CS 598: Provably Efficient Algorithms for Numerical and Combinatorial Problems

Part 1: Background Numerical Analysis Tools

Edgar Solomonik

University of Illinois at Urbana-Champaign

Matrices and Tensors

- What is a matrix?
  - *A collection of numbers arranged into an array of dimensions $m \times n$, e.g., $\boldsymbol{M} \in \mathbb{R}^{m \times n}$*
  - *A linear operator $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x}$*
  - *A bilinear form $\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{y}$*
- What is a tensor?
  - *A collection of numbers arranged into an array of a particular order, with dimensions $l \times m \times n \times \cdots$, e.g., $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{l \times m \times n}$ is order 3*
  - *A multilinear operator $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$*

$$z_i = \sum_{j,k} t_{ijk} x_j y_k$$

  - *A multilinear form $\sum_{i,j,k} t_{ijk} x_i y_j z_k$*

Matrix and Tensor Decompositions

- What is a matrix factorization?
  - *A decomposition of a matrix in terms of other matrices with desirable properties*
  - *$\boldsymbol{M} = \boldsymbol{QR} = \boldsymbol{USV}^T = \boldsymbol{XDX}^{-1}$ are examples of factorizations, where $\boldsymbol{Q}, \boldsymbol{U}, \boldsymbol{V}$ are orthogonal, $\boldsymbol{R}$ is upper-triangular, while $\boldsymbol{S}$ and $\boldsymbol{D}$ are diagonal*
  - *Factorizations enable compression of $\boldsymbol{M}$, solution to linear systems and least squares problems with $\boldsymbol{M}$, and computation of eigenvalues of $\boldsymbol{M}$*
- What is a tensor decomposition?
  - *A decomposition of a tensors in terms of other tensors with desirable properties*
  - *For example, the canonical polyadic (CP) decomposition of an order 3 tensor is*

$$t_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

  - *Factorizations enable compression of $\mathcal{T}$, may uncover semantic structure of $\mathcal{T}$, or may permit tensor network algorithms that represent $\mathcal{T}$ implicitly*

Graphs and Hypergraphs

- What is a graph?
  - *A set of vertices and edges $G = (V, E)$, with $V = \{1, \ldots, n\}$ and $E \subseteq V \times V$ possibly with a weight function $w : E \to \mathbb{R}$*
  - *An adjacency matrix $\boldsymbol{A}$, where for each $e = (i, j) \in E$, $a_{ij} = w(e)$*
- What is a hypergraph?
  - *A set of vertices and hyperedges $G = (V, H)$, where each $h \in H$ is a subset of vertices, $h \subseteq V$, possibly with a weight function w for hyperedges*
  - *A set of tensors $\boldsymbol{\mathcal{T}}^{(1)}, \cdots, \boldsymbol{\mathcal{T}}^{(n)}$ where $\boldsymbol{\mathcal{T}}^{(k)}$ is order k and stores hyperedges of order k, e.g. if $h = (i, j, k, l) \in H$ then $\boldsymbol{\mathcal{T}}^{(4)}_{ijkl} = w(h)$*

Numerical and Combinatorial Problems

- What numerical problems will we look at?
  - *Matrix factorizations via direct and approximate methods*
  - *Iterative methods for linear systems and eigenvalue problems*
  - *Discrete convolution*
  - *Tensor decomposition, tensor completion, tensor networks, and associated numerical optimization problems*
- What combinatorial problems will we look at?
  - *Sorting and partitioning*
  - *Graph problems: shortest paths, connectivity, minimal spanning tree*
- What applications do these have?
  - *Data mining and data compression (matrix/tensor decomposition)*
  - *Numerical methods for PDEs (linear systems and eigenvalue problems)*
  - *Machine learning, e.g. collaborative filtering (tensor completion, convolution)*
  - *Computational quantum chemistry and physics (tensor networks)*
  - *Graph analytics*

Provably Efficient Algorithms

- What makes an algorithm provably efficient?
  - *Polynomial time with respect to size of input*
  - *Low asymptotic worst-case execution time*
  - *Low asymptotic average-case execution time*
  - *Low (leading order) constant factors in execution time*
  - *Numerical stability (small errors made during execution are not amplified)*
  - *Approximation quality / error bounds (for inexact algorithms)*
  - *The algorithm is parallelizable (low depth)*
  - *The algorithm exhibits data-reuse / requires little communication*
  - *The algorithm requires little synchronization between threads or processes*
  - *The algorithm is cache oblivious*
  - *The runtime of an implementation of the algorithm is lower than comparable implementations of alternative algorithms on representative problem instances*

Provably Efficient Parallel Schedules

- What makes a parallel schedule good
  - *Low parallel execution time given an infinite number of processors*
  - *Low parallel execution time given any fixed number of processors*
  - *Different processors work on different data*
  - *Low communication and synchronization costs*
  - *Few cache misses*
  - *Low interprocessor communication volume*
  - *Few messages communicated*
  - *Low critical path costs*
- What makes a parallel schedule for a given algorithm optimal?
  - *Its execution time with an infinite number of processors is equal to the depth of the algorithm*
  - *Its execution time decreases linearly with the number of processors until it equals the depth*
  - *Its communication and synchronization costs match communication lower bounds for the algorithm*

Error Analysis

- Forward Error:

  Forward error is the computational error of an algorithm

  - *Absolute:* $\hat{f}(x) - f(x)$
  - *Relative:* $(\hat{f}(x) - f(x))/f(x)$

- Backward Error:

  Backward error analysis enables us to measure computational error with respect to data propagation error

  - *An algorithm is backward stable if its a solution to a nearby problem*
  - *If the computed solution* $\hat{f}(x) = f(\hat{x})$ *then*

    $$\textit{backward error} = \hat{x} - x$$

  - *More precisely, we want the nearest* $\hat{x}$ *to* x *with* $\hat{f}(x) = f(\hat{x})$

Conditioning

- Conditioning measures the worst-case sensitivity of the output with respect to perturbations of the input
- The absolute condition number is a property of the problem, which measures its sensitivity to perturbations in input

  For scalar problem f with input x it is simply the derivative of f at x,

$$\kappa_{abs}(f) = \lim_{\Delta x \to 0} \left| \frac{f(x + \Delta x) - f(x)}{\Delta x} \right| = \left| \frac{df}{dx}(x) \right|$$

  When considering a space of inputs $\mathcal{X}$ it is $\kappa_{abs} = \max_{x \in \mathcal{X}} \left| \frac{df}{dx}(x) \right|$
- The relative condition number considers relative perturbations in input and output, so that

$$\kappa(f) = \kappa_{rel}(f) = \max_{x \in \mathcal{X}} \lim_{\Delta x \to 0} \left| \frac{(f(x + \Delta x) - f(x))/f(x)}{\Delta x / x} \right| = \frac{\kappa_{abs}(f)|x|}{|f(x)|}$$

Rounding Error in Floating Point Operations

- Addition and Subtraction
  - *Catastrophic cancellation occurs when the magnitude of the result is much smaller than the magnitude of both operands*
  - *Cancellation corresponds to losing significant digits, e.g.*

$$3.1423 \times 10^5 - 3.1403 \times 10^5 = 2.0 \times 10^2$$

  - *Generally, we can bound the error incurred during addition of two real numbers x, y in floating point (ignoring final rounding, which has relative error ϵ) as*

$$\frac{|(x+y) - (fl(x) + fl(y)|}{|x+y|} \leq \frac{\epsilon(|x| + |y|)}{|x+y|}$$

    by this we can also observe that the condition number of addition of x, y i.e. $f(x, y) = x + y$, is $\kappa(f(x, y)) = (|x| + |y|)/|x + y|$
  - *Consequently, when $x + y = 0$ and $x, y \neq 0$ addition is ill-posed (has infinite condition number) unless we restrict the space of possible inputs x, y*

Matrix Condition Number

- The matrix condition number $\kappa(\boldsymbol{A})$ is the ratio between the max and min distance from the surface to the center of the unit ball transformed by $\kappa(\boldsymbol{A})$:
  - *The max distance to center is given by the vector maximizing* $\max_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||_2$.
  - *The min distance to center is given by the vector minimizing* $\min_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||_2 = 1/(\max_{||\boldsymbol{x}||=1} ||\boldsymbol{A}^{-1}\boldsymbol{x}||_2)$.
  - *Thus, we have that* $\kappa(\boldsymbol{A}) = ||\boldsymbol{A}||_2||\boldsymbol{A}^{-1}||_2$
- The matrix condition number bounds the worst-case amplification of error in a matrix-vector product: *Consider* $\boldsymbol{y} + \boldsymbol{\delta y} = \boldsymbol{A}(\boldsymbol{x} + \boldsymbol{\delta x})$, *assume* $||\boldsymbol{x}||_2 = 1$
  - *In the worst case,* $||\boldsymbol{y}||_2$ *is minimized, that is* $||\boldsymbol{y}||_2 = 1/||\boldsymbol{A}^{-1}||_2$
  - *In the worst case,* $||\boldsymbol{\delta y}||_2$ *is maximized, that is* $||\boldsymbol{\delta y}||_2 = ||\boldsymbol{A}||_2||\boldsymbol{\delta y}||_2$
  - *So* $||\boldsymbol{\delta y}||_2/||\boldsymbol{y}||_2$ *is at most* $\kappa(\boldsymbol{A})||\boldsymbol{\delta x}||_2/||\boldsymbol{x}||_2$

Singular Value Decomposition

- The singular value decomposition (SVD)

  We can express any matrix $\boldsymbol{A}$ as

  $$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

  where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal, and $\boldsymbol{\Sigma}$ is square nonnegative and diagonal,

  $$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{max} & & \\ & \ddots & \\ & & \sigma_{min} \end{bmatrix}$$

  Any matrix is diagonal when expressed as an operator mapping vectors from a coordinate system given by $\boldsymbol{V}$ to a coordinate system given by $\boldsymbol{U}^T$.

- Condition number in terms of singular values
  - *We have that $\|\boldsymbol{A}\|_2 = \sigma_{max}$ and if $\boldsymbol{A}^{-1}$ exists, $\|\boldsymbol{A}^{-1}\|_2 = 1/\sigma_{min}$*
  - *Consequently, $\kappa(\boldsymbol{A}) = \sigma_{max}/\sigma_{min}$*

Linear Least Squares

- Find $\boldsymbol{x}^\star = \operatorname{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2$ where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$:

  Since $m \geq n$, the minimizer generally does not attain a zero residual $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$. We can rewrite the optimization problem constraint via

$$\boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2 = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} \left[(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\right]$$

- Given the SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ we have $\boldsymbol{x}^\star = \underbrace{\boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T}_{\boldsymbol{A}^\dagger}\boldsymbol{b}$, where $\boldsymbol{\Sigma}^\dagger$ contains the reciprocal of all nonzeros in $\boldsymbol{\Sigma}$, and more generally $\dagger$ denotes pseudoinverse:
  - *The minimizer satisfies $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{x}^\star \cong \boldsymbol{b}$ and consequently also satisfies*

$$\boldsymbol{\Sigma}\boldsymbol{y}^\star \cong \boldsymbol{d} \quad \text{where } \boldsymbol{y}^\star = \boldsymbol{V}^T\boldsymbol{x}^\star \text{ and } \boldsymbol{d} = \boldsymbol{U}^T\boldsymbol{b}.$$

  - *The minimizer of the reduced problem is $\boldsymbol{y}^\star = \boldsymbol{\Sigma}^\dagger\boldsymbol{d}$, so $y_i = d_i/\sigma_i$ for $i \in \{1,\ldots,n\}$ and $y_i = 0$ for $i \in \{n+1,\ldots,m\}$.*

Normal Equations

Demo: *Normal equations vs Pseudoinverse*
Demo: *Issues with the normal equations*

- *Normal equations* are given by solving $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$:
  If $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$ *then*

$$
\begin{aligned}
(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{x} &= (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T\boldsymbol{b} \\
\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{x} &= \boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{b} \\
\boldsymbol{V}^T\boldsymbol{x} &= (\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{b} = \boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T\boldsymbol{b} \\
\boldsymbol{x} &= \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T\boldsymbol{b} = \boldsymbol{x}^\star
\end{aligned}
$$

- However, solving the normal equations is a more ill-conditioned problem then the original least squares algorithm
  Generally we have $\kappa(\boldsymbol{A}^T\boldsymbol{A}) = \kappa(\boldsymbol{A})^2$ *(the singular values of* $\boldsymbol{A}^T\boldsymbol{A}$ *are the squares of those in* $\boldsymbol{A}$*). Consequently, solving the least squares problem via the normal equations may be unstable because it involves solving a problem that has worse conditioning than the initial least squares problem.*

Solving the Normal Equations

- If $\boldsymbol{A}$ is full-rank, then $\boldsymbol{A}^T\boldsymbol{A}$ is symmetric positive definite (SPD):
  - *Symmetry is easy to check* $(\boldsymbol{A}^T\boldsymbol{A})^T = \boldsymbol{A}^T\boldsymbol{A}$.
  - $\boldsymbol{A}$ *being full-rank implies* $\sigma_{min} > 0$ *and further if* $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ *we have*

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}^T\boldsymbol{\Sigma}^2\boldsymbol{V}$$

    which implies that rows of $\boldsymbol{V}$ *are the eigenvectors of* $\boldsymbol{A}^T\boldsymbol{A}$ *with eigenvalues* $\boldsymbol{\Sigma}^2$ *since* $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{V}^T = \boldsymbol{V}^T\boldsymbol{\Sigma}^2$.

- Since $\boldsymbol{A}^T\boldsymbol{A}$ is SPD we can use Cholesky factorization, to factorize it and solve linear systems:

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$$

QR Factorization

- If $\boldsymbol{A}$ is full-rank there exists an orthogonal matrix $\boldsymbol{Q}$ and a unique upper-triangular matrix $\boldsymbol{R}$ with a positive diagonal such that $\boldsymbol{A} = \boldsymbol{QR}$
  - *Given $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{LL}^T$, we can take $\boldsymbol{R} = \boldsymbol{L}^T$ and obtain $\boldsymbol{Q} = \boldsymbol{AL}^{-T}$, since $\underbrace{\boldsymbol{L}^{-1}\boldsymbol{A}^T}_{\boldsymbol{Q}^T}\underbrace{\boldsymbol{AL}^{-T}}_{\boldsymbol{Q}} = \boldsymbol{I}$ implies that $\boldsymbol{Q}$ has orthonormal columns.*
- A reduced QR factorization (unique part of general QR) is defined so that $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$ has orthonormal columns and $\boldsymbol{R}$ is square and upper-triangular

  A full QR factorization gives $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{R} \in \mathbb{R}^{m\times n}$, but since $\boldsymbol{R}$ is upper triangular, the latter $m-n$ columns of $\boldsymbol{Q}$ are only constrained so as to keep $\boldsymbol{Q}$ orthogonal. The reduced QR factorization is given by taking the first n columns $\boldsymbol{Q}$ and $\hat{\boldsymbol{Q}}$ the upper-triangular block of $\boldsymbol{R}$, $\hat{\boldsymbol{R}}$ giving $\boldsymbol{A} = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}$.
- We can solve the normal equations (and consequently the linear least squares problem) via reduced QR as follows

$$\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b} \quad \Rightarrow \quad \hat{\boldsymbol{R}}^T\underbrace{\hat{\boldsymbol{Q}}^T\hat{\boldsymbol{Q}}}_{\boldsymbol{I}}\hat{\boldsymbol{R}}\boldsymbol{x} = \hat{\boldsymbol{R}}^T\hat{\boldsymbol{Q}}^T\boldsymbol{b} \quad \Rightarrow \quad \hat{\boldsymbol{R}}\boldsymbol{x} = \hat{\boldsymbol{Q}}^T\boldsymbol{b}$$

Eigenvalue Decomposition

- If a matrix $\boldsymbol{A}$ is diagonalizable, it has an *eigenvalue decomposition*

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^{-1}$$

where $\boldsymbol{X}$ are the right eigenvectors, $\boldsymbol{X}^{-1}$ are the left eigenvectors and $\boldsymbol{D}$ are eigenvalues

$$\boldsymbol{A}\boldsymbol{X} = \begin{bmatrix}\boldsymbol{A}\boldsymbol{x}_1 & \cdots \boldsymbol{A}\boldsymbol{x}_n\end{bmatrix} = \boldsymbol{X}\boldsymbol{D} = \begin{bmatrix}d_{11}\boldsymbol{x}_1 & \cdots & d_{nn}\boldsymbol{x}_n\end{bmatrix}.$$

  - *If $\boldsymbol{A}$ is symmetric, its right and left singular vectors are the same, and consequently are its eigenvectors.*
  - *More generally, any normal matrix, $\boldsymbol{A}^H\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H$, has unitary eigenvectors.*
- $\boldsymbol{A}$ and $\boldsymbol{B}$ are *similar*, if there exist $\boldsymbol{Z}$ such that $\boldsymbol{A} = \boldsymbol{Z}\boldsymbol{B}\boldsymbol{Z}^{-1}$
  - *Normal matrices are unitarily similar ($\boldsymbol{Z}^{-1} = \boldsymbol{Z}^H$) to diagonal matrices*
  - *Symmetric real matrices are orthogonally similar ($\boldsymbol{Z}^{-1} = \boldsymbol{Z}^T$) to real diagonal matrices*
  - *Hermitian matrices are unitarily similar to real diagonal matrices*

Similarity of Matrices

matrix	*similarity*	*reduced form*
SPD	orthogonal	real positive diagonal
real symmetric	orthogonal	real tridiagonal real diagonal
Hermitian	unitary	real diagonal
normal	unitary	diagonal
real	orthogonal	real Hessenberg
diagonalizable	invertible	diagonal
arbitrary	unitary invertible	triangular bidiagonal

Rayleigh Quotient

- For any vector $\boldsymbol{x}$ that is close to an eigenvector, the *Rayleigh quotient* provides an estimate of the associated eigenvalue of $\boldsymbol{A}$:

$$\rho_{\boldsymbol{A}}(\boldsymbol{x}) = \frac{\boldsymbol{x}^H \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^H \boldsymbol{x}}.$$

  - *If $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$, then $\rho_{\boldsymbol{A}}(\boldsymbol{x})$ is the associated eigenvalue.*
  - *Moreover, for $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, the Rayleigh quotient is the best possible eigenvalue estimate given $\boldsymbol{x}$ and $\boldsymbol{y}$, as it is the solution α to $\boldsymbol{x}\alpha \cong \boldsymbol{y}$.*
    - *The normal equations for this scalar-output least squares problem are (assuming $\boldsymbol{A}$ is real),*

$$\boldsymbol{x}^T \boldsymbol{x}\alpha = \boldsymbol{x}^T \boldsymbol{y} \quad \Rightarrow \quad \alpha = \frac{\boldsymbol{x}^T \boldsymbol{y}}{\boldsymbol{x}^T \boldsymbol{x}} = \frac{\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}.$$

Introduction to Krylov Subspace Methods

- *Krylov subspace methods* work with information contained in the $n \times k$ matrix

$$\boldsymbol{K}_k = \begin{bmatrix} \boldsymbol{x_0} & \boldsymbol{A x_0} & \cdots & \boldsymbol{A}^{k-1}\boldsymbol{x_0} \end{bmatrix}$$

We seek to best use the information from the matrix vector product results (columns of $\boldsymbol{K}_k$) to solve eigenvalue problems.

- $\boldsymbol{A}$ is similar to *companion matrix* $\boldsymbol{C} = \boldsymbol{K}_n^{-1}\boldsymbol{A}\boldsymbol{K}_n$:

Letting $\boldsymbol{k}_n^{(i)} = \boldsymbol{A}^{i-1}\boldsymbol{x}$, we observe that

$$\boldsymbol{A}\boldsymbol{K}_n = \begin{bmatrix} \boldsymbol{A}\boldsymbol{k}_n^{(1)} & \cdots & \boldsymbol{A}\boldsymbol{k}_n^{(n-1)} & \boldsymbol{A}\boldsymbol{k}_n^{(n)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{k}_n^{(2)} & \cdots & \boldsymbol{k}_n^{(n)} & \boldsymbol{A}\boldsymbol{k}_n^{(n)} \end{bmatrix},$$

therefore premultiplying by $\boldsymbol{K}_m^{-1}$ transforms the first $n-1$ columns of $\boldsymbol{A}\boldsymbol{K}_n$ into the last $n-1$ columns of $\boldsymbol{I}$,

$$\begin{aligned} \boldsymbol{K}_n^{-1}\boldsymbol{A}\boldsymbol{K}_n &= \begin{bmatrix} \boldsymbol{K}_n^{-1}\boldsymbol{k}_n^{(2)} & \cdots & \boldsymbol{K}_n^{-1}\boldsymbol{k}_n^{(n)} & \boldsymbol{K}_n^{-1}\boldsymbol{A}\boldsymbol{k}_n^{(n)} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_n & \boldsymbol{K}_n^{-1}\boldsymbol{A}\boldsymbol{k}_n^{(n)} \end{bmatrix} \end{aligned}$$

Krylov Subspaces

- Given $\boldsymbol{Q}_k\boldsymbol{R}_k = \boldsymbol{K}_k$, we obtain an orthonormal basis for the Krylov subspace,

$$\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{x}_0) = span(\boldsymbol{Q}_k) = \{p(\boldsymbol{A})\boldsymbol{x}_0 : deg(p) < k\},$$

  where p is any polynomial of degree less than k.

- The Krylov subspace includes the $k-1$ approximate dominant eigenvectors generated by $k-1$ steps of power iteration:
  - *The approximation obtained from $k-1$ steps of power iteration starting from $\boldsymbol{x}_0$ is given by the Rayleigh-quotient of $\boldsymbol{y} = \boldsymbol{A}^k\boldsymbol{x}_0$.*
  - *This vector is within the Krylov subspace, $\boldsymbol{y} \in \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{x}_0)$.*
  - *Consequently, Krylov subspace methods will generally obtain strictly better approximations of the dominant eigenpair than power iteration.*

Krylov Subspace Methods

- The $k \times k$ matrix $\boldsymbol{H}_k = \boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k$ minimizes $||\boldsymbol{A}\boldsymbol{Q}_k - \boldsymbol{Q}_k \boldsymbol{H}_k||_2$:
  The minimizer $\boldsymbol{X}$ for the linear least squares problem $\boldsymbol{Q}_k \boldsymbol{X} \cong \boldsymbol{A}\boldsymbol{Q}_k$ is (via the normal equations) $\boldsymbol{X} = \boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k = \boldsymbol{H}_k$.
- $\boldsymbol{H}_k$ is Hessenberg, because the companion matrix $\boldsymbol{C}_k$ is Hessenberg:

$$\boldsymbol{H}_k = \boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k = \boldsymbol{R}_k \boldsymbol{K}_k^{-1} \boldsymbol{A} \boldsymbol{K}_k \boldsymbol{R}_k^{-1} = \boldsymbol{R}_k \boldsymbol{C}_k \boldsymbol{R}_k^{-1}$$

  is a product of three matrices: upper-triangular $\boldsymbol{R}_k$, upper-Hessenberg $\boldsymbol{C}_k$, and upper-triangular $\boldsymbol{R}_k^{-1}$, which results in upper-Hessenberg $\boldsymbol{H}_k$.

Rayleigh-Ritz Procedure

- The eigenvalues/eigenvectors of $\boldsymbol{H}_k$ are the *Ritz values/vectors*:

$$\boldsymbol{H}_k = \boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^{-1}$$

eigenvalue approximations based on Ritz vectors $\boldsymbol{X}$ are given by $\boldsymbol{Q}_k\boldsymbol{X}$.

- The Ritz vectors and values are the *ideal approximations* of the actual eigenvalues and eigenvectors based on only $\boldsymbol{H}_k$ and $\boldsymbol{Q}_k$:

Assuming $\boldsymbol{A}$ is a symmetric matrix with positive eigenvalues, the largest Ritz value $\lambda_{max}(\boldsymbol{H}_k)$ will be the maximum Rayleigh quotient of any vector in $\mathcal{K}_k = span(\boldsymbol{Q}_k)$,

$$\max_{\boldsymbol{x}\in span(\boldsymbol{Q}_k)} \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} = \max_{\boldsymbol{y}\neq 0} \frac{\boldsymbol{y}^T\boldsymbol{Q}_k^T\boldsymbol{A}\boldsymbol{Q}_k\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{y}} = \max_{\boldsymbol{y}\neq 0} \frac{\boldsymbol{y}^T\boldsymbol{H}_k\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{y}} = \lambda_{max}(\boldsymbol{H}_k),$$

which is the best approximation to $\lambda_{max}(\boldsymbol{A}) = \max_{\boldsymbol{x}\neq 0} \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ available in $\mathcal{K}_k$. The quality of the approximation can also be shown to be optimal for other eigenvalues/eigenvectors.

General Multidimensional Optimization

- Steepest descent: minimize f in the direction of the negative gradient:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

such that $f(\boldsymbol{x}_{k+1}) = \min_{\alpha_k} f(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k))$*, i.e. perform a line search (solve 1D optimization problem) in the direction of the negative gradient.*

- Given quadratic optimization problem $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T \boldsymbol{x}$ where $\boldsymbol{A}$ is symmetric positive definite, the error $\boldsymbol{e}_k = \boldsymbol{x}_k - \boldsymbol{x}^*$ satisfies

$$||\boldsymbol{e}_{k+1}||_{\boldsymbol{A}} = \boldsymbol{e}_{k+1}^T \boldsymbol{A} \boldsymbol{e}_{k+1} = \frac{\sigma_{\mathsf{max}}(\boldsymbol{A}) - \sigma_{\mathsf{min}}(\boldsymbol{A})}{\sigma_{\mathsf{max}}(\boldsymbol{A}) + \sigma_{\mathsf{min}}(\boldsymbol{A})} ||\boldsymbol{e}_k||_{\boldsymbol{A}}$$

  - When sufficiently close to a local minima, general nonlinear optimization problems are described by such an SPD quadratic problem.
  - Convergence rate depends on the conditioning of $\boldsymbol{A}$, since

$$\frac{\sigma_{max}(\boldsymbol{A}) - \sigma_{min}(\boldsymbol{A})}{\sigma_{max}(\boldsymbol{A}) + \sigma_{min}(\boldsymbol{A})} = \frac{\kappa(\boldsymbol{A}) - 1}{\kappa(\boldsymbol{A}) + 1}.$$

Gradient Methods with Extrapolation

- We can improve the constant in the linear rate of convergence of steepest descent by leveraging *extrapolation methods*, which consider two previous iterates (maintain *momentum* in the direction $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}$):

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) + \beta_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})$$

- The *heavy ball method*, which uses constant $\alpha_k = \alpha$ and $\beta_k = \beta$, achieves better convergence than steepest descent:

$$||\boldsymbol{e}_{k+1}||_{\boldsymbol{A}} = \frac{\sqrt{\kappa(\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{A})} + 1} ||\boldsymbol{e}_k||_{\boldsymbol{A}}$$

Nesterov's gradient optimization method is another instance of an extrapolation method that provides further improved optimality guarantees.

Conjugate Gradient Method

- The *conjugate gradient method* is capable of making the optimal (for a quadratic objective) choice of α_k and β_k at each iteration of an extrapolation method:

$$(\alpha_k, \beta_k) = \underset{\alpha_k, \beta_k}{\operatorname{argmin}} \left[f\Big(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) + \beta_k (\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \Big) \right]$$

  - *For SPD quadratic programming problems, conjugate gradient is an optimal first order method, converging in n iterations.*
  - *It implicitly computes Lanczos iteration, searching along $\boldsymbol{A}$-orthogonal directions at each step.*

- *Parallel tangents* implementation of the method proceeds as follows

  1. *Perform a step of steepest descent to generate $\hat{\boldsymbol{x}}_k$ from $\boldsymbol{x}_k$.*
  2. *Generate $\boldsymbol{x}_{k+1}$ by minimizing over the line passing through $\boldsymbol{x}_{k-1}$ and $\hat{\boldsymbol{x}}_k$.*

The method is equivalent to CG for a quadratic objective function.

Krylov Optimization

- Conjugate Gradient finds the minimizer of $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$ (which satisfies optimality condition $\boldsymbol{A}\boldsymbol{x} = -\boldsymbol{c}$) within the Krylov subspace of $\boldsymbol{A}$:
  - *It constructs Krylov subspace* $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{c}) = span(\boldsymbol{c}, \boldsymbol{A}\boldsymbol{c}, \ldots, \boldsymbol{A}^{r-1}\boldsymbol{c})$.
  - *At the* k*th step conjugate gradient yields iterate*

$$\boldsymbol{x}_k = -||\boldsymbol{c}||_2 \boldsymbol{Q}_k \boldsymbol{T}_k^{-1} \boldsymbol{e}_1,$$

    where $\boldsymbol{Q}_k$ *is an orthogonal basis for Krylov subspace* $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{c})$ *and* $\boldsymbol{T}_k = \boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k$.
  - *This choice of* $\boldsymbol{x}_k$ *minimizes* $f(\boldsymbol{x})$ *since*

$$\begin{aligned}
\min_{\boldsymbol{x} \in \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{c})} f(\boldsymbol{x}) &= \min_{\boldsymbol{y} \in \mathbb{R}^k} f(\boldsymbol{Q}_k \boldsymbol{y}) \\
&= \min_{\boldsymbol{y} \in \mathbb{R}^k} \boldsymbol{y}^T \boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k \boldsymbol{y} + \boldsymbol{c}^T \boldsymbol{Q}_k \boldsymbol{y} \\
&= \min_{\boldsymbol{y} \in \mathbb{R}^k} \boldsymbol{y}^T \boldsymbol{T}_k \boldsymbol{y} + ||\boldsymbol{c}||_2 \boldsymbol{e}_1^T \boldsymbol{y}
\end{aligned}$$

    is minimized by $\boldsymbol{y} = -||\boldsymbol{c}||_2 \boldsymbol{T}_k^{-1} \boldsymbol{e}_1$.

Newton's Method

- Newton's method in n dimensions is given by finding minima of n-dimensional quadratic approximation using the gradient and Hessian of f:

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx \hat{f}(\boldsymbol{s}) = f(\boldsymbol{x}_k) + \boldsymbol{s}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2} \boldsymbol{s}^T \boldsymbol{H}_f(\boldsymbol{x}_k) \boldsymbol{s}.$$

The minima of this function can be determined by identifying critical points

$$\mathbf{0} = \nabla \hat{f}(\boldsymbol{s}) = \nabla f(\boldsymbol{x}_k) + \boldsymbol{H}_f(\boldsymbol{x}_k) \boldsymbol{s},$$

thus to determine $\boldsymbol{s}$ we solve the linear system,

$$\boldsymbol{H}_f(\boldsymbol{x}_k) \boldsymbol{s} = -\nabla f(\boldsymbol{x}_k).$$

Assuming invertibility of the Hessian, we can write the Newton's method iteration as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \underbrace{\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k)}_{\boldsymbol{s}}.$$

Quadratic convergence follows by equivalence to Newton's method for solving nonlinear system of optimality equations $\nabla f(\boldsymbol{x}) = \mathbf{0}$.

Nonlinear Least Squares

- An important special case of multidimensional optimization is *nonlinear least squares*, the problem of fitting a nonlinear function $f_{\boldsymbol{x}}(t)$ so that $f_{\boldsymbol{x}}(t_i) \approx y_i$:
  For example, consider fitting $f_{[x_1,x_2]}(t) = x_1 \sin(x_2 t)$ *so that*

$$\begin{bmatrix} f_{[x_1,x_2]}(1.5) \\ f_{[x_1,x_2]}(1.9) \\ f_{[x_1,x_2]}(3.2) \end{bmatrix} \approx \begin{bmatrix} -1.2 \\ 4.5 \\ 7.3 \end{bmatrix}.$$

- We can cast nonlinear least squares as an optimization problem to minimize residual error and solve it by Newton's method:
  Define residual vector function $\boldsymbol{r}(\boldsymbol{x})$ *so that* $r_i(\boldsymbol{x}) = y_i - f_{\boldsymbol{x}}(t_i)$ *and minimize*

$$\phi(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{r}(\boldsymbol{x})||_2^2 = \frac{1}{2}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}).$$

  Now the gradient is $\nabla\phi(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x})$ *and the Hessian is*

$$\boldsymbol{H}_\phi(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})\boldsymbol{H}_{r_i}(\boldsymbol{x}).$$

Gauss-Newton Method

- The Hessian for nonlinear least squares problems has the form:

$$\boldsymbol{H}_\phi(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})\boldsymbol{H}_{r_i}(\boldsymbol{x}).$$

The second term is small when the residual function $\boldsymbol{r}(\boldsymbol{x})$ is small, so approximate

$$\boldsymbol{H}_\phi(\boldsymbol{x}) \approx \hat{\boldsymbol{H}}_\phi(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}).$$

- The *Gauss-Newton* method is Newton iteration with an approximate Hessian:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \hat{\boldsymbol{H}}_\phi(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - (\boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x}_k)\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k))^{-1}\boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x}_k)\boldsymbol{r}(\boldsymbol{x}_k).$$

Recognizing the normal equations, we interpret the Gauss-Newton method as solving linear least squares problems $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\boldsymbol{s}_k \cong \boldsymbol{r}(\boldsymbol{x}_k), \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$.

Tensors

- A *tensor* $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ has
  - *Order* d *(i.e.* d *modes / indices)*
  - *Dimensions* n_1*-by-*$\cdots$*-by-*n_d
  - *Elements* $t_{i_1 \ldots i_d} = t_{\boldsymbol{i}}$ *where* $\boldsymbol{i} \in \bigotimes_{i=1}^{d} \{1, \ldots, n_i\}$
- Order d tensors represent d-dimensional arrays
  - $(d \geq 3)$*-dimensional arrays are prevalent in scientific computing*
    - *Regular grids, collections of matrices, multilinear operators*
    - *Experimental data, visual/graphic data*
    - *Higher-order derivatives and correlation*

Reshaping Tensors

When using tensors, it is often necessary to transition between high-order and low-order representations of the same object

- Recall for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ its *unfolding* is given by

$$\boldsymbol{v} = \operatorname{vec}(\boldsymbol{A}) \Rightarrow \boldsymbol{v} \in \mathbb{R}^{mn}, v_{i+jm} = a_{ij}$$

- A tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ can be fully unfolded the same way

$$\boldsymbol{v} = \operatorname{vec}(\boldsymbol{\mathcal{T}}) \Rightarrow \boldsymbol{v} \in \mathbb{R}^{n_1 \cdots n_d}, v_{i_1 + i_2 n_1 + i_3 n_1 n_2 + \ldots} = t_{i_1 i_2 i_3 \ldots}$$

- Often we also want to *fold* tensors into higher-order ones
- Generally, we can *reshape* (fold or unfold) any tensor

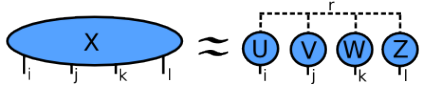
$$\boldsymbol{\mathcal{U}} = o_{n_1 \times \cdots \times n_d}(\boldsymbol{\mathcal{V}}) \Rightarrow \boldsymbol{U} \in \mathbb{R}^{n_1 \times \cdots \times n_d}, \quad \operatorname{vec}(\boldsymbol{\mathcal{U}}) = \operatorname{vec}(\boldsymbol{\mathcal{V}})$$

Canonical Polyadic (CP) Decomposition

- A rank R *CP decomposition* of an $s \times s \times s \times s$ tensor is

$$x_{ijkl} = \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} z_{lr}$$

- We can represent the CP using the following *tensor diagram*:



- Finding an approximate tensor decomposition corresponds to a nonlinear least squares problem:

$$f(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{Z}) = \sum_{i,j,k,l} \left(x_{ijkl} - \underbrace{\sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} z_{lr}}_{\hat{x}_{ijkl}} \right)^2$$

which is the squared Frobenius norm error $\|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\|_F^2$.