Numerical methods for matrix functions

SF2524 - Matrix Computations for Large-scale Systems

Lecture 14: Specialized methods

Specialized methods

- Matrix exponential - scaling-and-squaring
  - Matlab: `expm(A)`
  - Julia: `exp(A)`
- Matrix square root
  - Matlab: `sqrtm(A)`
  - Julia: `sqrt(A)`
- Matrix sign function

Matrix exponential

PDF Lecture notes 4.3.1

* $\exp(A + B)$ properties on board *

From basic properties of matrix functions:

* $\exp(A+B)$ properties on board *

$$\exp(A) = \exp(A/2)\exp(A/2).$$

From basic properties of matrix functions: * $\exp(A+B)$ properties on board *

$$\exp(A) = \exp(A/2)\exp(A/2).$$

Repeat:

$$\exp(A) = \exp(A/4)\exp(A/4)\exp(A/4)\exp(A/4).$$

From basic properties of matrix functions: * $\exp(A+B)$ properties on board *

$$\exp(A) = \exp(A/2)\exp(A/2).$$

Repeat:

$$\exp(A) = \exp(A/4)\exp(A/4)\exp(A/4)\exp(A/4).$$

...

For any j

$$\exp(A) = \left(\exp(A/2^j)\right)^{2^j}$$

From basic properties of matrix functions: * $\exp(A+B)$ properties on board *

$$\exp(A) = \exp(A/2)\exp(A/2).$$

Repeat:

$$\exp(A) = \exp(A/4)\exp(A/4)\exp(A/4)\exp(A/4).$$

...

For any j

$$\exp(A) = \left(\exp(A/2^j)\right)^{2^j}$$

Repeated squaring

Given $C = \exp(A/2^j)$, we can compute $\exp(A)$ with j matrix-matrix multiplications: $C_0 = C$

$$C_i = C_{i-1}^2, \quad i = 1, \ldots, j$$

We have $C_j = \exp(A)$.

From basic properties of matrix functions: * $\exp(A+B)$ properties on board *

$$\exp(A) = \exp(A/2)\exp(A/2).$$

Repeat:

$$\exp(A) = \exp(A/4)\exp(A/4)\exp(A/4)\exp(A/4).$$

...

For any j

$$\exp(A) = \left(\exp(A/2^j)\right)^{2^j}$$

Repeated squaring

Given $C = \exp(A/2^j)$, we can compute $\exp(A)$ with j matrix-matrix multiplications: $C_0 = C$

$$C_i = C_{i-1}^2, \quad i = 1, \ldots, j$$

We have $C_j = \exp(A)$. * Julia: squaring property *

Computing $\exp(A/m)$

How to compute $\exp(A/m)$, where $m = 2^j$ for large m?

Computing $\exp(A/m)$

How to compute $\exp(A/m)$, where $m = 2^j$ for large m?
Note: $\|\frac{1}{m}A\| \ll 1$ when m is large.

Computing $\exp(A/m)$

How to compute $\exp(A/m)$, where $m = 2^j$ for large m?
Note: $\|\frac{1}{m}A\| \ll 1$ when m is large.
Use approximation of matrix exponential which is good close to origin.

Computing $\exp(A/m)$

How to compute $\exp(A/m)$, where $m = 2^j$ for large m?
Note: $\|\frac{1}{m}A\| \ll 1$ when m is large.
Use approximation of matrix exponential which is good close to origin.

Idea 0: Naive

Use Truncated Taylor with expansion $\mu = 0$

$$\exp(B) \approx I + \frac{1}{1!}B + \cdots + \frac{1}{N!}B^N$$

Computing $\exp(A/m)$

How to compute $\exp(A/m)$, where $m = 2^j$ for large m?
Note: $\|\frac{1}{m}A\| \ll 1$ when m is large.
Use approximation of matrix exponential which is good close to origin.

Idea 0: Naive

Use Truncated Taylor with expansion $\mu = 0$

$$\exp(B) \approx I + \frac{1}{1!}B + \cdots + \frac{1}{N!}B^N$$

From Theorem 4.1.2:

$$\mathrm{Error} \sim \|B\|^N = \|A/m\|^N = \|A\|^N/m^N$$

$\Rightarrow$ fast if $m \gg \|A\|$

Idea 1: Better (rational approx)

Use a rational approximation of matrix expoential:

$$\exp(B) \approx N_{pq}(B) D_{pq}(B)^{-1}$$

where $N_{pq} \in P_p$ and $D_{pq} \in P_q$.

Idea 1: Better (rational approx)

Use a rational approximation of matrix expoential:

$$\exp(B) \approx N_{pq}(B) D_{pq}(B)^{-1}$$

where $N_{pq} \in P_p$ and $D_{pq} \in P_q$.

$\exp(z) \approx \frac{\sum_{i=0}^{p} \alpha_i z^i}{\sum_{i=0}^{q} \beta_i z^i}$

Idea 1: Better (rational approx)

Use a rational approximation of matrix expoential:

$$\exp(B) \approx N_{pq}(B) D_{pq}(B)^{-1}$$

where $N_{pq} \in P_p$ and $D_{pq} \in P_q$. $\exp(z) \approx \frac{\sum_{i=0}^{p} \alpha_i z^i}{\sum_{i=0}^{q} \beta_i z^i}$

More precisely, for *Padé approximation* of exponential we have

$$\begin{aligned} N_{pq}(z) &= \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+q)!k!(p-k)!} z^k \\ D_{pq}(z) &= \sum_{k=0}^{q} \frac{(p+q-k)!q!}{(p+q)!k!(q-k)!} (-z)^k. \end{aligned}$$

Parameters p and q can be chosen such that a specific error can be guaranteed.

Idea 1: Better (rational approx)

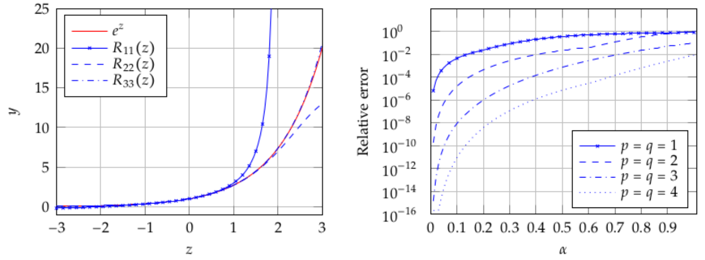
Use a rational approximation of matrix expoential:

$$\exp(B) \approx N_{pq}(B) D_{pq}(B)^{-1}$$

where $N_{pq} \in P_p$ and $D_{pq} \in P_q$. $\exp(z) \approx \frac{\sum_{i=0}^{p} \alpha_i z^i}{\sum_{i=0}^{q} \beta_i z^i}$

More precisely, for *Padé approximation* of exponential we have

$$\sum^{p} \quad (p+q-k)!p!$$



(a) Error in Padé approximation

(b) $\|R_{pq}(\alpha A) - \exp(\alpha A)\| / \|\exp(\alpha A)\|$

Parame … e

guarant

Input: $\delta > 0$ and $A \in \mathbb{R}^{n \times n}$
Output: $F = \exp(A + E)$ where $\|E\|_\infty \leq \delta \|A\|_\infty$.

begin

$j = \max(0, 1 + \text{floor}(\log_2(\|A\|_\infty)))$

$A = A/2^j$

Let q be the smallest non-negative integer such that $\varepsilon(q, q) \leq \delta$.

$D = I; N = I; X = I; c = 1$

for $k = 1 : q$ **do**

$c = c(q - k + 1)/((2q - k + 1)k)$

$X = AX; N = N + cX; D = D + (-1)^k cX$

end

Solve $DF = N$ for F

for $k = 1 : j$ **do**

$F = F^2$

end

end

Algorithm 2: Scaling-and-squaring for the matrix exponential

Why rational approximation?

Why rational approximation?

In general, rational functions is a “richer set of functions”.

Why rational approximation?

In general, rational functions is a "richer set of functions".
Padé approximant (p, q) for exponential has an error of order $p + q + 1$.

Why rational approximation?

In general, rational functions is a "richer set of functions".
Padé approximant (p, q) for exponential has an error of order $p + q + 1$.

A note on computational cost

- Matrix-vector product: $\mathcal{O}(n^2)$ (Exploit in next lecture for $f(A)b$)
- Matrix addition: $\mathcal{O}(n^2)$
- Scalar times matrix: $\mathcal{O}(n^2)$
- Matrix-matrix product: $\mathcal{O}(n^3)$
- Matrix inverse: $\mathcal{O}(n^3)$

Why rational approximation?

In general, rational functions is a "richer set of functions".
Padé approximant (p,q) for exponential has an error of order $p+q+1$.

A note on computational cost

- Matrix-vector product: $\mathcal{O}(n^2)$ (Exploit in next lecture for $f(A)b$)
- Matrix addition: $\mathcal{O}(n^2)$
- Scalar times matrix: $\mathcal{O}(n^2)$
- Matrix-matrix product: $\mathcal{O}(n^3)$
- Matrix inverse: $\mathcal{O}(n^3)$

Padé approximants for exponential (typically $p=q=13$)

$N_{pp}(B) = D_{pp}(-B)$ which gives that

- $N_{pp}(B) = V_{\text{even}}(B) + U_{\text{odd}}(B)$ (13 mat-mat
- $D_{pp}(B) = V_{\text{even}}(B) - U_{\text{odd}}(B)$ + 1 inverse)

Why rational approximation?

In general, rational functions is a "richer set of functions".
Padé approximant (p, q) for exponential has an error of order $p + q + 1$.

A note on computational cost

- Matrix-vector product: $\mathcal{O}(n^2)$ (Exploit in next lecture for $f(A)b$)
- Matrix addition: $\mathcal{O}(n^2)$
- Scalar times matrix: $\mathcal{O}(n^2)$
- Matrix-matrix product: $\mathcal{O}(n^3)$
- Matrix inverse: $\mathcal{O}(n^3)$

Padé approximants for exponential (typically $p = q = 13$)

$N_{pp}(B) = D_{pp}(-B)$ which gives that

- $N_{pp}(B) = \hat{V}_{\text{even}}(B^2) + B \cdot \hat{U}_{\text{odd}}(B^2)$ (7 mat-mat
- $D_{pp}(B) = \hat{V}_{\text{even}}(B^2) - B \cdot \hat{U}_{\text{odd}}(B^2)$ + 1 inverse)

Result: High-degree approximation can be evaluated cheaper than Taylor.

Matrix square root

PDF Lecture notes 4.3.2

Suppose

$$\lambda(A) \cap (-\infty, 0] = \emptyset$$

Suppose

$$\lambda(A) \cap (-\infty, 0] = \emptyset$$

Then, with $f(z) = \sqrt{z}$ the matrix function

$$F = f(A)$$

is well-defined with the Jordan definition or Cauchy definition.

Suppose

$$\lambda(A) \cap (-\infty, 0] = \emptyset$$

Then, with $f(z) = \sqrt{z}$ the matrix function

$$F = f(A)$$

is well-defined with the Jordan definition or Cauchy definition. Moreover,

$$F^2 = A$$

Newton's method for scalar-valued equation:

$$g(x) = x^2 - a = 0$$

Newton's method for scalar-valued equation:

$$g(x) = x^2 - a = 0$$

Simplifies to

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} = \ldots = \frac{1}{2}(x_k + a x_k^{-1})$$

Newton's method for scalar-valued equation:

$$g(x) = x^2 - a = 0$$

Simplifies to

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} = \ldots = \frac{1}{2}(x_k + a x_k^{-1})$$

Newton's method for matrix square root (Newton-SQRT)

$$\begin{aligned} X_0 &= A \\ X_{k+1} &= \frac{1}{2}(X_k + A X_k^{-1}) \end{aligned}$$

Newton's method for scalar-valued equation:

$$g(x) = x^2 - a = 0$$

Simplifies to

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} = \ldots = \frac{1}{2}(x_k + a x_k^{-1})$$

Newton's method for matrix square root (Newton-SQRT)

$$\begin{aligned} X_0 &= A \\ X_{k+1} &= \frac{1}{2}(X_k + A X_k^{-1}) \end{aligned}$$

* Julia demo *

Newton's method for scalar-valued equation:

$$g(x) = x^2 - a = 0$$

Simplifies to

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} = \ldots = \frac{1}{2}(x_k + a x_k^{-1})$$

Newton's method for matrix square root (Newton-SQRT)

$$\begin{aligned} X_0 &= A \\ X_{k+1} &= \frac{1}{2}(X_k + A X_k^{-1}) \end{aligned}$$

* Julia demo *

* Prove equivalence with Newton's method for $A = A^T$ *

Unfortunately: Newton's method for matrix square root is numerically unstable. Commutativity is important (not only our proof). * Julia demo *

Unfortunately: Newton's method for matrix square root is numerically unstable. Commutativity is important (not only our proof). * Julia demo *
Better in terms of stability:

Denman-Beavers algorithm

$$
\begin{aligned}
X_0 &= A \\
Y_0 &= I \\
X_{k+1} &:= \frac{1}{2}(X_k + Y_k^{-1}) \\
Y_{k+1} &:= \frac{1}{2}(Y_k + X_k^{-1})
\end{aligned}
$$

Unfortunately: Newton's method for matrix square root is numerically unstable. Commutativity is important (not only our proof). * Julia demo *
Better in terms of stability:

Denman-Beavers algorithm

$$\begin{aligned} X_0 &= A \\ Y_0 &= I \\ X_{k+1} &:= \frac{1}{2}(X_k + Y_k^{-1}) \\ Y_{k+1} &:= \frac{1}{2}(Y_k + X_k^{-1}) \end{aligned}$$

Properties of Denman-Beavers:

- Equivalent to Newton-SQRT in exact arithmetic, but very different in finite arithmetic

  * proof on black board *

Unfortunately: Newton's method for matrix square root is numerically unstable. Commutativity is important (not only our proof). * Julia demo *
Better in terms of stability:

Denman-Beavers algorithm

$$\begin{aligned} X_0 &= A \\ Y_0 &= I \\ X_{k+1} &:= \frac{1}{2}(X_k + Y_k^{-1}) \\ Y_{k+1} &:= \frac{1}{2}(Y_k + X_k^{-1}) \end{aligned}$$

Properties of Denman-Beavers:

- Equivalent to Newton-SQRT in exact arithmetic, but very different in finite arithmetic

  * proof on black board *

- Much less sensitive to round-off than Newton-SQRT

Unfortunately: Newton's method for matrix square root is numerically unstable. Commutativity is important (not only our proof). * Julia demo *

Better in terms of stability:

Denman-Beavers algorithm

$$
\begin{aligned}
X_0 &= A \\
Y_0 &= I \\
X_{k+1} &:= \frac{1}{2}(X_k + Y_k^{-1}) \\
Y_{k+1} &:= \frac{1}{2}(Y_k + X_k^{-1})
\end{aligned}
$$

Properties of Denman-Beavers:

- Equivalent to Newton-SQRT in exact arithmetic, but very different in finite arithmetic

  * proof on black board *

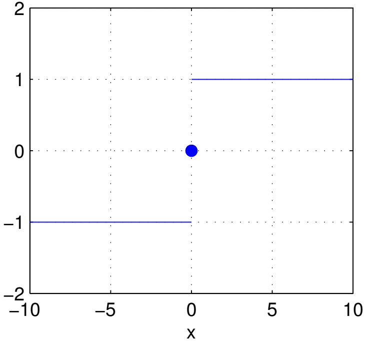
- Much less sensitive to round-off than Newton-SQRT
- One step requires two matrix inverses

Matrix sign function

PDF Lecture notes 4.3.3

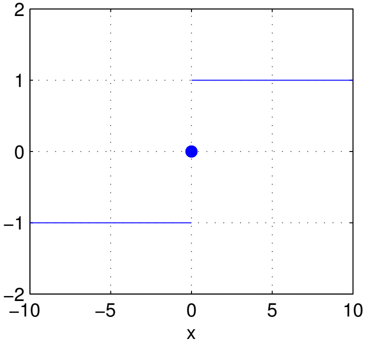
Scalar-valued sign function

$$\mathrm{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$



Scalar-valued sign function

$$\mathrm{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$



Now: Matrix version.

Applications

Quantum Chemistry (linear scaling DFT-code) and systems and control (Riccati equation)

Applications

Quantum Chemistry (linear scaling DFT-code) and systems and control (Riccati equation)

For all cases except $x = 0$:

$$\begin{aligned} |x| &= \sqrt{x^2} \\ \operatorname{sign}(x) &= \frac{|x|}{x} = \frac{\sqrt{x^2}}{x} \end{aligned}$$

Applications

Quantum Chemistry (linear scaling DFT-code) and systems and control (Riccati equation)

For all cases except $x = 0$:

$$
\begin{aligned}
|x| &= \sqrt{x^2} \\
\mathrm{sign}(x) &= \frac{|x|}{x} = \frac{\sqrt{x^2}}{x}
\end{aligned}
$$

Definition matrix sign

$$\mathrm{sign}(A) = \sqrt{A^2}A^{-1}$$

Naive method

Compute directly

$$\operatorname{sign}(A) = \sqrt{A^2}A^{-1}$$

Naive method

Compute directly

$$\operatorname{sign}(A) = \sqrt{A^2} A^{-1}$$

We can do better: Combine Newton-SQRT with A^2 and A^{-1}

Naive method

Compute directly

$$\mathrm{sign}(A) = \sqrt{A^2}A^{-1}$$

We can do better: Combine Newton-SQRT with A^2 and A^{-1}

Derivation based on defining $S_k = A^{-1}X_k$ where X_k Newton-SQRT for $\sqrt{A^2}$ ...

* On black board *

Naive method

Compute directly

$$\mathrm{sign}(A) = \sqrt{A^2}A^{-1}$$

We can do better: Combine Newton-SQRT with A^2 and A^{-1}

Derivation based on defining $S_k = A^{-1}X_k$ where X_k Newton-SQRT for $\sqrt{A^2}$ ...

* On black board *

Matrix sign iteration

$$\begin{aligned} S_0 &= A \\ S_{k+1} &= \frac{1}{2}(S_k + S_k^{-1}) \end{aligned}$$

Convergence

- Local quadratic convergence follows from Newton equivalence.

Convergence

- Local quadratic convergence follows from Newton equivalence.
- We even have global convergence ...

Convergence

- Local quadratic convergence follows from Newton equivalence.
- We even have global convergence ...

Theorem (Global quadratic convergence of sign iteration)

Suppose $A \in \mathbb{R}^{n\times n}$ has no eigenvalues on the imaginary axis. Let $S = \operatorname{sign}(A)$, and S_k be generated by Sign iteration. Let

$$G_k := (S_k - S)(S_k + S)^{-1}. \tag{1}$$

Then,

- $S_k = S(I + G_k)(I - G_k)^{-1}$ *for all* k,
- $G_k \to 0$ *as* $k \to \infty$,
- $S_k \to S$ *as* $k \to \infty$, *and*
- $$\|S_{k+1} - S\| \leq \frac{1}{2}\|S_k^{-1}\|\|S_k - S\|^2. \tag{2}$$