Numerical Methods I
MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Computing eigenvalues
- Singular value decomposition (SVD)

Today

- Computing the SVD
- Iterative methods for systems of linear equations

Announcements

- Homework 4 posted, is due Mon, Nov 7 before class
- There will be midterm grades, which are solely based on HW 1–2 (so not very informative!)

Recap: Singular value decomposition

Let $A \in \mathbb{C}^{m \times n}$. A singular value decomposition of A is a factorization

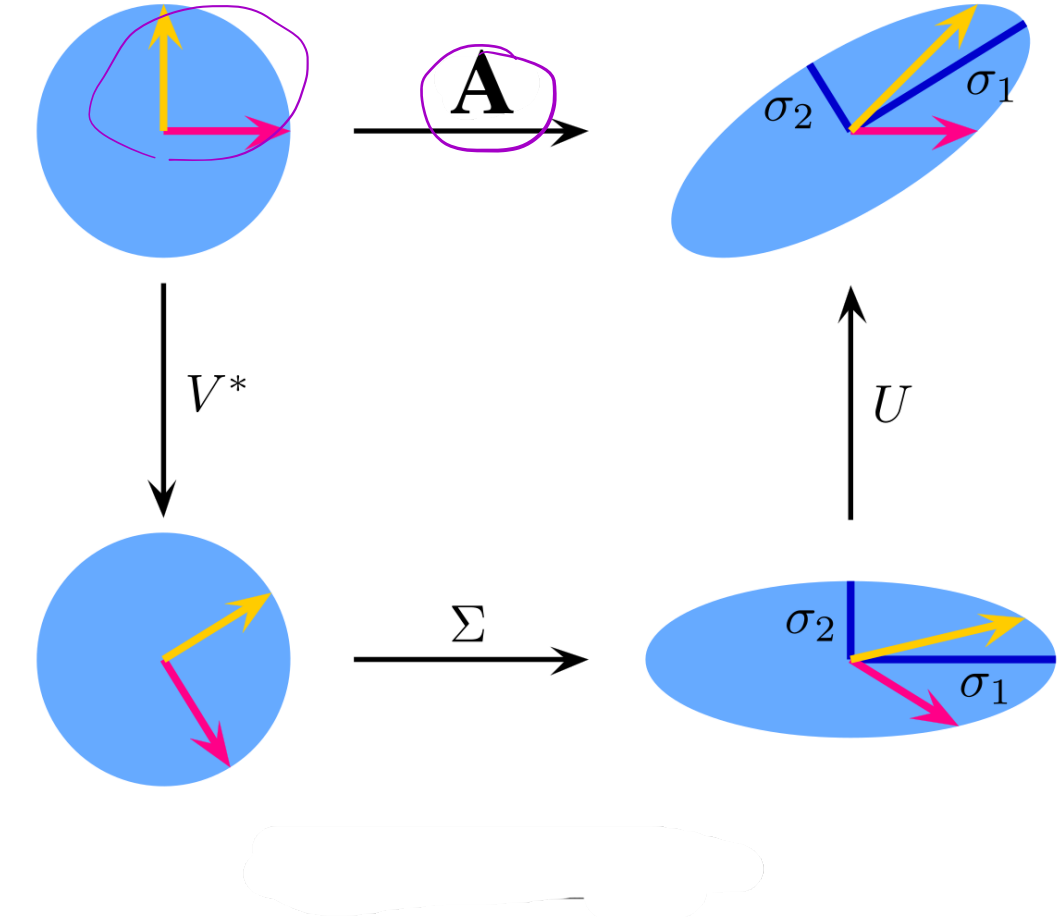
$$A = U \Sigma V^H,$$

where

$$U \in \mathbb{C}^{m \times m} \text{ is unitary} \tag{8}$$
$$V \in \mathbb{C}^{n \times n} \text{ is unitary} \tag{9}$$
$$\Sigma \in \mathbb{R}^{m \times n} \text{ is diagonal.} \tag{10}$$

Additionally, the diagonal entries σ_j of Σ are non-negative and in non-decreasing order so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p \in \min(m, n)$.

- The diagonal matrix Σ is real and has the same shape as A even when A is not square
- The matrices U and V are always square



[Edited from figure by Georg-Johann, Wikipedia]

The image of the unit sphere under a map A is a hyperellipse (in $\mathbb{R}^m$). Thus, with $A = U\Sigma V^H$, have

- The unitary map V^H preserves the sphere (rotating a sphere is a sphere)
- The diagonal matrix Σ stretches the sphere into a hyperellipse aligned with the canonical basis
- The unitary map U rotates or reflects the hyperellipse without changing shape

Recap: SVD vs. eigenvalue decomposition

SVD expresses a matrix in proper bases for column and row span to represent it as a diagonal matrix

$$A = U\Sigma V^H$$

We have seen something similar with eigenvectors: A non-defective square matrix A can be expressed as a diagonal matrix of eigenvalues Λ if the range and domain are presented in a basis of eigenvectors

$$A = X\Lambda X^{-1}$$

There are fundamental differences between the SVD and eigenvalue decomposition

- SVD uses two different bases (left and right singular vectors); eigenvalue decomposition uses just one (eigenvectors)
- SVD uses orthonormal bases, whereas eigenvalue basis generally is not orthogonal
- Not all matrices (even square ones) have an eigendecomposition, but all matrices (even rectangular ones) have a singular value decomposition

Typically, eigenvalues tell us something about the behavior of iterative processes that involve the matrix A such as A^k and e^{tA}

Singular values tend to tell us something about A itself

Recap: The SVD and matrix properties

In the following:

- The matrix A is of dimension $m \times n$
- $p = \min(m, n)$
- $r \leq p$ is the number of non-zero singular values of A

We now list how the SVD is related to fundamental properties of the matrix A

- The rank of A is r, the number of non-zero singular values.
- The range (column span) of A is $\text{span}(u_1, \ldots, u_r)$, the kernel is $\text{span}(v_{r+1}, \ldots, v_n)$
- $\|A\|_2 = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$
- For square A, $|\det(A)| = \prod_{i=1}^{m} \sigma_i$
- The non-zero singular values of A are the square roots of the non-zero eigenvalues of $A^H A$ and AA^H

Recap: Versions of SVD

Full SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times m}\underbrace{\boldsymbol{\Sigma}}_{m\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Reduced SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times n}\underbrace{\boldsymbol{\Sigma}}_{n\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Rank-revealing SVD of a rank r matrix with dimension $m \times n$

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times r}\underbrace{\boldsymbol{\Sigma}}_{r\times r}\underbrace{\boldsymbol{V}^H}_{r\times n}$$

Recap: SVD for low-rank approximation

Consider the SVD $A = U\Sigma V^H$, then

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^H$$

Let us truncate the sum after $1 \leq q \leq r$ terms and define

$$A_q = \sum_{j=1}^{q} \sigma_j u_j v_j^H.$$

Then, A_q is a best rank q approximation of A in the $\|\cdot\|_2$ norm

$$\|A - A_q\|_2 = \inf_{\substack{B \in \mathbb{C}^{m\times n} \\ \text{rank}(B) \leq q}} \|A - B\|_2 = \sigma_{q+1}$$

The error is the first left out singular value σ_{q+1}!

Recap: SVD and pseudo inverse

The (Moore-Penrose) pseudo inverse of an $m \times n$ matrix that is regular and square is $A^+ = A^{-1}$.

Otherwise, the pseudo inverse is given by

$$A^+ = V \Sigma^+ U^H,$$

where

$$\Sigma^+ = \text{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$$

Thus, we can solve least-squares problems $\min_x \|b - Ax\|_2$ by taking the SVD of A, computing A^+, and setting $x = A^+ b$ for the minimal norm solution w.r.t. $\|\cdot\|_2$

The approach we discussed via the QR decomposition is cheaper but the approach via the SVD is sometimes preferred because it allows to easily regularize the problem by truncating small singular values.

How to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea?

[†] https://nhigham.com/2022/10/11/seven-sins-of-numerical-linear-algebra/

How **not** to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea? $\rightsquigarrow$ as we have seen before, it is typically dangerous[†] from a stability perspective to compute something of the matrix A via the matrix $A^H A$ (think of least-squares regression) $\rightsquigarrow$ board

[†] https://nhigham.com/2022/10/11/seven-sins-of-numerical-linear-algebra/

- Perturbation of EV of $A^H A$

$$|\lambda_k(A^H A + \delta B) - \lambda_k(A^H A)| \leq \|\delta B\|_2 \quad (**)$$

- singular values

$$|\sigma_k(A + \delta A) - \sigma_k(A)| \leq \|\delta A\|_2 \quad (*)$$

Backward stable algo. for SV

$$\tilde{\sigma}_k = \sigma_k(A + \delta A), \quad \frac{\|\delta A\|}{\|A\|} \in O(\varepsilon)$$

$$|\tilde{\sigma}_k - \sigma_k| \leq \|\delta A\| \text{ with } \|\delta A\| \in O(\varepsilon \|A\|)$$

Now compute $\lambda_k(A^H A)$

$$\tilde{\lambda}_k = \lambda_k(A^H A + \delta B), \quad \frac{\|\delta B\|}{\|A^H A\|} \in O(\varepsilon)$$

$$|\tilde{\lambda}_k - \lambda_k| \leq \|\delta B\| \in O(\varepsilon \|A^H A\|)$$
$$= O(\varepsilon \|A\|^2)$$

- Compute $\tilde{\sigma}_q = \sqrt{\tilde{\lambda}_q}$

$$|\tilde{\sigma}_q - \sigma_q| = |\sqrt{\tilde{\lambda}_q} - \sqrt{\lambda_q}| \leq \frac{|\tilde{\lambda}_q - \lambda_q|}{\underbrace{\sqrt{\tilde{\lambda}_q}}_{\geq 0} + \underbrace{\sqrt{\lambda_q}}_{=\sigma_q}}$$

$|\tilde{\underline{\sigma}}_q - \underline{\sigma}_q|$ behaves as $O\left(\varepsilon \frac{\|A\|^2}{\sigma_q}\right)$

How we actually compute the SVD

For the sake of the argument, assume that A is a square $m \times m$ (following is applicable to rectangular matrices too) $\rightsquigarrow$ board

$$H = \begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

$$A = U \Sigma V^H$$

$$AV = U\Sigma$$

$$A^H U = V \Sigma^H U^H U = V \underline{\Sigma}^H = V\Sigma$$

$$\underbrace{\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} V & V \\ U & -U \end{bmatrix}}_{X} = \begin{bmatrix} A^H U & -A^H U \\ AV & AV \end{bmatrix}$$

$$= \begin{bmatrix} V\Sigma & -V\Sigma \\ U\Sigma & U\Sigma \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

$$\Lambda$$

How we actually compute the SVD

For the sake of the argument, assume that A is a square $m \times m$ (following is applicable to rectangular matrices too) $\rightsquigarrow$ board

Consider the $2m \times 2m$ Hermitian matrix

$$H = \begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

Since $A = U\Sigma V^H$, we have

$$AV = U\Sigma \text{ and } A^H U = V\Sigma^H = V\Sigma \quad \text{(recall that singular values are real)}$$

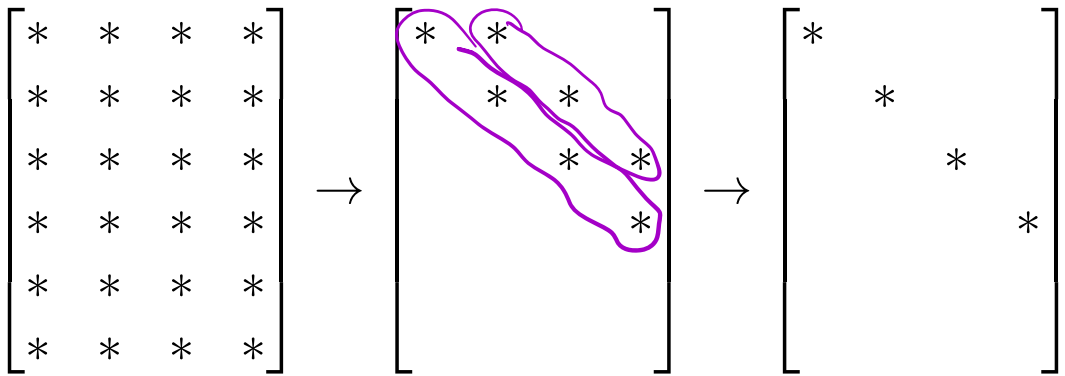
which we write in matrix form as

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

which is the eigendecomposition of H $\rightsquigarrow$ avoids using $A^H A$ and AA^H and is stable

Computing the SVD is typically a two-phase procedure:

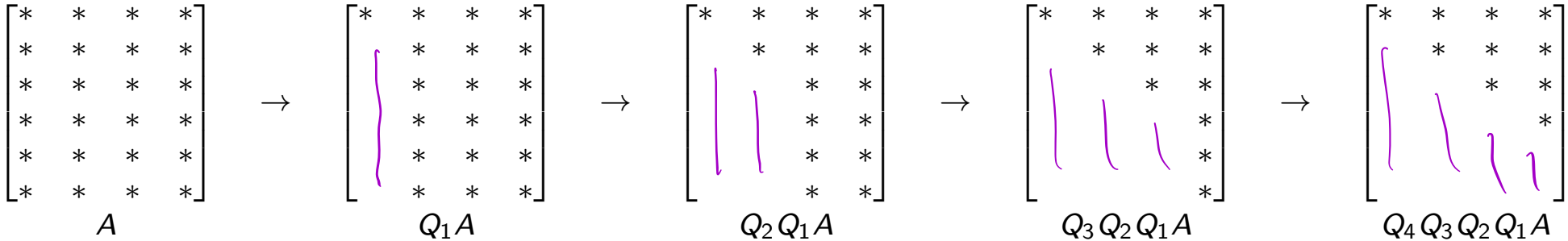
First, reduce the matrix A to bidiagonal form, then diagonalize the bidiagonal matrix



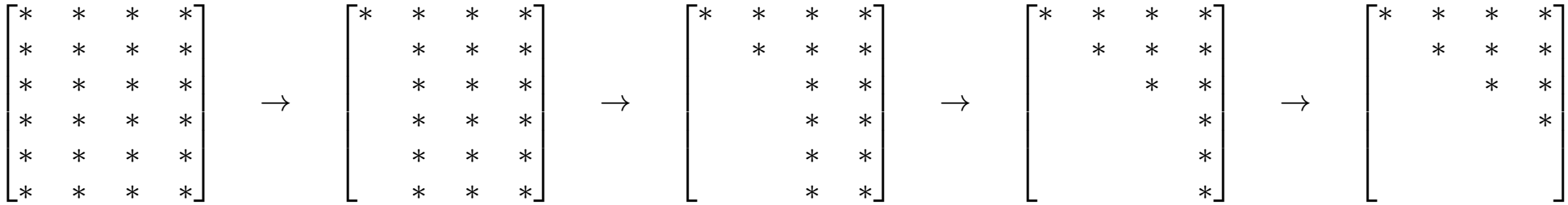
- Phase 1 involves a finite number of operations that scale as $\mathcal{O}(mn^2)$
- Phase 2 (recall eigenvalue problems) is iterative but converges very quickly; in practice achieves convergence in $\mathcal{O}(n)$ to machine precision
- Thus, state-of-the-art computation of the SVD has costs that scale as $\mathcal{O}(mn^2)$

This is the celebrated Golub-Kahan approach from the 1960s

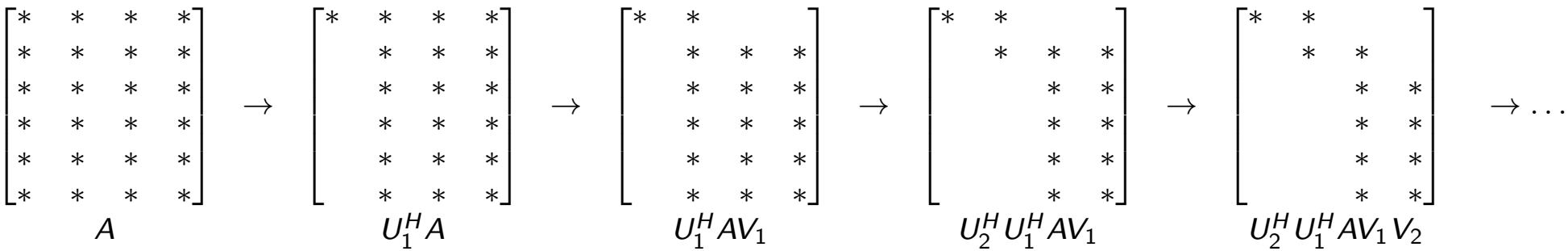
How can we bidiagonalize a matrix? Recall that we already know how to triangualize a matrix via (unitary) Householder reflection



How can we bidiagonalize a matrix? Recall that we already know how to triangulize a matrix via (unitary) Householder reflection



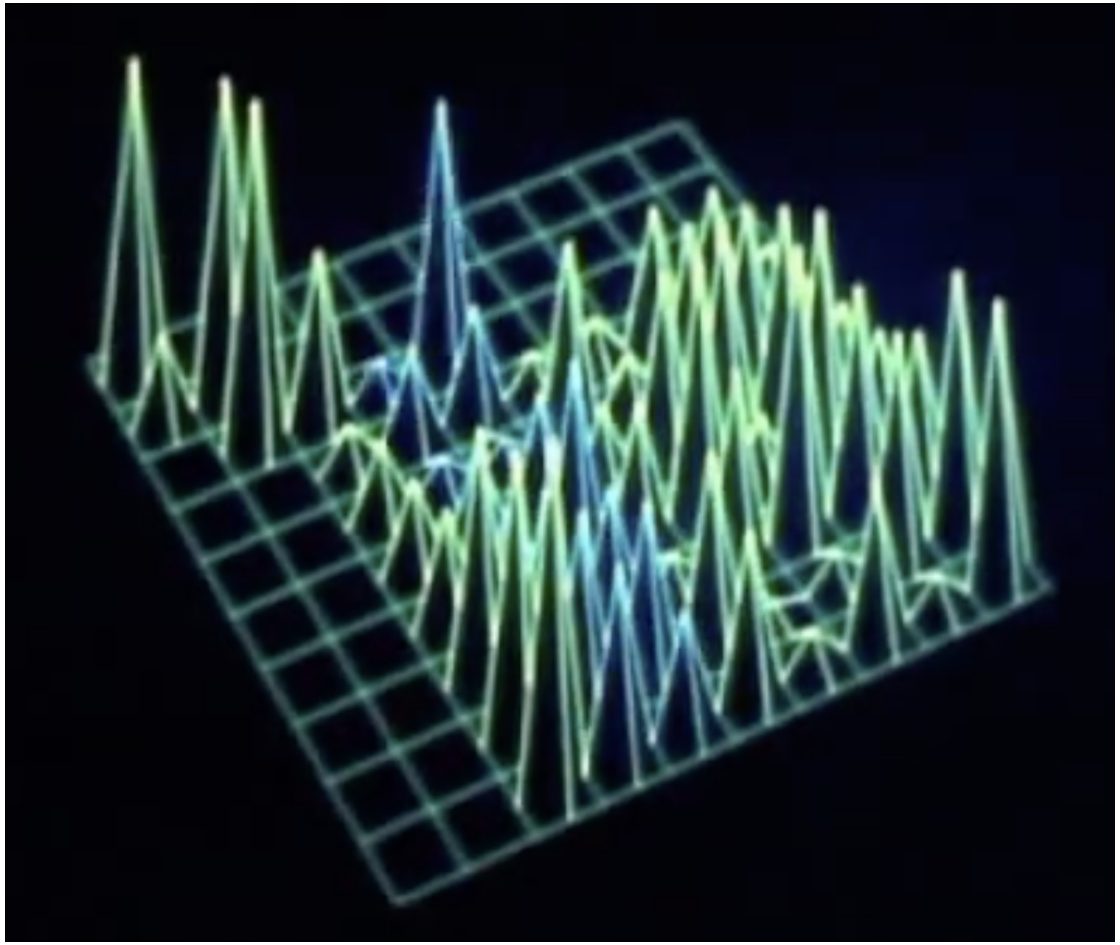
Now, we apply interleaved Householder reflection from left and right



At the end, n reflectors are applied from the left and $n-2$ from the right

A variant of the QR algorithm is then applied to the bidiagonal matrix (or other eigendecomposition algorithms) $\rightsquigarrow$ details in Golub et al., Matrix Computations.

Visualizing SVD computation



Video: http://youtu.be/R9UoFyqJca8

SVD saves the universe

The first Star Trek movie came out in 1979. The producers had asked Los Alamos for computer graphics to run on the displays on the bridge of the Enterprise:

[Figure: Paramount Pictures]

Link: https://blogs.mathworks.com/cleve/2012/12/10/1976-matrix-singular-value-decomposition-film/

Matlab

```
>> X = randn(10, 4);
>> [U, S, V] = svd(X); % full SVD
>> [Ur, Sr, Vr] = svd(X, 0); % reduced (economic) SVD
>> size(S)
>> size(Sr)
ans =
     10      4
ans =
      4      4
```

In most cases, we want the reduced SVD and then we should explicitly compute it:

```
>> X = randn(10000, 50);
>> tic; [U, S, V] = svd(X); toc
Elapsed time is 6.746243 seconds.
>> tic; [U, S, V] = svd(X, 0); toc
Elapsed time is 0.064572 seconds.
```

For very large and sparse matrices, or if the matrix is unavailable and we only have a procedure that returns the matrix-vector product, we can iteratively compute the first few singular vectors via `svds`

```
>> X = gallery('poisson', 100);
>> whos X
Name           Size                     Bytes  Class
    Attributes

  X          10000x10000                1033608  double
      sparse

>> tic; [U, S, V] = svds(X, 1); toc

Elapsed time is 0.307015 seconds.
```

SVD is great because its computation is (very) stable $\rightsquigarrow$ building block for computing many basic linear algebra quantities

type 'edit rank.m' in Matlab

```
 1: function r = rank(A,tol)
 2: %RANK   Matrix rank.
 3: %   RANK(A) provides an estimate of the number of linearly
 4: %   independent rows or columns of a matrix A.
 5: %
 6: %   RANK(A,TOL) is the number of singular values of A
 7: %   that are larger than TOL. By default, TOL = max(size(A)) * eps(norm(A)
        ).
 8: %
 9: %   Class support for input A:
10: %      float: double, single
11:
12: %   Copyright 1984-2015 The MathWorks, Inc.
13:
14: s = svd(A);
15: if nargin==1
16:     tol = max(size(A)) * eps(max(s));
17: end
18: r = sum(s > tol);
```

type 'edit pinv.m' in Matlab

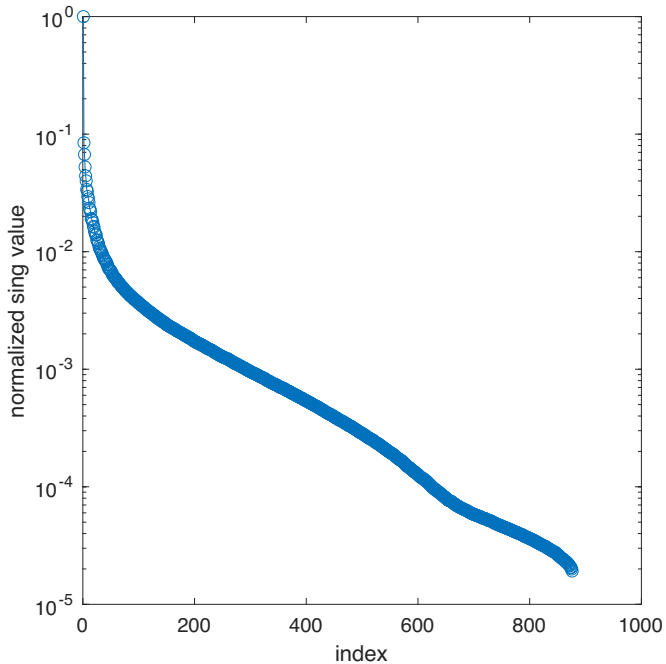
```
function X = pinv(A,tol)
%PINV    Pseudoinverse.
%   X = PINV(A) produces a matrix X of the same dimensions
%   as A' so that A*X*A = A, X*A*X = X and A*X and X*A
%   are Hermitian. The computation is based on SVD(A) and any
%   singular values less than a tolerance are treated as zero.
%
%   PINV(A,TOL) treats all singular values of A that are less than TOL as
%   zero. By default, TOL = max(size(A)) * eps(norm(A)).
%
%   Class support for input A:
%      float: double, single
%
%   See also RANK.

%   Copyright 1984-2015 The MathWorks, Inc.

[U,S,V] = svd(A,'econ');
s = diag(S);
if nargin < 2
    tol = max(size(A)) * eps(norm(s,inf));
end
r1 = sum(s > tol)+1;
V(:,r1:end) = [];
U(:,r1:end) = [];
s(r1:end) = [];
s = 1./s(:);
X = (V.*s.')*U';
```

type 'edit orth.m' in Matlab

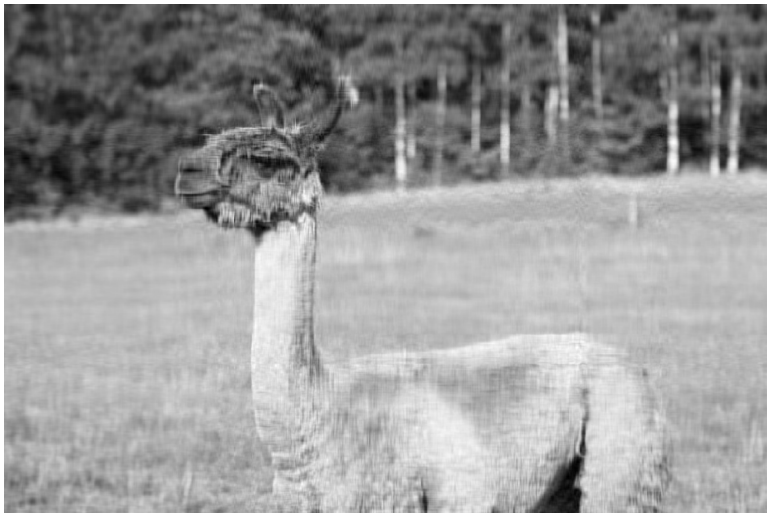
```
function Q = orth(A)
%ORTH   Orthogonalization.
%   Q = ORTH(A) is an orthonormal basis for the range of A.
%   That is, Q'*Q = I, the columns of Q span the same space as
%   the columns of A, and the number of columns of Q is the
%   rank of A.
%
%   Class support for input A:
%      float: double, single
%
%   See also SVD, RANK, NULL.

%   Copyright 1984-2015 The MathWorks, Inc.

[Q,S] = svd(A,'econ'); %S is always square.
s = diag(S);
tol = max(size(A)) * eps(max(s));
r = sum(s > tol);
Q(:, r+1:end) = [];
```

Application of SVD for image compression

```
1: >> A = rgb2gray(imread('llama.jpg'));
2: >> figure; imshow(A);
3: >> [U, S, V] = svd(double(A));
4: >> figure; semilogy(diag(S)/S(1, 1), '-o');
5: >> xlabel('index');ylabel('normalized sing value');
6:
7: >> r = 50;
8: >> Aapprx = U(:, 1:r)*S(1:r, 1:r)*V(:, 1:r)';
9: >> figure; imshow(uint8(Aapprx));
```

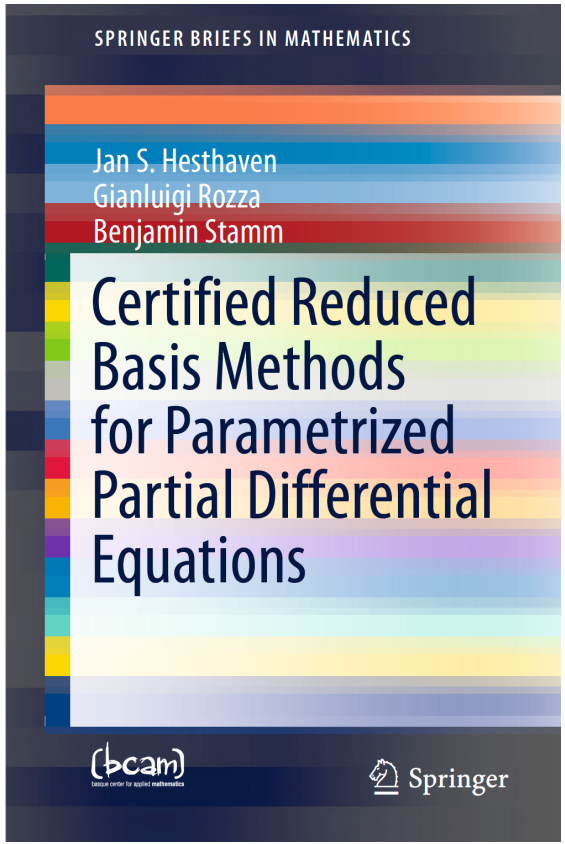


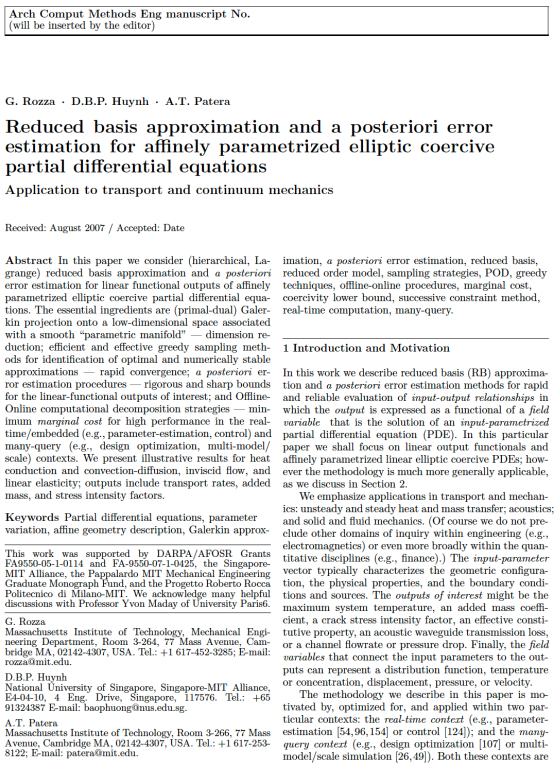
Original picture and singular values



Original picture (left) and reduced picture (right)

Outlook: Model reduction and latent dynamics





Arch Comput Methods Eng manuscript No.
(will be inserted by the editor)

G. Rozza · D.B.P. Huynh · A.T. Patera

Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations

Application to transport and continuum mechanics

Received: August 2007 / Accepted: Date

Abstract In this paper we consider (hierarchical, Lagrange) reduced basis approximation and *a posteriori* error estimation for linear functional outputs of affinely parametrized elliptic coercive partial differential equations. The essential ingredients are (primal-dual) Galerkin projection onto a low-dimensional space associated with a smooth "parametric manifold" — dimension reduction; efficient and effective greedy sampling methods for identification of optimal and numerically stable approximations — rapid convergence; *a posteriori* error estimation procedures — rigorous and sharp bounds for the linear-functional outputs of interest; and Offline-Online computational decomposition strategies — minimum *marginal cost* for high performance in the real-time/embedded (e.g., parameter-estimation, control) and many-query (e.g., design optimization, multi-model/scale) contexts. We present illustrative results for heat conduction and convection-diffusion, inviscid flow, and linear elasticity; outputs include transport rates, added mass, and stress intensity factors.

Keywords Partial differential equations, parameter variation, affine geometry description, Galerkin approximation, *a posteriori* error estimation, reduced basis, reduced order model, sampling strategies, POD, greedy techniques, offline-online procedures, marginal cost, coercivity lower bound, successive constraint method, real-time computation, many-query.

This work was supported by DARPA/AFOSR Grants FA9550-05-1-0114 and FA-9550-07-1-0425, the Singapore-MIT Alliance, the Pappalardo MIT Mechanical Engineering Graduate Monograph Fund, and the Progetto Roberto Rocca Politecnico di Milano-MIT. We acknowledge many helpful discussions with Professor Yvon Maday of University Paris6.

G. Rozza
Massachusetts Institute of Technology, Mechanical Engineering Department, Room 3-264, 77 Mass Avenue, Cambridge MA, 02142-4307, USA. Tel.: +1 617-452-3285; E-mail: rozza@mit.edu.

D.B.P. Huynh
National University of Singapore, Singapore-MIT Alliance, E4-04-10, 4 Eng. Drive, Singapore, 117576. Tel.: +65 91324387 E-mail: bsophuong@nus.edu.sg.

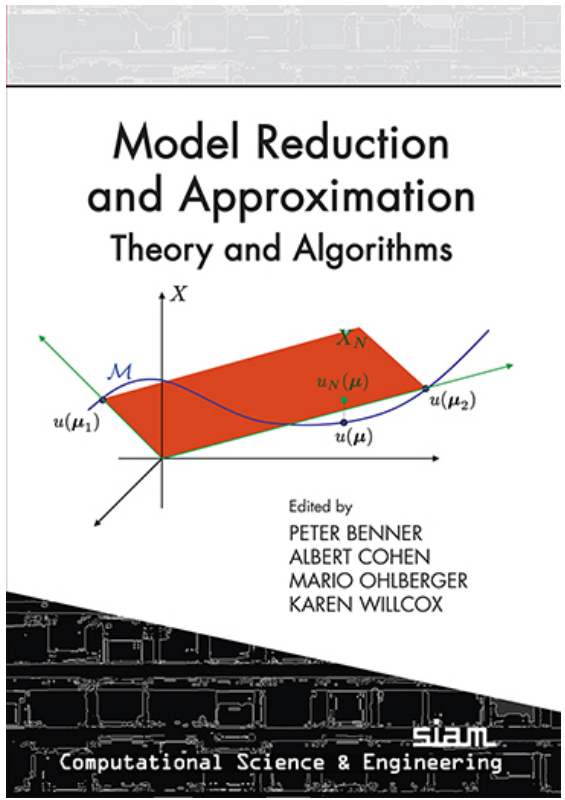
A.T. Patera
Massachusetts Institute of Technology, Room 3-266, 77 Mass Avenue, Cambridge MA, 02142-4307, USA. Tel.: +1 617-253-8122; E-mail: patera@mit.edu.

1 Introduction and Motivation

In this work we describe reduced basis (RB) approximation and *a posteriori* error estimation methods for rapid and reliable evaluation of *input-output relationships* in which the *output* is expressed as a functional of a *field variable* that is the solution of an *input-parametrized* partial differential equation (PDE). In this particular paper we shall focus on linear output functionals and affinely parametrized linear elliptic coercive PDEs; however the methodology is much more generally applicable, as we discuss in Section 2.

We emphasize applications in transport and mechanics: unsteady and steady heat and mass transfer; acoustics; and solid and fluid mechanics. (Of course we do not preclude other domains of inquiry within engineering (e.g., electromagnetics) or even more broadly within the quantitative disciplines (e.g., finance).) The *input-parameter* vector typically characterizes the geometric configuration, the physical properties, and the boundary conditions and sources. The *outputs of interest* might be the maximum system temperature, an added mass coefficient, a crack stress intensity factor, an effective constitutive property, an acoustic waveguide transmission loss, or a channel flowrate or pressure drop. Finally, the *field variables* that connect the input parameters to the outputs can represent a distribution function, temperature or concentration, displacement, pressure, or velocity.

The methodology we describe in this paper is motivated by, optimized for, and applied within two particular contexts: the *real-time context* (e.g., parameter-estimation [54,96,154] or control [124]); and the *many-query context* (e.g., design optimization [107] or multi-model/scale simulation [26,49]). Both these contexts are



Iterative methods for solving systems of linear equations

Why iterative methods for linear systems?

- Costs of solving a linear system with direct method are $\mathcal{O}(n^3)$. This is too much! If n gets large, then n^3 is huge and costs become intractable.

- History of matrix computations over the years (according to Trefethen & Bau)
  1950: $m = 20$
  1965: $m = 200$
  1980: $m = 2000$
  1995: $m = 20000$
  This is an increase of a factor 10^3. However, computing power (FLOP/sec) increased by about 10^9. Notice that $(10^3)^3 = 10^9$, which reflects the $\mathcal{O}(n^3)$ bottleneck

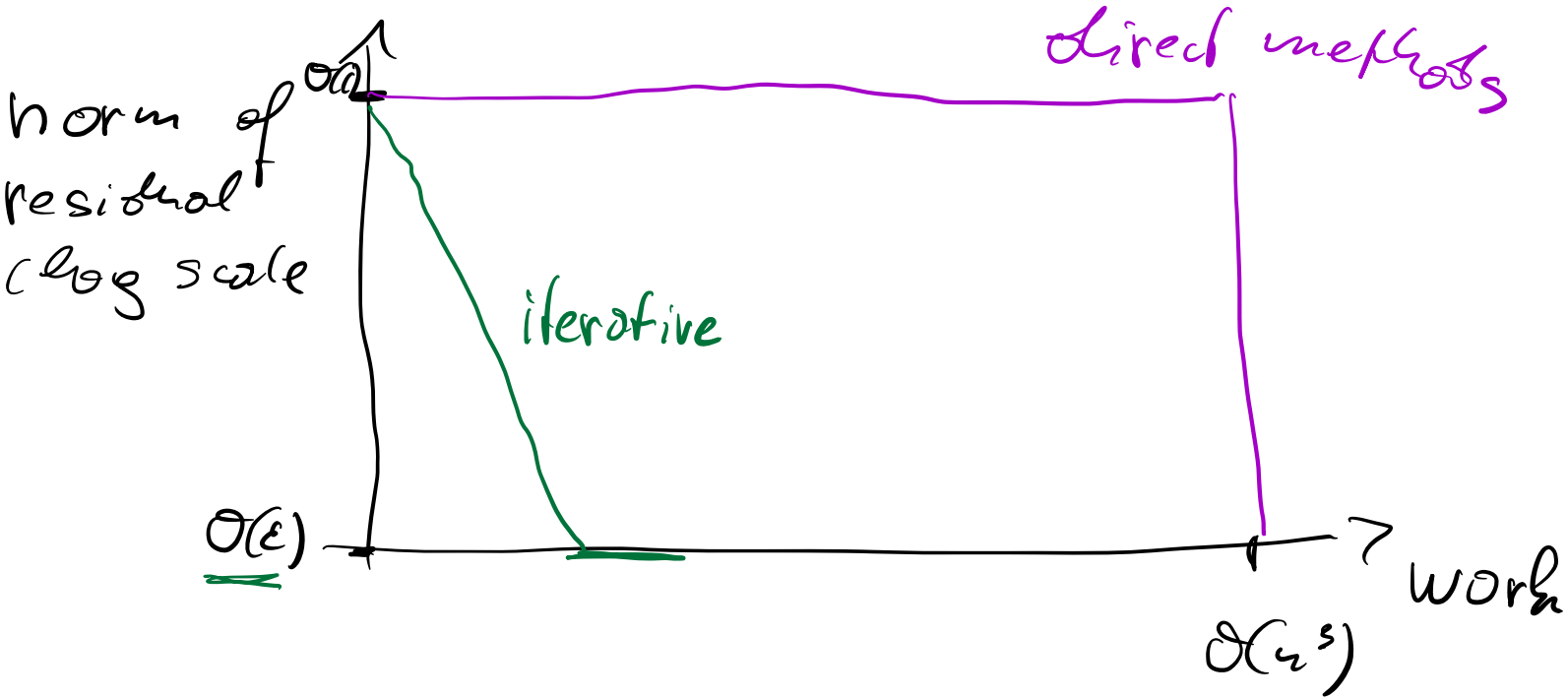
- (Side remark: There are direct methods that beat the complexity $\mathcal{O}(n^3)$ (think of Strassen's algorithm); however,

Why iterative methods for linear systems?

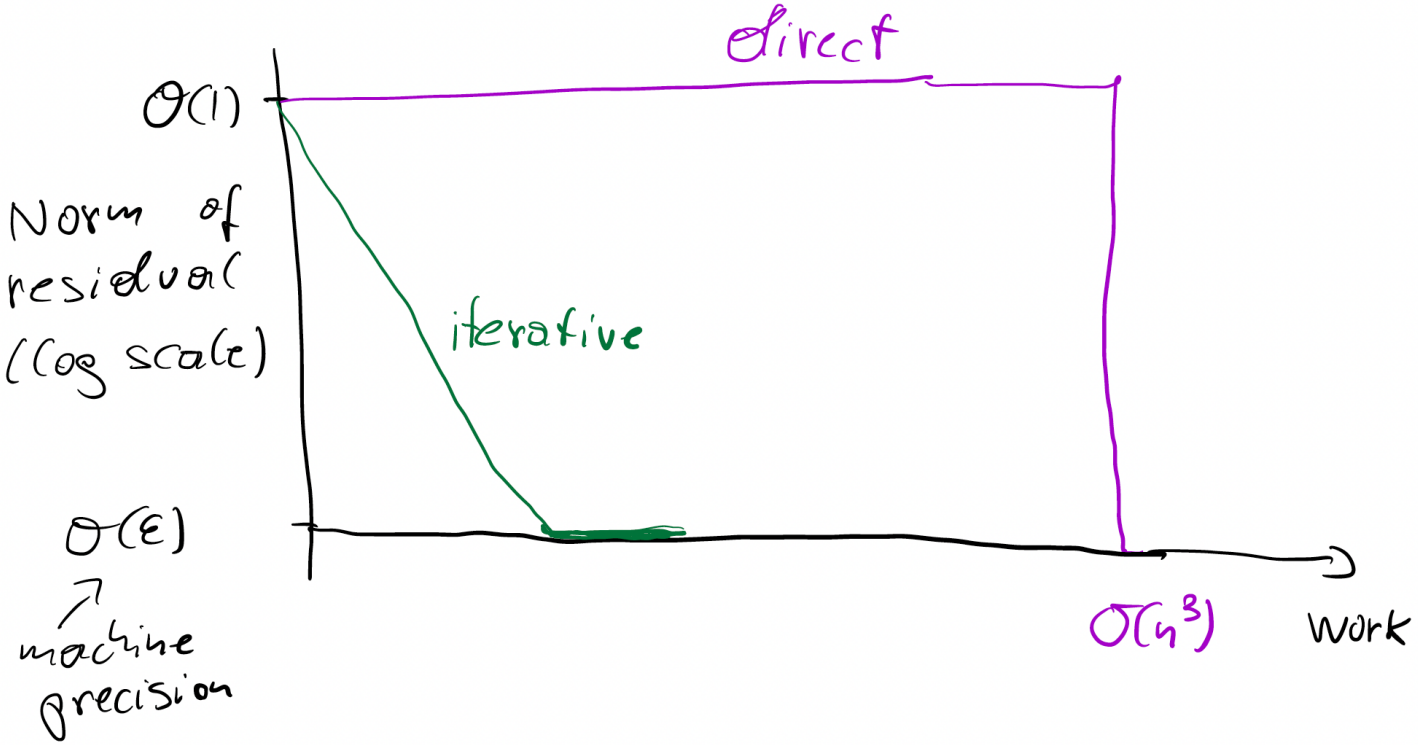
- Costs of solving a linear system with direct method are $\mathcal{O}(n^3)$. This is too much! If n gets large, then n^3 is huge and costs become intractable.

- History of matrix computations over the years (according to Trefethen & Bau)
  1950: $m = 20$
  1965: $m = 200$
  1980: $m = 2000$
  1995: $m = 20000$
  This is an increase of a factor 10^3. However, computing power (FLOP/sec) increased by about 10^9. Notice that $(10^3)^3 = 10^9$, which reflects the $\mathcal{O}(n^3)$ bottleneck

- (Side remark: There are direct methods that beat the complexity $\mathcal{O}(n^3)$ (think of Strassen's algorithm); however, the numerical stability of these algorithms is not well understood and the constants hidden in the complexity results are huge so that we "never" see the improved rate in practice.)



- Iterative methods *can* converge geometrically until residual is below machine precision
- Direct methods make no progress at all until $\mathcal{O}(n^3)$ work is done, and then lead to residual on the order of machine precision



- Iterative methods *can* converge geometrically until residual is below machine precision
- Direct methods make no progress at all until $\mathcal{O}(n^3)$ work is done, and then lead to residual on the order of machine precision

Why iterative methods for linear systems? (cont'd)

- Classical direct methods (e.g., Gauss elimination) follow the pattern of taking $\mathcal{O}(n)$ steps and each step costs $\mathcal{O}(n^2)$ but don't exploit properties of the matrix
- Iterative methods reduce the number of steps and the costs of each step, depending on properties of the problem at hand (e.g., spectral properties of A in $Ax = b$)
- The ideal iterative method requires $\mathcal{O}(1)$ steps and $\mathcal{O}(n)$ costs per step to reach machine precision (think of multigrid and pre-conditioned CG)
- Iterative methods, even in the absence of rounding errors, do not deliver the exact answer (recall that LU, QR give us exact answer in finite number of steps if we are in exact arithmetic).
- However, we only can be as accurate as machine precision anyway if we do calculations on a computer. Furthermore, often matrices are stemming from discretizations of PDEs and then we need the solution only up to the discretization error.

Iterative solution of linear systems

Target problems: very large ($n = 10^5, 10^6, \ldots$), A is usually sparse and has specific properties.

To solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

we construct a sequence

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

of iterates that converges fast to the solution $\boldsymbol{x}$, where $\boldsymbol{x}_{k+1}$ can be cheaply computed from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ (e.g., one matrix-vector multiplication).

Thought experiment: If we can compute one iteration with cost $\mathcal{O}(n)$ (e.g., one matrix-vector multiplication with a sparse matrix) and need a constant $\mathcal{O}(1)$ number of iterations to reach desired precision, then we solve $Ax = b$ with costs $\mathcal{O}(n)$. Intuitively, we cannot do better than that because we solve for n quantities and thus need to touch each at least once.

Prototype of iterative method

Let's start by trying to write $\boldsymbol{Ax} = \boldsymbol{b}$ more generally as

$$\boldsymbol{x} = f(\boldsymbol{x})$$

For example set $f(\boldsymbol{x}) = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}$ to obtain

$$\boldsymbol{x} = f(\boldsymbol{x}) = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}$$

We can now try to solve this via a fixed-point iteration

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

which is in our case

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

Instead of picking a specific f, let's look at the prototype

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

where $\boldsymbol{G}$ is an iteration matrix somehow related to $\boldsymbol{A}$ and $\boldsymbol{c}$ is related to $\boldsymbol{b}$.

We must have the actual solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ as a unique fixed point of the iteration:

$$\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c}$$

And we need that the sequence $(\boldsymbol{x}_k)$ converges

Theorem: The fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with an invertible G converges for each starting point $\boldsymbol{x}_o$ if and only if

$$\rho(G) < 1,$$

where $\rho(G) = \max_j |\lambda_j|$ is the largest magnitude of all eigenvalue of G (i.e., the spectral radius).

$\rightsquigarrow$ proof

G is real, sym

$$Q G Q^T = D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

Because $|\lambda_i| \leq \rho(G) < 1$

$$\lim_{k \to \infty} \underbrace{\begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix}}_{D^k} = 0$$

$$\lim_{k \to \infty} G^k = \lim_{k \to \infty} Q D^k Q^{-1} = 0$$

$m \geq 1$

$$\|x_{k+m} - x_k\| = \dots = \|G^k (x_m - x_0)\|$$

$k \to \infty$ $\longrightarrow 0$

Often want $\|G\| < 1$, which implies

$$\rho(G) < 1$$

x ← fix point

$$\|x_k - x\| \leq \|G\|^k \|(x_0 - x)\|$$

$\Rightarrow$ smaller $\|G\|$ means faster convergence

Theorem: The fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with an invertible G converges for each starting point $\boldsymbol{x}_o$ if and only if

$$\rho(G) < 1,$$

where $\rho(G) = \max_j |\lambda_j|$ is the largest magnitude of all eigenvalue of G (i.e., the spectral radius).

$\rightsquigarrow$ proof

In particular, for any induced matrix norm $\|\cdot\|$ we have $\rho(G) \leq \|G\|$ and thus $\|G\|$ is a sufficient criterion for convergence. We then obtain

$$\|x_k - x\| \leq \|G\|^k \|x_0 - x\|$$

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is stationary point

$$\boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x} - \boldsymbol{Q}^{-1}\boldsymbol{b} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x}$$

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (recall: $\boldsymbol{Q}$ needs to be invertible)?

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (recall: $\boldsymbol{Q}$ needs to be invertible)?

Choose $Q = A$, then our iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}\,, \qquad \boldsymbol{G} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}\,, \quad \boldsymbol{c} = \boldsymbol{Q}^{-1}\boldsymbol{b}$$

becomes

$$(I - A^{-1}A)\boldsymbol{x}_k + \underbrace{A^{-1}\boldsymbol{b}}_{\boldsymbol{x}} = \boldsymbol{x}_{k+1}$$

$$\boldsymbol{0} + \boldsymbol{x} = \boldsymbol{x}_{k+1}$$

and we are done in just a single step

$$x_{k+1} = x$$

Thus, if we "know the solution" (in form of having the inverse A^{-1}) then no further work is needed here because we already did all the work when finding A^{-1}

The other extreme is setting $Q = I$, this leads to the Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

We have invested zero costs in finding Q (and Q^{-1}) and so we can expect that $Q = I$ will require high costs in terms of number of iterations to converge in general, if it converges at all

When does Richardson method converge? $\rightsquigarrow$ board

Let A spd, then

$$\rho(G) = \rho(I - wA) = \max\{|1 - \lambda_{max}(A)|, |1 - \lambda_{min}(A)|\}$$

$$\lambda_{max}(A) < 2$$

The Richardson method is consistent (solution is a stationary point) but it may not converge or converge very slowly

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

What could we do instead of Richardson method?

The Richardson method is consistent (solution is a stationary point) but it may not converge or converge very slowly

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

What could we do instead of Richardson method?
Let us think of $\boldsymbol{Q}$ as a preconditioner of the Richardson method:

$$\boldsymbol{Q}^{-1} \approx \boldsymbol{A}^{-1}$$

and transform

$$Q^{-1}Ax = Q^{-1}b$$

Applying the Richardson iteration to this modified system gives

$$x_{k+1} = (I - Q^{-1}A)x_k + Q^{-1}b = Gx_k + c$$

with an iteration matrix $I - Q^{-1}A \approx 0$ and thus typically more rapid convergence

Common choices for Q

Requirements on a good Q are

Common choices for Q

Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Common choices for Q

Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Split the matrix A as follows

$$A \quad = \quad L \quad + \quad D \quad + \quad U$$

Jacobi method

Theorem: Select now $Q = D$ ... Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

$\rightsquigarrow$ board

$$A = L + D + R$$

$$x_{k+1} = (I - D^{-1}A)\,x_k + D^{-1}b$$

$$= (I - D^{-1}(L + D + R))\,x_k + D^{-1}b$$

$$= \underbrace{-D^{-1}(L+R)}_{G}\,x_k + D^{-1}b$$

$$\rho(-D^{-1}(L+R)) = \rho(D^{-1}(L+R))$$

$$\leq \|D^{-1}(L+R)\|_\infty$$

$$= \max_i \left| \sum_{i \neq j} \frac{|a_{ij}|}{|a_{ii}|} \right| < 1$$

↑ Diagonal dom.

Jacobi method

Theorem: Select now $Q = D$ ... Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

$\rightsquigarrow$ board

Notice that $Q = D$ is a good choice in terms of computational costs because we can very quickly solve $Dy = z$ for the diagonal matrix D

Gauss-Seidel method

Theorem: Choose $Q = D + L$ . . . Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).

$\rightsquigarrow$ board

Gauss-Seidel method

Theorem: Choose $Q = D + L$ ... Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).
$\rightsquigarrow$ board

Notice that $Q = D + L$ is a good choice in terms of computational costs because we can very quickly solve $(D + L)y = z$ for the lower triangular matrix $D + L$ (forward substitution)