l = D^{-1}(b - Px^l)$$

$$\Updownarrow$$

$$x^l = x^{l+1}$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b-(Px^l)\right)$$

Jacobi (Fixed Point):

If x^l is the true solution, $Ax^l-b=0$, then:

$$(P+D)x^l - b = 0$$

$$\Updownarrow$$

The true solution is a fixed point of the Jacobi iterative process.

$$x^l = D^{-1}(b - Px^l)$$

$$\Updownarrow$$

$$x^l = x^{l+1}$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

To show this, we need to show that the errors tend to zero:

$$\lim_{l\to\infty}\left(x^l - x\right) = 0$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

Expressing the solution property in terms of the decomposition *A*=*P*+*D* gives:

$$Ax = b$$
$$\Updownarrow$$
$$(P+D)x = b$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

Expressing the solution property in terms of the decomposition *A*=*P*+*D* gives:

$$Ax = b$$
$$\Updownarrow$$
$$(P+D)x = b$$
$$\Updownarrow$$
$$x = D^{-1}(b - Px)$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$
$$x = D^{-1}\left(b - (Px)\right)$$

Jacobi (Convergence):

Subtracting the two properties we get:

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$
$$-\qquad x = D^{-1}\left(b - (Px)\right)$$
$$x^{l+1} - x = D^{-1}\left(b - \left(Px^l\right)\right) - D^{-1}\left(b - (Px)\right)$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$
$$x = D^{-1}\left(b - (Px)\right)$$

Jacobi (Convergence):

Subtracting the two properties we get:

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$
$$-\qquad x = D^{-1}\left(b - (Px)\right)$$
$$x^{l+1} - x = D^{-1}\left(b - \left(Px^l\right)\right) - D^{-1}\left(b - (Px)\right)$$
$$= D^{-1}P\left(x - x^l\right)$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$
$$x = D^{-1}\left(b - (Px)\right)$$

Jacobi (Convergence):

Subtracting the two properties we get:

$$x^{l+1} - x = -D^{-1}P\left(x^l - x\right)$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

$$x = D^{-1}\left(b - \left(Px\right)\right)$$

Jacobi (Convergence):

Subtracting the two properties we get:

$$x^{l+1} - x = -D^{-1}P\left(x^l - x\right)$$

So, the error at the (l+1)-th iteration is obtained by applying the matrix $-D^{-1}P$ to the error at the l-th iteration.

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

Subtracting the two properties we get:

$$x^{l+1} - x = -D^{-1}P\left(x^l - x\right)$$

So, the error at the (l+1)-th iteration is obtained by applying the matrix $-D^{-1}P$ to the error at the l-th iteration.

Thus, if the matrix $-D^{-1}P$ is guaranteed to "shrink" vectors, we will have convergence.

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

A sufficient condition for convergence is that each row vector ($0 \le i < n$), the matrix $-D^{-1}P$ satisfies:

$$\sum_{j=0}^{n-1} \left|\left(D^{-1}P\right)[i][j]\right| < 1$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

A sufficient condition for convergence is that each row vector ($0 \le i < n$), the matrix $-D^{-1}P$ satisfies:

$$\sum_{j=0}^{n-1} \left|\left(D^{-1}P\right)[i][j]\right| < 1$$

$$\Updownarrow$$

$$\sum_{j=0}^{n-1} \left|\frac{P[i][j]}{A[i][i]}\right| < 1$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

A sufficient condition for convergence is that each row vector ($0 \le i < n$), the matrix $-D^{-1}P$ satisfies:

$$\sum_{j=0}^{n-1} \left|\left(D^{-1}P\right)[i][j]\right| < 1$$

$$\Updownarrow$$

$$\sum_{j=0}^{n-1} \left|\frac{P[i][j]}{A[i][i]}\right| < 1$$

$$\Updownarrow$$

$$\sum_{\substack{j=0 \\ j \ne i}}^{n-1} \left|A[i][j]\right| < \left|A[i][i]\right|$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Convergence):

Thus, a sufficient condition for convergence is that the matrix A is diagonal-dominant:

$$\sum_{\substack{j=0 \\ j \ne i}}^{n-1} \left|A[i][j]\right| < \left|A[i][i]\right|$$

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Complexity):

As opposed to the direct method, the Jacobi method requires one matrix-vector multiply in order to update all of the coefficients.

Iterative / Relaxation Methods: *Ax=b*

$$x^{l+1} = D^{-1}\left(b - \left(Px^l\right)\right)$$

Jacobi (Complexity):

As opposed to the direct method, the Jacobi method requires one matrix-vector multiply in order to update all of the coefficients.

When the matrix *A* is sparse, an iteration updating all of the coefficients takes O(*n*) time.

Iterative / Relaxation Methods: *Ax=b*

Gauss-Seidel Sovler:

In the Jacobi solver, we compute the updated coefficient *x*[*k*] in one iteration, but do not use it until the next iteration.

In the Gauss-Seidel solver, we use it in the same iteration for updating *x*[*k'*] with $k'>k$:

- Does not require additional memory for storing in-between vector.
- Tends to converge more efficiently.

Outline

Direct Methods

Preliminaries

Iterative / Relaxation Methods

- Jacobi
- Steepest Descent

Iterative / Relaxation Methods: *Ax=b*

Approach:

In order to efficiently solve the Poisson equation, the next method focuses on effectively choosing directions for performing the update.

Iterative / Relaxation Methods: *Ax=b*

Motivation:

If the matrix A is symmetric and positive definite, the value of *x* satisfying the condition *Ax=b* can be realized as the minimizer of the equation:

$$F(x) = \frac{x^t A x}{2} - b^t x$$

Iterative / Relaxation Methods: *Ax=b*

Proof:

We first show that the point *Ax=b* is the unique critical point, and then we shown that it is the minimum.

Iterative / Relaxation Methods: *Ax=b*

Proof (Unique Critical):

Computing the gradient of the equation, we get:

$$F(x) = \frac{x^t A x}{2} - b^t x$$

$$\Downarrow$$

$$\nabla F(x) = Ax - b$$

so that *Ax=b* is a critical point.

Furthermore, since *A* is definite, it is invertible and *Ax=b* is the only critical point.

Iterative / Relaxation Methods: *Ax=b*

$$F(x) = \frac{x^t A x}{2} - b^t x$$

Proof (Minima):

Fixing a position p_0 and direction v_0 we can define the 1D function:

$$f(t) = F(p_0 + t v_0)$$

Iterative / Relaxation Methods: *Ax=b*

$$F(x) = \frac{x^t A x}{2} - b^t x$$

Proof (Minima):

Fixing a position p_0 and direction v_0 we can define the 1D function:

$$f(t) = F(p_0 + t v_0)$$

Computing the second derivative of *f* we get:

$$f''(t) = v_0^t A v_0$$

and since *A* is positive, this implies that the second derivative is positive.

Iterative / Relaxation Methods: *Ax=b*

$$F(x) = \frac{x^t A x}{2} - b^t x$$

Steepest Descent:

Given an initial guess x_0, we obtain the next guess by stepping away from x_0 in a direction opposite the gradient:

$$x_1 = x_0 - t \cdot \nabla F(x_0)$$

Iterative / Relaxation Methods: *Ax=b*

$$F(x) = \frac{x^t A x}{2} - b^t x$$

Steepest Descent:

Given an initial guess x_0, we obtain the next guess by stepping away from x_0 in a direction opposite the gradient:

$$x_1 = x_0 - t \cdot \nabla F(x_0)$$

Specifically, if we define *f* to be the 1D function:

$$f(t) = F\big(x_0 - t \cdot \nabla F(x_0)\big)$$

we need to find the value of *t* minimizing *f*.

Iterative / Relaxation Methods: *Ax=b*

$$F(x)=\frac{x^tAx}{2}-b^tx$$

Steepest Descent:

Denoting $r_i=\nabla F(x_i)$, we can re-write the equation:

$$f(t)=F\left(x_0-t\cdot\nabla F(x_0)\right)$$

to get:

$$f(t)=\frac{1}{2}(x_0-tr_0)^tA(x_0-tr_0)-b^t(x_0-tr_0)$$

Iterative / Relaxation Methods: *Ax=b*

$$F(x)=\frac{x^tAx}{2}-b^tx$$

Steepest Descent:

Denoting $r_i=\nabla F(x_i)$, we can re-write the equation:

$$f(t)=F\left(x_0-t\cdot\nabla F(x_0)\right)$$

to get:

$$f(t)=\frac{1}{2}(x_0-tr_0)^tA(x_0-tr_0)-b^t(x_0-tr_0)$$

$$\Downarrow$$

$$f'(t)=t\cdot r_0^tAr_0-r_0^t(b-Ax_0)$$

Iterative / Relaxation Methods: *Ax=b*

$$F(x)=\frac{x^tAx}{2}-b^tx$$

Steepest Descent:

Denoting $r_i=\nabla F(x_i)$, we can re-write the equation:

$$f(t)=F\left(x_0-t\cdot\nabla F(x_0)\right)$$

to get:

$$f(t)=\frac{1}{2}(x_0-tr_0)^tA(x_0-tr_0)-b^t(x_0-tr_0)$$

$$\Downarrow$$

$$f'(t)=t\cdot r_0^tAr_0-r_0^t(b-Ax_0)$$

$$=t\cdot r_0^tAr_0-r_0^tr_0$$

Iterative / Relaxation Methods: *Ax=b*

$$F(x)=\frac{x^tAx}{2}-b^tx$$

Steepest Descent:

Thus, the 1D function $f(t)$ is minimized at:

$$t=\frac{r_0^tr_0}{r_0^tAr_0}$$

and the next guess is obtained by stepping in the direction opposite the gradient:

$$x_1=x_0-\frac{r_0^tr_0}{r_0^tAr_0}\nabla F(x_0)$$