Probabilistic Graphical Models

Lecture 4: Essential Numerical Mathematics. Vector Calculus

Volkan Cevher, Matthias Seeger
Ecole Polytechnique Fédérale de Lausanne

10/10/2011

Outline

Why Numerical Mathematics?

1. Great new idea? Have to run it on an imperfect machine!
   - Continuous variables: Messages are vectors / matrices, manipulated large number of times
   - Without care, errors (roundoff, cancellation, . . .) accumulate
   - Greatest idea worth nothing if not numerically stable

Why Numerical Mathematics?

1. Great new idea? Have to run it on an imperfect machine!
   - Continuous variables: Messages are vectors / matrices, manipulated large number of times
   - Without care, errors (roundoff, cancellation, . . .) accumulate
   - Greatest idea worth nothing if not numerically stable
2. Thousands / millions of variables? Need iterative approximations
   - Backbone of about any continuous inference approximation: Iterative solvers from numerical mathematics
   - They are bottlenecks: Need to understand their properties

Positive Definite Matrices

Positive definite $\boldsymbol{A}$ (symmetric positive definite)

$$\boldsymbol{A}^T = \boldsymbol{A}, \quad \boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} > 0 \text{ for all } \boldsymbol{v} \neq \boldsymbol{0}$$

Equivalent to:

- All eigenvalues positive F2
- $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{X}$ full rank

Positive Definite Matrices

Positive definite $\boldsymbol{A}$ (symmetric positive definite)

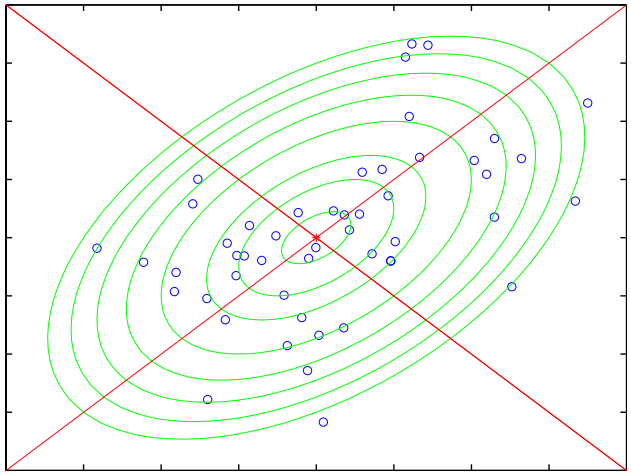
$$\boldsymbol{A}^T = \boldsymbol{A}, \quad \boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} > 0 \text{ for all } \boldsymbol{v} \neq \boldsymbol{0}$$

Equivalent to:

- All eigenvalues positive
- $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{X}$ full rank

$\boldsymbol{A}$ pos. def. $\Leftrightarrow$ Covariance matrix of Gaussian

F2(2)



Positive Definite Matrices

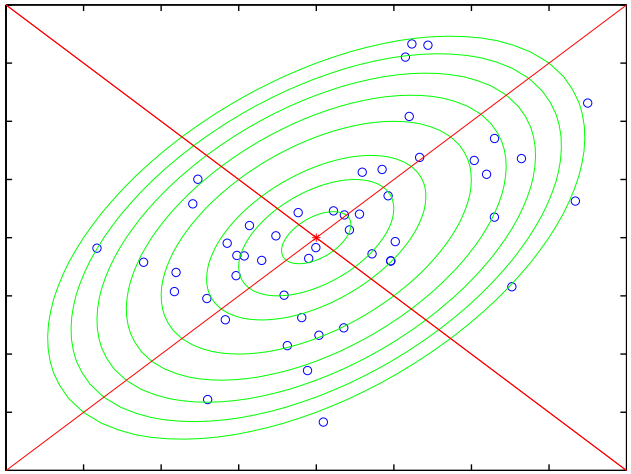
Positive definite $\boldsymbol{A}$ (symmetric positive definite)

$$\boldsymbol{A}^T = \boldsymbol{A}, \quad \boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} > 0 \text{ for all } \boldsymbol{v} \neq \boldsymbol{0}$$

Equivalent to:

- All eigenvalues positive
- $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{X}$ full rank

$\boldsymbol{A}$ pos. def. $\Leftrightarrow$ Covariance matrix of Gaussian



Numbers	Matrices
$\mathbb{C}$	any square
$\mathbb{R}$	symmetric (hermitian)
> 0	positive definite

$\Rightarrow$ Pos. def. simplifies many methods

Positive Definite Matrices

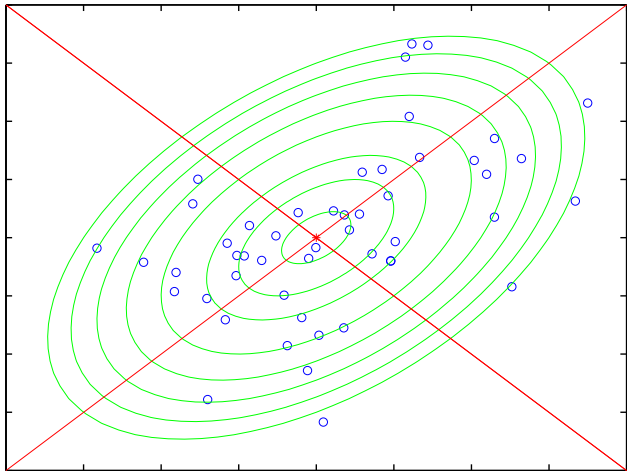
Positive definite $\boldsymbol{A}$ (symmetric positive definite)

$$\boldsymbol{A}^T = \boldsymbol{A}, \quad \boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} > 0 \text{ for all } \boldsymbol{v} \neq \boldsymbol{0}$$

Equivalent to:

- All eigenvalues positive
- $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{X}$ full rank

$\boldsymbol{A}$ pos. def. $\Leftrightarrow$ Covariance matrix of Gaussian



Positive semidefinite $\boldsymbol{A}$: "$\geq$" instead of "$>$"
$\Leftrightarrow \boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, any $\boldsymbol{X}$

Positive Definite Matrices

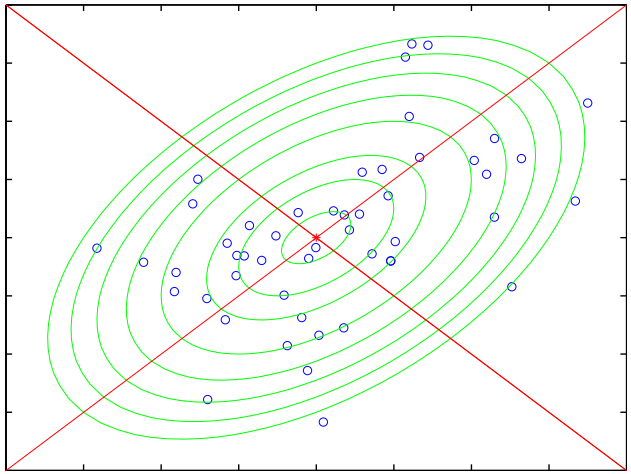
Positive definite $\boldsymbol{A}$ (symmetric positive definite)

$$\boldsymbol{A}^T = \boldsymbol{A}, \quad \boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} > 0 \text{ for all } \boldsymbol{v} \neq \boldsymbol{0}$$

Equivalent to:

- All eigenvalues positive
- $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{X}$ full rank

$\boldsymbol{A}$ pos. def. $\Leftrightarrow$ Covariance matrix of Gaussian



Positive semidefinite $\boldsymbol{A}$: "$\geq$" instead of "$>$"
$\Leftrightarrow \boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, any $\boldsymbol{X}$

$\boldsymbol{A}$ pos. semidef. $\Leftrightarrow$ Covariance matrix of degenerate Gaussian
[variance 0 along some directions]

[More about matrices? Horn, Johnson: Matrix Analysis (1985)]

Cholesky Decomposition

What is a matrix decomposition? The right way to use $\boldsymbol{A}^{-1}$!

Rule 1 for Matrix Computations

Do not invert a matrix. Decompose it
[Rule 2: Do not code it yourself. Use BLAS / LAPACK]

Cholesky Decomposition

What is a matrix decomposition? The right way to use $\boldsymbol{A}^{-1}$!

Rule 1 for Matrix Computations

Do not invert a matrix. Decompose it
[Rule 2: Do not code it yourself. Use BLAS / LAPACK]

Positive definite $\boldsymbol{A}$: Cholesky decomposition

$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$. Can I use lower triangular $\boldsymbol{L} = \boldsymbol{X}$?
$\Rightarrow$ Yes, exactly one: Cholesky factor of $\boldsymbol{A}$ F3

Cholesky Decomposition

What is a matrix decomposition? The right way to use $\boldsymbol{A}^{-1}$!

Rule 1 for Matrix Computations

Do not invert a matrix. Decompose it
[Rule 2: Do not code it yourself. Use BLAS / LAPACK]

Positive definite $\boldsymbol{A}$: Cholesky decomposition

$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$. Can I use lower triangular $\boldsymbol{L} = \boldsymbol{X}$?
$\Rightarrow$ Yes, exactly one: Cholesky factor of $\boldsymbol{A}$

Remarkable facts:

- Maybe the only algorithm in numerical mathematics that is so simple. If it fails, $\boldsymbol{A}$ is not (numerically) pos. def.
- In-place algorithm: $\boldsymbol{L}$ can overwrite $\boldsymbol{A}$. $l_{i,i} > 0$
- Complexity $O(n^3)$ [$\boldsymbol{A} \in \mathbb{R}^{n \times n}$]

Working with Cholesky Factors

Suppose: $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T \in \mathbb{R}^{n \times n}$, $\boldsymbol{L}$ lower triangular

- Solving linear system: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{b}$

$$[1]: \ \boldsymbol{L}\boldsymbol{v} = \boldsymbol{b}, \qquad [2]: \ \boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{v}$$

Two backsubstitutions, at $O(n^2)$ F4

[Beware: Some books distinguish between forward-, back-substitutions. I don't: It's just the same algorithm]

Working with Cholesky Factors

Suppose: $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T \in \mathbb{R}^{n \times n}$, $\boldsymbol{L}$ lower triangular

- Solving linear system: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{b}$

$$[1]: \ \boldsymbol{L}\boldsymbol{v} = \boldsymbol{b}, \qquad [2]: \ \boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{v}$$

Two backsubstitutions, at $O(n^2)$

[Beware: Some books distinguish between forward-, back-substitutions. I don't: It's just the same algorithm]

- Log determinant:

$$\log|\boldsymbol{A}| = 2\log|\boldsymbol{L}| = 2\sum_{i=1}^{n}\log l_{i,i}$$

- Other expressions:

$$\boldsymbol{b}^T\boldsymbol{A}^{-1}\boldsymbol{b} = \|\boldsymbol{v}\|^2, \quad \boldsymbol{L}\boldsymbol{v} = \boldsymbol{b}$$

Sequential Bayesian Updates

Example:

- Robot unsure about location, velocity: $P(\boldsymbol{u}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{u} \in \mathbb{R}^n$
- Obtains noisy linear measurements: $y = \boldsymbol{x}^T \boldsymbol{u} + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$
- Sequential update of belief state: $P(\boldsymbol{u}) \rightarrow P(\boldsymbol{u}|\boldsymbol{x}, y)$, ...

Sequential Bayesian Updates

Example:

- Robot unsure about location, velocity: $P(\boldsymbol{u}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{u} \in \mathbb{R}^n$
- Obtains noisy linear measurements: $y = \boldsymbol{x}^T \boldsymbol{u} + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$
- Sequential update of belief state: $P(\boldsymbol{u}) \rightarrow P(\boldsymbol{u}|\boldsymbol{x}, y), \ldots$

Recall last lecture:

$$\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{x}(\sigma^2 + \boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x})^{-1}\boldsymbol{x}^T\boldsymbol{\Sigma},$$
$$\boldsymbol{\mu}' = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{x}(\sigma^2 + \boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x})^{-1}(y - \boldsymbol{x}^T\boldsymbol{\mu})$$

State representation: Could simply maintain $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$

Sequential Bayesian Updates

Example:

- Robot unsure about location, velocity: $P(\boldsymbol{u}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{u} \in \mathbb{R}^n$
- Obtains noisy linear measurements: $y = \boldsymbol{x}^T \boldsymbol{u} + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$
- Sequential update of belief state: $P(\boldsymbol{u}) \rightarrow P(\boldsymbol{u}|\boldsymbol{x}, y), \ldots$

Recall last lecture:

$$\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{x}(\sigma^2 + \boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x})^{-1}\boldsymbol{x}^T\boldsymbol{\Sigma},$$
$$\boldsymbol{\mu}' = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{x}(\sigma^2 + \boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x})^{-1}(y - \boldsymbol{x}^T\boldsymbol{\mu})$$

State representation: Could simply maintain $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$

Better: Use Cholesky representation: $\boldsymbol{L}$, $\boldsymbol{a}$ s.t. $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$, $\boldsymbol{\mu} = \boldsymbol{L}\boldsymbol{a}$

- Slightly more efficient
- Better numerical properties [details in exercise]

[Cholesky up-/downdates: TR at people.mmci.uni-saarland.de/∼mseeger/papers/cholupdate.pdf
Code at people.mmci.uni-saarland.de/∼mseeger/software.html]

Singular Value Decomposition

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T \in \mathbb{R}^{m\times n}, \quad \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I},\ \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I},\ \boldsymbol{\Lambda} \text{ diagonal}, \lambda_i \geq 0$$

- Singular values $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times d}$ diagonal ($d \leq \min\{m, n\}$): λ_i^2 positive eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$
- $\boldsymbol{A}$ symmetric $\rightarrow \boldsymbol{U} = \boldsymbol{V}\boldsymbol{D}$, $d_i \in \{\pm 1\}$: Eigenvectors of $\boldsymbol{A}$
- $\boldsymbol{A}$ positive definite $\rightarrow$ SVD $\equiv$ eigendecomposition

Singular Value Decomposition

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T \in \mathbb{R}^{m\times n}, \quad \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I},\ \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I},\ \boldsymbol{\Lambda} \text{ diagonal}, \lambda_i \geq 0$$

- Singular values $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times d}$ diagonal ($d \leq \min\{m, n\}$): λ_i^2 positive eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$
- $\boldsymbol{A}$ symmetric $\rightarrow \boldsymbol{U} = \boldsymbol{V}\boldsymbol{D}$, $d_i \in \{\pm 1\}$: Eigenvectors of $\boldsymbol{A}$
- $\boldsymbol{A}$ positive definite $\rightarrow$ SVD $\equiv$ eigendecomposition
- Can be computed robustly for about any matrix, in $O(n^3)$
- For pos. def. matrices: SVD more expensive than Cholesky $\Rightarrow$ Use latter if sufficient for your goal

Singular Value Decomposition

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T \in \mathbb{R}^{m\times n}, \quad \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I},\ \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I},\ \boldsymbol{\Lambda} \text{ diagonal}, \lambda_i \geq 0$$

- Singular values $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times d}$ diagonal ($d \leq \min\{m, n\}$):
  λ_i^2 positive eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$
- $\boldsymbol{A}$ symmetric $\rightarrow \boldsymbol{U} = \boldsymbol{V}\boldsymbol{D}$, $d_i \in \{\pm 1\}$: Eigenvectors of $\boldsymbol{A}$
- $\boldsymbol{A}$ positive definite $\rightarrow$ SVD $\equiv$ eigendecomposition
- Can be computed robustly for about any matrix, in $O(n^3)$
- For pos. def. matrices: SVD more expensive than Cholesky
  $\Rightarrow$ Use latter if sufficient for your goal
- Some applications in machine learning:
  - Principal components analysis (PCA)
  - Optimal low-rank matrix approximation (covariance explained)
    $\Rightarrow$ Basis for linear approximations
  - Spectral clustering, manifold regularization, . . . :
    Leading eigenvectors carry lot of information about $\boldsymbol{A}$

Example: Design Optimization

You know / have learned: $\boldsymbol{u} \sim N(\mu, \Sigma)$ [but want to know more]

You can do: One more noisy linear measurement,
$y \sim N(\boldsymbol{x}^T \boldsymbol{u}, \sigma^2)$, $\|\boldsymbol{x}\| = 1$

Which $\boldsymbol{x}$ do you choose?

F7

Example: Design Optimization

You know / have learned: $\boldsymbol{u} \sim N(\mu, \Sigma)$ [but want to know more]
You can do: One more noisy linear measurement,
$y \sim N(\boldsymbol{x}^T \boldsymbol{u}, \sigma^2)$, $\|\boldsymbol{x}\| = 1$

Which $\boldsymbol{x}$ do you choose?

$$\boldsymbol{x}_* = \text{argmax}\{\boldsymbol{x}^T \Sigma \boldsymbol{x} \mid \|\boldsymbol{x}\| = 1\} : \quad \text{Leading eigenvector } \Sigma$$

Remember MRI sampling optimization [first lecture]?
$\Rightarrow$ That's (part of) how it works [stay on for rest]

Why Iterative Solvers?

- Moderate-sized high-resolution image: $n = 65536$ pixels.

  Storage: 32G (single matrix)
  Time for Cholesky decomposition: $\approx$ 3h (if enough memory)

  $\Rightarrow$ Bayesian methods over images, stacks of images, videos, . . . ?

Why Iterative Solvers?

- Moderate-sized high-resolution image: $n = 65536$ pixels.

  Storage: 32G (single matrix)
  Time for Cholesky decomposition: $\approx$ 3h (if enough memory)

  $\Rightarrow$ Bayesian methods over images, stacks of images, videos, . . . ?

- And how about structure?

$$\mathrm{E}[\boldsymbol{u}|\boldsymbol{y}] = (\boldsymbol{X}^T \boldsymbol{\Psi}^{-1} \boldsymbol{X} + \boldsymbol{\Pi})^{-1} (\boldsymbol{X}^T \boldsymbol{\Psi}^{-1} \boldsymbol{y} + \boldsymbol{b})$$

  - $\boldsymbol{X}$ sparse (most entries $= 0$) [e.g., consumer-product ratings]
  - $\boldsymbol{X}$ structured (banded, Toeplitz, . . .) [e.g., Markov grid structure]
  - $\boldsymbol{X}$ fast operator (FFT) [e.g., MRI measurements]

  Matrix decompositions do not make use of that

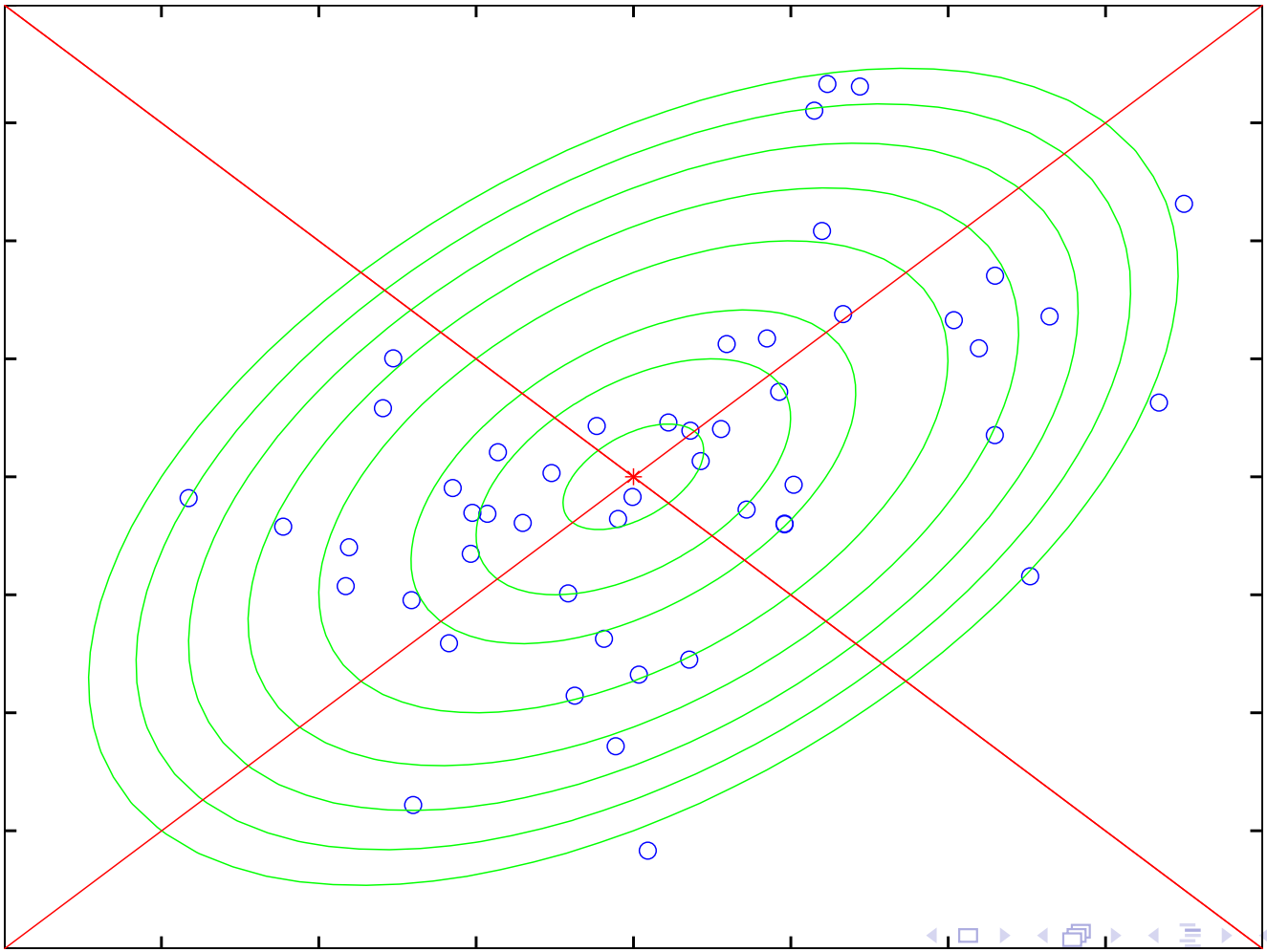
[Some do: sparse Cholesky factorization]

Minimizing Quadratic Functions

For positive definite $\boldsymbol{A}$:

F9

$$\boldsymbol{x}_* = \text{argmin}\{q(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}\} \quad \Leftrightarrow \quad \boldsymbol{A}\boldsymbol{x}_* = \boldsymbol{b}$$



Minimizing Quadratic Functions

$$q(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}, \quad \boldsymbol{g}(\boldsymbol{x}) = \nabla q(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$$

Require: Operator $\boldsymbol{A}$. Initial $\boldsymbol{x}_0$

for $k = 1, 2, \ldots$ **do**

Pick search direction $\boldsymbol{d}_k$, based on $\boldsymbol{g}_{k-1} = \boldsymbol{g}(\boldsymbol{x}_{k-1})$, $\boldsymbol{d}_l$, $l < k$

Line minimization:

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{d}_k, \quad \alpha_k = \operatorname{argmin}_\alpha q(\boldsymbol{x}_{k-1} + \alpha \boldsymbol{d}_k)$$

end for

Conjugate Directions

- Why, of course down as steep as possible

$$q(\boldsymbol{x}_{k-1} + d\boldsymbol{x}) = q(\boldsymbol{x}_{k-1}) + \underbrace{\boldsymbol{g}_{k-1}^T(d\boldsymbol{x})}_{\text{Smallest: } d\boldsymbol{x} \propto -\boldsymbol{g}_{k-1}} + O(\|d\boldsymbol{x}\|^2)$$

Steepest descent: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1}$ F11

Conjugate Directions

- Why, of course down as steep as possible

$$q(\boldsymbol{x}_{k-1} + d\boldsymbol{x}) = q(\boldsymbol{x}_{k-1}) + \underbrace{\boldsymbol{g}_{k-1}^T(d\boldsymbol{x})}_{\text{Smallest: } d\boldsymbol{x} \propto -\boldsymbol{g}_{k-1}} + O(\|d\boldsymbol{x}\|^2)$$

  Steepest descent: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1}$

- Wrong: For steepest descent: $\boldsymbol{d}_{k+1}^T \boldsymbol{d}_k = 0$ F11(2)
  $\Rightarrow$ Improvements from previous iterations rapidly tempered with

Conjugate Directions

- Why, of course down as steep as possible

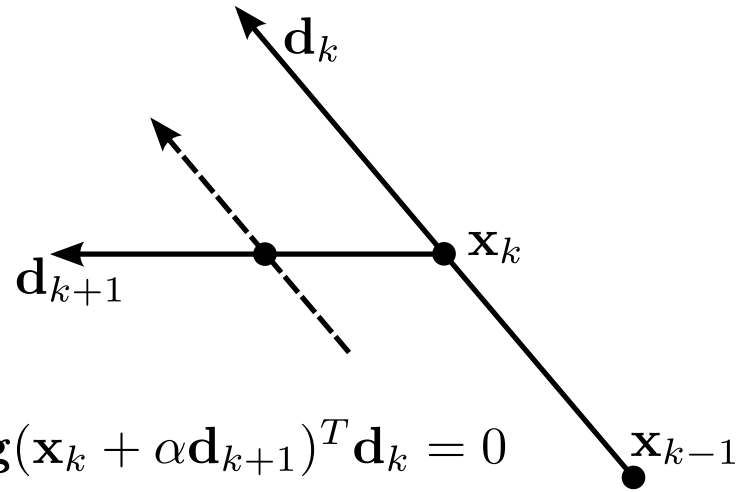
$$q(\boldsymbol{x}_{k-1} + d\boldsymbol{x}) = q(\boldsymbol{x}_{k-1}) + \underbrace{\boldsymbol{g}_{k-1}^T(d\boldsymbol{x})}_{\text{Smallest: } d\boldsymbol{x} \propto -\boldsymbol{g}_{k-1}} + O(\|d\boldsymbol{x}\|^2)$$

  Steepest descent: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1}$

- Wrong: For steepest descent: $\boldsymbol{d}_{k+1}^T \boldsymbol{d}_k = 0$
  $\Rightarrow$ Improvements from previous iterations rapidly tempered with

New gradients $\perp$ old directions?
$\Rightarrow$ Retains previous efforts:
$\boldsymbol{g}_k^T \boldsymbol{d}_k = 0 \rightarrow \boldsymbol{g}_{k+1}^T \boldsymbol{d}_k = 0 \ldots$

F11(3)



Conjugate Directions

- Why, of course down as steep as possible

$$q(\boldsymbol{x}_{k-1} + d\boldsymbol{x}) = q(\boldsymbol{x}_{k-1}) + \underbrace{\boldsymbol{g}_{k-1}^T(d\boldsymbol{x})}_{\text{Smallest: } d\boldsymbol{x} \propto -\boldsymbol{g}_{k-1}} + O(\|d\boldsymbol{x}\|^2)$$

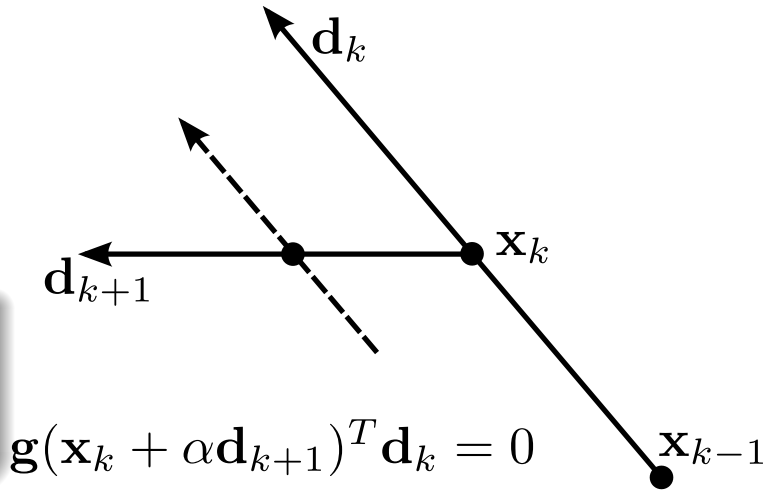
  Steepest descent: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1}$

- Wrong: For steepest descent: $\boldsymbol{d}_{k+1}^T \boldsymbol{d}_k = 0$
  $\Rightarrow$ Improvements from previous iterations rapidly tempered with

New gradients $\perp$ old directions?
$\Rightarrow$ Retains previous efforts:
$\boldsymbol{g}_k^T \boldsymbol{d}_k = 0 \rightarrow \boldsymbol{g}_{k+1}^T \boldsymbol{d}_k = 0 \ldots$

Conjugate Directions

$\boldsymbol{d}_k^T \boldsymbol{A} \boldsymbol{d}_j$ for all $j < k$



Towards Conjugate Gradients

Details: Handout people.mmci.uni-saarland.de/∼mseeger/lectures/bml09/handout_lect4.pdf

1. Directions conjugate: Gradient $\boldsymbol{g}_k \perp$ all previous directions: $\boldsymbol{g}_k^T \boldsymbol{d}_j = 0$ for all $j \leq k$

Towards Conjugate Gradients

Details: Handout people.mmci.uni-saarland.de/∼mseeger/lectures/bml09/handout_lect4.pdf

1. Directions conjugate: Gradient $\boldsymbol{g}_k \perp$ all previous directions: $\boldsymbol{g}_k^T \boldsymbol{d}_j = 0$ for all $j \leq k$
2. After n steps we are done: $\boldsymbol{g}_n = \boldsymbol{0}$
3. Construct conjugate directions by recurrence: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1} + \beta_{k-1} \boldsymbol{d}_{k-1}$

Towards Conjugate Gradients

Details: Handout people.mmci.uni-saarland.de/∼mseeger/lectures/bml09/handout_lect4.pdf

1. Directions conjugate: Gradient $\boldsymbol{g}_k \perp$ all previous directions: $\boldsymbol{g}_k^T \boldsymbol{d}_j = 0$ for all $j \leq k$
2. After n steps we are done: $\boldsymbol{g}_n = \boldsymbol{0}$
3. Construct conjugate directions by recurrence: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1} + \beta_{k-1} \boldsymbol{d}_{k-1}$
4. All gradients are orthogonal: $\boldsymbol{g}_k^T \boldsymbol{g}_j = 0$, $j < k$
   [Bit of misnomer: Directions are conjugate]

Towards Conjugate Gradients

Details: Handout people.mmci.uni-saarland.de/∼mseeger/lectures/bml09/handout_lect4.pdf

1. Directions conjugate: Gradient $\boldsymbol{g}_k \perp$ all previous directions: $\boldsymbol{g}_k^T \boldsymbol{d}_j = 0$ for all $j \leq k$
2. After n steps we are done: $\boldsymbol{g}_n = \boldsymbol{0}$
3. Construct conjugate directions by recurrence: $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1} + \beta_{k-1} \boldsymbol{d}_{k-1}$
4. All gradients are orthogonal: $\boldsymbol{g}_k^T \boldsymbol{g}_j = 0$, $j < k$
   [Bit of misnomer: Directions are conjugate]
5. What is α_k? From line minimization:

$$\alpha_k = \frac{\|\boldsymbol{g}_{k-1}\|^2}{\boldsymbol{d}_k^T \boldsymbol{A} \boldsymbol{d}_k}$$

6. What is β_k? The great synthesis!

$$\beta_k = \frac{\|\boldsymbol{g}_k\|^2}{\|\boldsymbol{g}_{k-1}\|^2}$$

Conjugate Gradients (for reference)

Require: Operator $\boldsymbol{A}$. Initial $\boldsymbol{x}_0$. $\boldsymbol{g}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$

for $k = 1, 2, \ldots$ (no more than n) **do**

$\rho_{k-1} = \|\boldsymbol{g}_{k-1}\|^2$

if $k = 1$ **then**

$\boldsymbol{d}_1 = -\boldsymbol{g}_0$

else

$\beta_{k-1} = \rho_{k-1}/\rho_{k-2}$; $\boldsymbol{d}_k = -\boldsymbol{g}_{k-1} + \beta_{k-1}\boldsymbol{d}_{k-1}$

end if

$\boldsymbol{q}_k = \boldsymbol{A}\boldsymbol{d}_k$; $\alpha_k = \rho_{k-1}/(\boldsymbol{d}_k^T\boldsymbol{q}_k)$

$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k\boldsymbol{d}_k$; $\boldsymbol{g}_k = \boldsymbol{g}_{k-1} + \alpha_k\boldsymbol{q}_k$

Check for convergence (say $\|\boldsymbol{g}_k\|/\|\boldsymbol{b}\| < \varepsilon$)

end for

Krylov Subspaces

Let $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$. Then, F14

$$\boldsymbol{g}_k^T \boldsymbol{d}_j = 0,\ j \leq k \quad \Rightarrow \quad \boldsymbol{x}_k = \text{argmin}_{\boldsymbol{x} \in_k} q(\boldsymbol{x})$$

Krylov Subspaces

Let $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$. Then,

$$\boldsymbol{g}_k^T \boldsymbol{d}_j = 0,\ j \leq k \quad \Rightarrow \quad \boldsymbol{x}_k = \text{argmin}_{\boldsymbol{x} \in \mathcal{K}_k}\, q(\boldsymbol{x})$$

But $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{A}^j \boldsymbol{g}_0 \,|\, j < k\}$
$\Rightarrow$ Optimal with k ($\boldsymbol{A}\cdot$) multiplications!

F14(2)

- $\mathcal{K}_k \subset \mathcal{K}_{k+1} \subset \ldots$, $\boldsymbol{x}_* \in \mathcal{K}_n$ (Cayley/Hamilton)

Krylov Subspaces

Let $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$. Then,

$$\boldsymbol{g}_k^T \boldsymbol{d}_j = 0,\ j \leq k \quad \Rightarrow \quad \boldsymbol{x}_k = \text{argmin}_{\boldsymbol{x} \in \mathcal{K}_k}\, q(\boldsymbol{x})$$

But $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{A}^j \boldsymbol{g}_0 \,|\, j < k\}$
$\Rightarrow$ Optimal with k ($\boldsymbol{A}\cdot$) multiplications!

- $\mathcal{K}_k \subset \mathcal{K}_{k+1} \subset \ldots$, $\boldsymbol{x}_* \in \mathcal{K}_n$ (Cayley/Hamilton)
- What about $k \ll n$ for huge n? Depends on eigenspectrum of $\boldsymbol{A}$.
  $\boldsymbol{x}_k \approx \boldsymbol{x}_*$ in surprisingly many cases in practice
  $\Rightarrow$ Krylov subspace view key to convergence analysis [exercise]

Krylov Subspaces

Let $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$. Then,

$$\boldsymbol{g}_k^T \boldsymbol{d}_j = 0,\ j \leq k \quad \Rightarrow \quad \boldsymbol{x}_k = \text{argmin}_{\boldsymbol{x} \in \mathcal{K}_k} q(\boldsymbol{x})$$

But $\mathcal{K}_k = \boldsymbol{x}_0 + \text{span}\{\boldsymbol{A}^j \boldsymbol{g}_0 \,|\, j < k\}$
$\Rightarrow$ Optimal with k $(\boldsymbol{A}\cdot)$ multiplications!

- $\mathcal{K}_k \subset \mathcal{K}_{k+1} \subset \ldots$, $\boldsymbol{x}_* \in \mathcal{K}_n$ (Cayley/Hamilton)
- What about $k \ll n$ for huge n? Depends on eigenspectrum of $\boldsymbol{A}$.
  $\boldsymbol{x}_k \approx \boldsymbol{x}_*$ in surprisingly many cases in practice
  $\Rightarrow$ Krylov subspace view key to convergence analysis [exercise]
- Preconditioning: $\boldsymbol{M} = \boldsymbol{C}\boldsymbol{C}^T \approx \boldsymbol{A}$, but easy to solve systems with
  - Work on $(\boldsymbol{C}^{-T}\boldsymbol{A}\boldsymbol{C}^{-1})\boldsymbol{C}\boldsymbol{x} = \boldsymbol{C}^{-T}\boldsymbol{b}$
    $\Rightarrow$ Better spectral properties $\rightarrow$ Faster convergence
  - CG as before, with one $(\boldsymbol{M}^{-1}\cdot)$ per iteration
  - Preconditioning: Art of iterative linear solvers

Why Vector Calculus?

- Remember differentiation? A bunch of rules, no-brainer.
  Do that in $\mathbb{R}^n$: $(\partial f_i)/(\partial x_j) = \sum_k (\partial g_i)/(\partial y_k) \cdot (\partial y_k)/(\partial x_j)$, ...?

Why Vector Calculus?

- Remember differentiation? A bunch of rules, no-brainer.
  Do that in $\mathbb{R}^n$: $(\partial f_i)/(\partial x_j) = \sum_k (\partial g_i)/(\partial y_k) \cdot (\partial y_k)/(\partial x_j)$, ...?
- No! Use bunch of rules on vectors, matrices.
  No $\sum_{j,k}$, no ∂_i/∂_j: Waste of paper / your time
- Vector calculus in machine learning:
  - $\nabla f(\boldsymbol{x}) = \mathbf{0}$, solve for $\boldsymbol{x}$ (if you're lucky)
  - Search directions always fed by gradient (steepest descent)
  - Newton (second order) optimization: Hessian as well

Like with Gaussian: Experience with the rules pays off!

Differential Notation

- Trace of square matrix

F16

$$\operatorname{tr} \boldsymbol{A} = \sum_i \alpha_{i,i} = \sum_i \lambda_i, \quad \operatorname{spec}(\boldsymbol{A}) = \{\lambda_1, \ldots, \lambda_n\}$$

$$\operatorname{tr} \boldsymbol{AB} = \operatorname{tr} \boldsymbol{BA}, \quad \boldsymbol{x}^T \boldsymbol{Ax} \stackrel{!}{=} \operatorname{tr} \boldsymbol{x}^T \boldsymbol{Ax} = \operatorname{tr} \boldsymbol{Axx}^T$$

Differential Notation

- Trace of square matrix

$$\operatorname{tr} \boldsymbol{A} = \sum_i \alpha_{i,i} = \sum_i \lambda_i, \quad \operatorname{spec}(\boldsymbol{A}) = \{\lambda_1, \ldots, \lambda_n\}$$

$$\operatorname{tr} \boldsymbol{AB} = \operatorname{tr} \boldsymbol{BA}, \quad \boldsymbol{x}^T \boldsymbol{Ax} \stackrel{!}{=} \operatorname{tr} \boldsymbol{x}^T \boldsymbol{Ax} = \operatorname{tr} \boldsymbol{Axx}^T$$

- Differential $d\boldsymbol{x}$: Tiny vector, situated at $\boldsymbol{x}$. F16(2)
  $df(\boldsymbol{x}) = f(\boldsymbol{x} + (d\boldsymbol{x})) - f(\boldsymbol{x}) = (\nabla f(\boldsymbol{x}))^T (d\boldsymbol{x}) + O(\|d\boldsymbol{x}\|^2), \ldots$
  $\Rightarrow$ For vector calculus: $d\boldsymbol{x}$ special vector, obeying one more rule:
  Term with $d\boldsymbol{x} \geq 2$ times $\rightarrow$ Term = 0 (≥ 3 for Hessian)

[All I do here: Minka's note, http://research.microsoft.com/en-us/um/people/minka/papers/matrix/]

Simple Rules

- Constant. Linear

$$d\boldsymbol{A} = \boldsymbol{0} \quad [\boldsymbol{A} \text{ constant}]$$
$$d(\alpha \boldsymbol{X} + \beta \boldsymbol{Y}) = \alpha(d\boldsymbol{X}) + \beta(d\boldsymbol{Y}), \quad d \operatorname{tr} \boldsymbol{X} = \operatorname{tr}(d\boldsymbol{X})$$

- Product rule

$$d(\boldsymbol{X}\boldsymbol{Y}) = (d\boldsymbol{X})\boldsymbol{Y} + \boldsymbol{X}(d\boldsymbol{Y}) \quad [\text{also for } \otimes, \circ]$$

- Permutation/extraction: $(\cdot)^*$ reorders/extracts entries

$$d(\boldsymbol{X}^*) = (d\boldsymbol{X})^*$$

Example: $(\cdot)^T$, $\operatorname{diag}^{-1}(\cdot)$, vectorization (`reshape` in Matlab)

Prove any of them: lhs = rhs + $O(\|d\boldsymbol{x}\|^2)$

More Interesting Rules

- Matrix inverse

F18

$$d(\boldsymbol{X}^{-1}) = -\boldsymbol{X}^{-1}(d\boldsymbol{X})\boldsymbol{X}^{-1}$$

More Interesting Rules

- Matrix inverse

$$d(\boldsymbol{X}^{-1}) = -\boldsymbol{X}^{-1}(d\boldsymbol{X})\boldsymbol{X}^{-1}$$

- Log determinant F18(2)

$$d \log |\boldsymbol{X}| = \operatorname{tr} \boldsymbol{X}^{-1}(d\boldsymbol{X})$$

And $d|\boldsymbol{X}| = d\, e^{\log |\boldsymbol{X}|} = |\boldsymbol{X}|(d \log |\boldsymbol{X}|)$

The "Algorithm"

Given: Really messy $f(\boldsymbol{x})$, $\boldsymbol{f}(\boldsymbol{x})$, $f(\boldsymbol{X})$, $\boldsymbol{F}(x)$.

1. Inward: Write $df(\boldsymbol{x})$ (or other forms). Use rules to push $d(\cdot)$ inside, until $d\boldsymbol{x}$ only

The "Algorithm"

Given: Really messy $f(\boldsymbol{x})$, $\boldsymbol{f}(\boldsymbol{x})$, $f(\boldsymbol{X})$, $\boldsymbol{F}(x)$.

1. Inward: Write $df(\boldsymbol{x})$ (or other forms). Use rules to push $d(\cdot)$ inside, until $d\boldsymbol{x}$ only
2. Outward: Use linear algebra rules ($d\boldsymbol{x}$ vector, $d\boldsymbol{X}$ matrix!) to pull $d\boldsymbol{x}$ out, until:

$df = a(dx)$	$d\boldsymbol{f} = \boldsymbol{a}(dx)$	$d\boldsymbol{F} = \boldsymbol{A}(dx)$
$df = \boldsymbol{a}^T(d\boldsymbol{x})$	$d\boldsymbol{f} = \boldsymbol{A}(d\boldsymbol{x})$	
$df = \operatorname{tr} \boldsymbol{A}^T(d\boldsymbol{X})$		

Just read off derivative: a, $\boldsymbol{a}$, $\boldsymbol{A}$.
Empty cells? Don't like tensors beyond matrices. Use $\operatorname{vec}(\boldsymbol{X})$

The "Algorithm"

Given: Really messy $f(\boldsymbol{x})$, $\boldsymbol{f}(\boldsymbol{x})$, $f(\boldsymbol{X})$, $\boldsymbol{F}(x)$.

1. Inward: Write $df(\boldsymbol{x})$ (or other forms). Use rules to push $d(\cdot)$ inside, until $d\boldsymbol{x}$ only
2. Outward: Use linear algebra rules ($d\boldsymbol{x}$ vector, $d\boldsymbol{X}$ matrix!) to pull $d\boldsymbol{x}$ out, until:

$df = a(dx)$	$d\boldsymbol{f} = \boldsymbol{a}(dx)$	$d\boldsymbol{F} = \boldsymbol{A}(dx)$
$df = \boldsymbol{a}^T(d\boldsymbol{x})$	$d\boldsymbol{f} = \boldsymbol{A}(d\boldsymbol{x})$	
$df = \operatorname{tr} \boldsymbol{A}^T(d\boldsymbol{X})$		

Just read off derivative: a, $\boldsymbol{a}$, $\boldsymbol{A}$.
Empty cells? Don't like tensors beyond matrices. Use $\operatorname{vec}(\boldsymbol{X})$

Constraints on $\boldsymbol{X}$? Inherited by $d\boldsymbol{X}$, $\boldsymbol{A}$ F19

$$\begin{aligned} \boldsymbol{X} \text{ symmetric} &\Rightarrow df = \operatorname{tr}(\operatorname{sym} \boldsymbol{A})(d\boldsymbol{X}),\ \operatorname{sym} \boldsymbol{A} = (\boldsymbol{A} + \boldsymbol{A}^T)/2 \\ \boldsymbol{X} \text{ diagonal} &\Rightarrow df = (\operatorname{diag} \operatorname{diag}^{-1}(\boldsymbol{A}))(d\boldsymbol{X}) \end{aligned}$$

Example

Model: $\boldsymbol{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ \boldsymbol{x} \in \mathbb{R}^d$
Data: Independent draws $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, $n > d$

Suppose $\boldsymbol{\mu}$ known. Maximum likelihood estimator for $\boldsymbol{\Sigma}$? F20

Summary

$\boldsymbol{A} = (a_{ij})_{ij}$. Then:

$\boldsymbol{A}^T = (a_{ji})_{ij}$	Transpose
$\lvert\boldsymbol{A}\rvert$	Determinant $\quad[\lvert\boldsymbol{I} + \boldsymbol{AB}\rvert = \lvert\boldsymbol{I} + \boldsymbol{BA}\rvert]$
$\operatorname{tr}\boldsymbol{A} = \sum_i a_{ii}$	Trace $\quad[\operatorname{tr}\boldsymbol{AB} = \operatorname{tr}\boldsymbol{BA}]$
$\operatorname{diag}^{-1}(\boldsymbol{A}) = (a_{ii})_i$	Diagonal of matrix
$(\operatorname{diag}\boldsymbol{v}) = (v_i \mathrm{I}_{\{i=j\}})_{ij}$	Diagonal matrix
$\operatorname{sym}\boldsymbol{A} = (\boldsymbol{A} + \boldsymbol{A}^T)/2$	Symmetrization
$\boldsymbol{A}_{I,J} = (a_{ij})_{i\in I, j\in J}$	Subselection ("i" for $\{i\}$, "$\cdot$" for full range)
$\mathrm{I}_{\{I,\cdot\}}\boldsymbol{v} = \boldsymbol{v}_I$	Subselection matrix
$\mathrm{I}_{\{\cdot,I\}}\boldsymbol{v} = (\sum_k \mathrm{I}_{\{j=i_k\}} v_k)_j$	Distribution matrix ($I = \{i_k\}$)

Wrap-Up

- Lots of stuff: Seems hard, tedious at first
- As computer scientists, we are engineers (get things done): These rules / basic algorithms are our toolbox
- Better know your basic tools very well (with practice, you will)