# Linear equations and iterative methods

Judd Chapter 3

David Childers (thanks to Y. Kryukov, K. Judd, and U. Doraszelski)

CMU, Tepper School of Business

March 15, 2023

## Numerical Linear Algebra

- Linear algebra is basic building block of numerical mathematics
- Algorithms extremely well developed: every language has low and medium level library implementing efficient algorithms
  - *BLAS*: Basic Linear Algebra Subprograms to implement basics, eg OpenBLAS/MKL
  - Mid-level: LAPACK almost universal for standard implementations of applied algorithms
  - High-level: Julia LinearAlgebra, Python scipy/numpy `linalg`
  - Specialized libraries: (eg for deep learning) reimplement to take advantage of hardware (GPU, etc) or extra structure
- You should understand building blocks of these algorithms: speed, robustness, accuracy in different situations
- Core problems
  - `A*B` / `A @ B`, `A\b`, `eig(A)` / `eigen(A)`, etc.
  - Many implementations each, with different tradeoffs

## Warmup: Matrix Muliplication

- Let A, B be $n \times n$ matrices
- Naive algorithm follows from definition of $A * B$

$$[A * B]_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

- Takes $n$ multiplications and additions for each of $n^2$ entries
  - $O(2n^3)$ operations total
- Coppersmith-Winograd algorithm is $\approx O(n^{2.37})$
  - But constant large: only use if n huge
- Default implementations have cost in between (e.g. $O(n^{2.81})$)
- Matrix-vector multiply $A * b$, $b$ $n \times 1$ is $O(2n^2)$ by same logic

## Exploiting Structure

- Many matrices arising in practice have additional structure
    - Symmetric, Diagonal, Toeplitz, triangular, sparse, low rank, etc
- Sparse matrices have $m << n^2$ nonzero entries
- Toeplitz & circulant matrices occur under invariances
- Fast algorithms exist which depend on structure: e.g.
    - Diagonal (or tridiagonal) matrix multiplication is $O(n)$
    - Circulant matrix-vector multiplication is $O(n \log(n))$: FFT
    - Will use this for interpolation, etc
- Some algorithms work for all matrices, faster when structured
- Others need explicit info about structure: use *types* to represent
- See `scipy.linalg` or JuliaLinearAlgebra library families

## The Plan: systems of linear equations

- Direct methods – reliable but slow
  - Back substitution
  - LU decomposition
  - QR and Cholesky decompositions
- Measuring precision: Eigenvalues and condition numbers
- Iterative methods:
  - Classic version – Gauss-Jacobi
  - An improvement – Gauss-Seidel
  - Convergence, dampening and acceleration
  - Alternatives
- Technical improvements: parallelization, randomization, sparsity
- Application: Eigenvectors and Markov chains

## Systems of linear equations

$$Ax = b,$$

where $A$ is an $n \times n$ matrix, $x$ and $b$ are $n \times 1$ vectors

- Some simple economic problems are linear:
  - General Equilibrium with linear supply and demand
  - OLS estimator: $(X'X)\,\hat{\beta} = X'Y$
- Used as a building block in other methods
  - Newton's method for nonlinear equations or optimization
  - Polynomial or spline coefficients in approximation
  - Value function in policy iteration
  - Ergodic (stable) distribution of Markov Chain
- Simple example used to illustrate iterative methods

## Back-substitution

Suppose $A$ is lower triangular, i.e,

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}.$$

Then we can easily solve it:

$$\begin{aligned} x_1 &= \frac{b_1}{a_{11}}, \\ x_k &= \frac{b_k - \sum_{j=1}^{k-1} a_{kj} x_j}{a_{kk}}, \quad k = 2, 3, \ldots, n. \end{aligned}$$

## LU decomposition

- Find $L$ and $U$ such that:

$$A = LU$$

    - $L$ is **L**ower triangular
    - $U$ is **U**pper triangular
    - Matlab/Julia (LinearAlgebra)/Scipy: `lu(A)`
- Solving system of equations $Ax = b \Leftrightarrow L[Ux] = b$:
    - Find $L$ and $U$
    - Solve $Lz = b$ for $z$ by back-substitution
    - Solve $Ux = z$ for $x$ by back-substitution
    - Matlab/Julia: `x = A\b` (backslash or "left division")
      Numpy/scipy `solve(A,b)`

## Other Decompositions

- **QR decomposition**: $A = QR$.
  - $Q$ is orthogonal ($Q'Q$ is an identity matrix),
  - $R$ is upper triangular.
  - Matlab/Julia/Numpy/Scipy: `qr(A)`
- Solution: $Ax = b \Leftrightarrow Q'QRx = Q'b$,
  - $Q'QR$ is upper triangular $\Rightarrow$ solve for $x$ by back-substitution
  - One back-substitution, but decomposition takes longer
- **Cholesky decomposition** ("square root" of $A$)
  - Exists if $A$ is symmetric and positive definite
  - $A = CC'$, $C$ is lower triangular ($\Rightarrow C'$ – upper).
  - Matlab: `chol(A)` Julia/Numpy/Scipy: `cholesky(A)`

## A bit of theory: matrix analysis

- $\lambda \in \mathbb{C}$ is an **eigenvalue** of $n \times n$ matrix $A$ iff:
  - there is an eigenvector $v \in \mathbb{C}^n$, $v \neq 0$, ...
  - ... such that $Av = \lambda v$, and $\det(A - \lambda I) = 0$.
- **Spectrum** $\sigma(A)$ = set of all $n$ eigenvalues
  - Matlab/Numpy/Scipy: `eig(A)` Julia: `eigvals(A)`
- **Spectral radius** $\rho(A) := \max |\sigma(A)|$
- **Norm**: $||A|| = \max_{x \neq 0} \frac{||Ax||}{||x||} = \max_{||x||=1} ||Ax||$ `= norm(A) / opnorm(A)`
- **Condition number**: $cond(A) = ||A|| \cdot ||A^{-1}||$ `= cond(A)`
  - If $A$ is singular, then the condition number is $+\infty$.
  - If $A$ is near-singular, condition number is high ($>10^{10}$)
- **Spectral condition number**: $cond_*(A) = \rho(A) / \min |\sigma(A)|$
  - $cond(A) \geq cond_*(A)$
  - They tend to have same order of magnitude ($m$ in $10^m$)

## Bounding the errors

- Perturb the system by $r$: $A\tilde{x} = b + r$ (e.g. due to rounding) $\Longrightarrow$ Error $e = \tilde{x} - x$.
- Elasticity of solution w.r.t. error:

$$\frac{1}{cond(A)} \leq \frac{||e||/||x||}{||r||/||b||} \leq cond(A).$$

- Rule of thumb: each order of magnitude in condition number loses one decimal digit of accuracy from $x$
- Since we only care about orders of magnitude,
we can use $cond_*(A)$ instead – it is a lot faster to compute.
- $\Rightarrow$ Pre-conditioning: pick $D$ so $DA$ is well-conditioned, solve

$$(DA)\,x = (Db)$$

## Speed of computation

- LU decomposition:
    - Decomposition: $n^3/3$ multiplications and divisions
    - Backward substitution: $n^2$ multiplications and divisions
- QR decomposition – also $Kn^3$ operations
- Cramer's rule (determinants) $=$ $n!$ operations $=$ very slow
    - It is useful in theory work, as an expression for solution
- We can often improve speed at cost of a bit of precision
    - $\Longrightarrow$ Iterative methods

## Gauss-Jacobi iterations

- Idea: solve $i$-th equation for $x_i$ alone:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i$$
$$\Rightarrow x_i = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j \neq i} a_{ij} x_j \right\}$$

- Use it to iterate: have guess $x^k$, compute $x^{k+1}$ as:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j \neq i}^{n} a_{ij} x_j^k \right\}, \quad i = 1, \ldots, n$$

- Need a starting guess $x^0$.
- Need a stopping rule, e.g.:

$$\frac{\left\| x^{k+1} - x^k \right\|}{\left\| x^k \right\| + 1} < \delta$$

## Gauss-Seidel

- Gauss-Jacobi computes the last element ($x_n^{k+1}$) as:

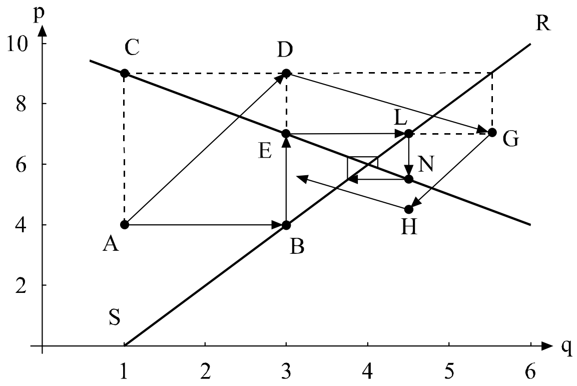
$$x_n^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \textstyle\sum_{j=1}^{n-1} a_{ij} x_j^k \right\}$$

  - If we update elements in natural order $(1, ..., n)$ we already have $x_1^{k+1}, ..., x_{n-1}^{k+1}$ computed.
  - It would speed things up to use them rather than $x^k$
- Gauss-Seidel does the same for every element $x_i^{k+1}$:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \textstyle\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right\}$$

- See Figure 3.2 in the textbook for an illustration.

# Figure 3.2



Gauss-Jacobi (ADGH) versus Gauss-Seidel (ABELN..)

## Convergence and operator splitting

- Operator splitting: $A = (N - P)$

$$\begin{aligned} Nx &= b + Px \\ x^{k+1} &= N^{-1}(b + Px^k) \end{aligned}$$

- $N$ is selected so it is easy to invert:
    - GJ: $N$ is the diagonal of $A$
    - GS (using natural order): $N$ is the lower triangle of $A$
- GJ/GS are linearly convergent at rate $\rho(N^{-1}P)$
- Condition for convergence: $\rho(N^{-1}P) < 1$
    - $\iff$ all eigenvalues are inside the unit circle
    - $\Leftarrow$ Matrix $A$ diagonally dominant: $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$
    - 1-dimensional interpretation: slope less than 1

## Precision under linear convergence

- Linear convergence (plus some conditions) imply: $\left\|x^{k+1}-x^{k}\right\| \leq \beta\left\|x^{k}-x^{k-1}\right\|$
  - where $\beta$ is the linear convergence rate
- It follows that $\left\|x^{k}-x^{*}\right\| \leq\left\|x^{k+1}-x^{k}\right\| /(1-\beta)$
  - To prove, use triangle inequality
- So, if we want to ensure $\left\|x^{k}-x^{*}\right\|<\delta$, stop when

$$\left\|x^{k+1}-x^{k}\right\| \leq \delta(1-\beta)$$

- Since we do not know $\beta$, we can estimate: $\hat{\beta}=\max_{\kappa=1, \ldots, k}\left\{\left\|x^{\kappa}-x^{\kappa-1}\right\| /\left\|x^{\kappa-1}-x^{\kappa-2}\right\|\right\}$
- Of course, this approach is valid only if there is convergence

## Guarantees for iterative methods

- Gerschgorin circle theorem:
  - Let $A$ have eigenvalues $\{\lambda_s\}_{s=1}^n$, $R_i = \sum_{j\neq i} |a_{ij}|$
  - Then $\lambda_s \in \cup_{i=1}^n B(a_{ii}, R_i)\ \forall s$
- Corollary: If $A$ is diagonally dominant with $\beta = \sup\{|x| :\ x \in \cup_{i=1}^n B(a_{ii}^{-1}, R_i)\} < 1$, then Gauss-Jacobi converges to within error $\delta$ in $O(n^2 \log(\frac{1}{\delta}))$ operations
- Informal proof:
  - Each iteration is a diagonal matrix multiply, which is $O(n^2)$
  - Gerschgorin $\implies \rho(N^{-1}P) \leq \beta < 1, \implies$ linear convergence
- Compare $O(n^3)$ for direct methods
  - Substantial improvement even for machine precision error
- For G-S, can show $A$ diagonal dominant OR symmetric positive definite $\implies \rho(N^{-1}P) < 1$
  - Occurs in least squares problems, many 2nd order PDEs
- These structures are sufficient, not necessary

## Dampening: a tweak to iterations

- Gauss-Jacobi algorithm takes steps $\Delta x$ at each iteration:
  - Each iteration computes $x^{k+1} = N^{-1}Px^k + N^{-1}b = Gx^k + d$
  - Step is $\Delta x^{k+1} = x^{k+1} - x^k = Gx^k + d - x^k$
- We can choose to scale $\Delta x^{k+1}$ by $\omega$:

$$x^{k+1} = \omega[Gx^k + d] + (1-\omega)x^k$$

- $\omega < 1$: Stabilization / dampening:
  - can create or speed up convergence
  - avoids exploding or going in circles around the solution
  - See Figure 3.4 in the texbook for an illustration.
- $\omega > 1$: Extrapolation – speeds up convergence (Fig. 3.5)

# Figure 3.4

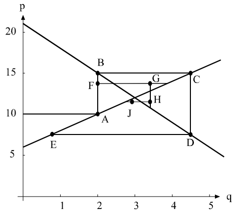
Dampening to Stabilize an Unstable “Hog Cycle”.

- Suppose inverse demand is $p = 21 - 3q$ and supply is $q = p/2 - 3$
- Linear system is not diagonally dominant:

$$\begin{pmatrix} 1 & 3 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} 21 \\ 6 \end{pmatrix} \tag{3.9.8}$$

- Gauss-Seidel is unstable:

$$p_{n+1} = 21 - 3q_n \tag{3.9.9a}$$
$$q_{n+1} = \frac{1}{2}p_{n+1} - 3 \tag{3.9.9b}$$



# Figure 3.5

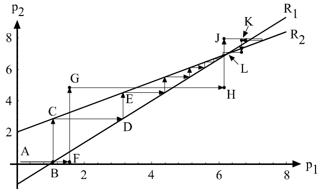
Exatrapolation to Accelerate Convergence in a Game

- Assume firm two's reaction curve is $p_2 = 2 + 0.80p_1 \equiv R_2(p_1)$, and firm one's reaction curve is $p_1 = 1 + 0.75p_2 \equiv R_1(p_2)$.
- Equilibrium system is diagonally dominant
- Gauss-Seidel is the iterative scheme

$$p_1^{n+1} = R_1\left(p_2^n\right) \tag{3.9.12a}$$
$$p_2^{n+1} = R_2\left(p_1^{n+1}\right) \tag{3.9.12b}$$

- Accelerate (3.9.12). If $\omega = 1.5$, we arrive at faster scheme:

$$p_1^{n+1} = 1.5R_1\left(p_2^n\right) - 0.5p_1^n, \tag{3.9.13a}$$
$$p_2^{n+1} = 1.5R_2\left(p_1^{n+1}\right) - 0.5p_2^n. \tag{3.9.13b}$$



## Dampening and convergence

- If all eigenvalues $\sigma(G)$ are real, optimal dampening is

$$\omega^* = \frac{2}{2 - \max|\sigma(G)| - \min|\sigma(G)|}$$

- Weyl's inequality states that $\sigma\{\omega G + (1-\omega)I\} \leq \omega\sigma(G) + (1-\omega)$
  - Dampening shifts (potentially complex) eigenvalues towards 1
  - If $\max \mathrm{real}\{\sigma(G)\} < 1$, dampening can restore stability and improve convergence speed.
  - If $\max \mathrm{real}\{\sigma(G)\} > 1$, dampening cannot restore stability.
- Can combine dampening with G-S (Successive overrelaxation)
  - Potentially even bigger gains

## Preview: nonlinear GS/GJ methods

- We can (and will) use iterative methods for nonlinear fixed-point problems: $x = F(x)$
- For $\sigma(G)$ and $\rho(G)$, $G$ is the Jacobian of $F$
- Convergence properties are local, and we can have more than one solution
- Dampening works exactly as described
- In general, convergence does not necessarily imply a solution
- For linear problem, we can **verify** solution by computing

$$\|Ax - b\|$$

  - For nonlinear fixed point problem, no general test

## Alternatives: Krylov subspace methods

- eg *Conjugate Gradient* for pos. def. $A$: *GMRES* for general $A$
- Combines strengths of iterative and direct methods
- "Exact" convergence in $n$ iterations by building solution from sequence of linearly independent vectors
    - Floating point error accumulates, worse than direct methods
    - Practical implementations require fixes: see Golub & van Loan
- Each iteration uses $A$ only in small # of matrix-vector multiplies
- No real gain over direct methods if no structure: $n \times O(n^2)$ ops
- Substantial gain in speed if
    1. Matrix-vector multiplies are fast: sparse, structured, etc
    2. $b$ (nearly) restricted to low-d subspace, or initial guess good
    3. Preconditioning can help ensure this
        - Cut off at small # of iterations for small loss in accuracy

## Parallelization

- GJ with single CPU computes $x_i^{k+1}$'s one after another
- If we have many CPUs, we can parallelize:
  - CPU1 computes $x_1^{k+1}$,
  - CPU2 computes $x_2^{k+1}$, and so on
  - These computations happen simultaneously
- Matlab has parallelization built-in, but it needs to be activated
  - In other languages, look for MPI libraries
- Not very useful with just 2 CPUs

## Randomized Methods

- Random sampling of data points or random feature transforms
- $P$ $m \times n$ random projection or transform, $m << p$
- Replace A by $P * A$ or $A * P'$ or even $PAP'$
- Reduce effective size of data matrices in econometrics/ML applications
- Trade off probability of error for gains in speed and memory
- Especially attractive if approximate structure exists
    - Sparse, low rank, or i.i.d. matrices amenable to accurate random transforms

## Structured linear problems – Sparse methods

- Matrix is "sparse" if most elements are zero
  - Can store only nonzeros elements (and their $i, j$)
- Skipping zero elements saves computation time
  - Computing $0 * 0$ would take same time as $3.14 * 2.72$
- To take advantage of it, sofware must be able to:
  - Store matrix as a list of nonzero elements
  - Use only nonzero elements in linear algebra operations
- *C*, *Fortran*, *Julia*, *Python* have sparse linear algebra libraries
  - Julia: Pkg `SparseArrays`, Python: `Scipy.sparse`
- Matlab/Julia have integrated support for sparse matrices:
  - `As=sparse(A);` $\Rightarrow$ most operations on `As` will use sparsity

## Sparse example – Markov transition

- Economic system with discrete states: $i = 1, ..., 100$:
  - Markov process: state transition depends only on current state
  - In each period, system can transit to a neighboring state, or stay in current state
- Markov transition matrix
  - $\pi_{ij} = \Pr\{\text{state} = j \text{ tomorrow} \mid \text{state} = i \text{ today}\}$
  - $\Pi = \{\pi_{i,j}\}$ has only 3 non-zero's per row, i.e. 97% of zeros:
  - If $x$ is the current distribution over states , then in the next period, we will have distribution $\Pi x$
- Computing $\Pi x$:
  - Normally requires $100^2$ multiplications & additions
  - Accounting for sparsity, we have only 300 multiplications
- Another example: stationary distribution solves $\Pi x = x$.

## Eigenvalue/vector computation

- Stationary distribution is example of an *eigenproblem*
- Find vector and complex scalar $(v, \lambda)$ s.t. $Av = \lambda v$
- Applications in dynamical systems & control (stability), statistics (PCA, CCA, etc), portfolio management
- Core component of several other problems: eg SVD
- Direct and iterative methods available here too
- Direct methods apply matrix decomposition
- Schur: $A = UTU^*$, $T$ upper triangular, $U$ unitary
    - Eigenvalues read off of diagonal of T
    - Computable in $O(n^3)$ by repeated $QR$ decompositions
    - Eigenvectors require additional $O(n^3)$ step: avoid if not needed
- Matlab/Julia/Python: use `eig()`/`eigvals()` for all eigenvalues by direct method

## Iterative eigenvalue computation

- Especially useful if not all eigenvalues needed
- Can find one at a time: often just need biggest or smallest
- Simplest method: *Power iteration*
  - $z_{k+1} = A * z_k / \|A * z_k\|$
  - Converges to eigenvector of largest eigenvalue if that is unique and if $z_0$ not orthogonal to it
  - Interpretation: Evolution of distribution under markov chain
  - Since uses multiplications, fast if sparse or structured
- Generalized power methods (Arnoldi or Lanczos, etc) improve speed especially for multiple eigenvalues
- Matlab: `eigs()` for subset of eigenvalues by iterative method
- Julia: `eigs()` in `Arpack`, or option in `eigvals()` for range

## Recommended References: General Numerical Linear Algebra

- Judd Chapter 3.
- Quantecon "Numerical Linear Algebra" https://julia.quantecon.org/tools_and_techniques/numerical_linear_algebra.html
- Golub, Gene and Charles van Loan. Matrix Computations
  - The source for matrix algorithms: LAPACK based on them
- Carron, Igor. *The Advanced Matrix Factorization Jungle* https://sites.google.com/site/igorcarron2/matrixfactorizations
- Higham, Nick. *Blog* https://nhigham.com/blog/
  - Linear algebra facts and algorithm explanations

## Specialized Topics References

- Iterative Methods
  - QuantEcon "Krylov Methods and Matrix Conditioning" https://julia.quantecon.org/tools_and_techniques/iterative_methods_sparsity.html
  - Goh, Gabriel. "Why Momentum Really Works" *Distill* 2017 http://distill.pub/2017/momentum
- Randomized methods
  - Mahoney, Michael. "Randomized algorithms for matrices and data" Foundations and Trends in Machine Learning: Vol. 3: No. 2, pp 123-224 https://arxiv.org/abs/1104.5557
  - Ng, Serena. "Opportunities and Challenges: Lessons from Analyzing Terrabytes of Scanner Data" 2017 NBER WP23673 http://www.nber.org/papers/w23673