Numerik Zusammenfassung HS2018

Linear Algebra Basics

$A^{m \times n}$:

$$\begin{array}{c|c} & n, j \rightarrow \\ \hline m, i \downarrow & \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix} \end{array}$$

- n Spalten (|||)
- m Reihen (☰)

Symmetric: $A = A^T$

Rank: # Not-Null Rows after Gauss

Regular	Singular
- Square ($m=n$)	- Any size ($m \neq n$)
- Full rank (Rank = n) $\Rightarrow$ injective	- Rank < n
- $\det(A) \neq 0$	- $\det(A) = 0$
- Invertible	- Not invertible
- Eigenvalues $\neq 0$	- Eigenvalue $= 0$ (min. one)

Unitary / orthogonal
- Regular
- Columns are orthogonal ($a_i \cdot a_j = 0 \quad \forall i \neq j$) / $\langle a, b \rangle = 0$
- Inverse: $A^{-1} = A^H$

Symmetric Positive definite
- A is regular & symmetric
- A is positive definite ($z^T A z > 0$, $z \neq 0$), all EW $\lambda_i > 0$
- A has Cholesky-Decompos.: $A = LL^T$, L = lower triangular
- $\Rightarrow$ X regular $\Rightarrow$ $X^T X$ is spd!

Range of A: $R(A) := \{ y \mid Ax = y \quad \forall x \}$

Nullspace of A: $N(A) := \{ x \mid Ax = 0 \}$

- $R(AB) = R(A)$ / $N(AB) = N(B)$
- $N(A) = R(A^T)^\perp$ / $N(A)^\perp = R(A^T)$
- $N(B^T) = R(B)^\perp$ / $N(B^T)^\perp = R(B)$
- $R(A^T A) = R(A^T)$ / $N(A^T A) = N(A)$

Symmetric 2: For $A \in \mathbb{R}^{n,m} \Rightarrow A^T A$ & AA^T are symmetric and their Eigenvectors are orthogonal to each other

Gauss-Algorithm:
- „Allowed Operations": Adding of Rows (or Adding a multiple of a row), swapping of Rows

Inverse of Matrix with Gauss: (A is regular)
1) Write $(A \mid I_n)$
2) Gauss until $(I_n \mid A^{-1})$

LU/LR Decomposition with Gauss (A is regular)
1) Write $(I_n \mid A)$
2) Gauss A and write the multiplication factor with inversed sign in I_n at the place where a zero is „created" in A
3) Get $(L \mid R)$, calculate $Lc = b$; $Rx = c$; check $Ax = b$.

Gram-Schmidt: Turn $\{a_1, \dots, a_n\}$ in orthonormal $\{b_1, \dots, b_n\}$
1) $b_1 := a_1 / \|a_1\|$
2) $\tilde{b}_k := a_k - \sum_{j=1}^{k-1} \langle b_j, a_k \rangle \cdot b_j$ — for $k = 2, \dots, n$
3) $b_k := \tilde{b}_k / \|\tilde{b}_k\|$

Euclidian Norm $\|a\|_2 := \sqrt{a_1^2 + a_2^2 + \dots + a_n^2}$

Skalarproduct $\langle x, y \rangle := x^T y = \sum_{i=1}^{n} x_i \cdot y_i$

QR Decomposition: Turn A into QR ($A = QR$), where Q is orthogonal and R an upper triang. Matrix

Reduced QR: 1) Use GS to turn $A = \{a_1 | a_2 | \dots\}$ into $Q = \{q_1 | q_2 | \dots\}$

2) Get R as follows:
- $r_{11} := \|a_1\|$
- $r_{jk} := \langle q_j, a_k \rangle \quad j = 1, \dots, k-1$
- $r_{kk} := \|\tilde{q}_k\|$

Full QR Decomposition

1) Extend Q by $m-n$ orthonormal Vectors such that Q is now a square $m \times m$ Matrix $\tilde{Q}$. The vectors can be calculated using the Nullspace and GS. $\tilde{Q} = (Q \mid Q_\perp)$
2) Extend R with $(m-n)$ zero-rows: $\tilde{R} = \binom{R}{0}$ $\Rightarrow A = \tilde{Q}\tilde{R}$

Determinant (for a square matrix A)

- 1×1: $\det(a) = a$ - 2×2: $\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$
- 3×3: $a_{11}a_{22}a_{33} + a_{21}a_{32}a_{13} + a_{31}a_{12}a_{23} - a_{31}a_{22}a_{13} - a_{21}a_{12}a_{33} - a_{11}a_{32}a_{23}$

$\Rightarrow$ Swapping two rows flips the sign of the determinant!

- $\det(A) = \det(A^T)$ / If A regular: $\det(A^{-1}) = \frac{1}{\det(A)}$
- For triangular matrices is the determinant the product of $\text{diag}(A)$
- $\det(AB) = \det(A) \cdot \det(B)$ / $\det(A+B) \neq \det(A) + \det(B)$

Laplacian Formula for Eigenvalues

- By i-th row: $\det(A) = \sum_{j=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det(A_{ij})$
- By j-th column: $\det(A) = \sum_{i=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det(A_{ij})$

where A_{ij} is the $(n-1) \times (n-1)$ submatrix taken from A by removing the i-th row and the j-th column

Eigenwerte / Eigenvalues EW

1) Calculate char. Polynom $\chi_A := \det(A - \lambda I)$ ($-\lambda$ on diag.)
2) Calculate zeroes of $\chi_A \Rightarrow$ Zeroes $=$ EW
3) # of occurences of an EW = algebraic multiplicity

Eigenvectors EV

1) For each EW λ_i: Find all v for which $(A - \lambda_i I)v = 0$ (apply Gauss on $A - \lambda_i I$ and solve for v)
2) Geometric Mult. := # of EV per EW

Eigenvalue Decomposition

- For $A \in \mathbb{R}^{n \times n}$, let $\lambda_1, \dots, \lambda_k$ be the EW, and $v_1, \dots, v_k$ the EV.
- If one (or both) is true, A is diagonalizable:
  1) $k = n$ and all EW are different
  2) For each EW: algebraic M. = geometric M.

$\Rightarrow A = V \Lambda V^{-1}$ with $V = (v_1 \mid v_2 \mid \dots)$ and $\Lambda = \text{diag}(\lambda_i)_{i=1}^k$

Spectralnorm $\|A\|_2$ for $A \in \mathbb{R}^{n \times n}$, symmetric $= \max |\lambda_i|$

Condition number $:= \frac{\sigma_{max}}{\sigma_{min}} = \frac{\max \sqrt{\lambda}}{\min \sqrt{\lambda}}$

Definiteness $\forall x \in \mathbb{R}^n$

- Pos. definite, if $x^T A x > 0$ $\rightarrow \forall$ EW > 0
- Pos. semidef., if $x^T A x \geq 0$ $\rightarrow \forall$ EW ≥ 0
- Neg. definite, if $x^T A x < 0$ $\rightarrow \forall$ EW < 0
- Neg. semidef., if $x^T A x \leq 0$ $\rightarrow \forall$ EW ≤ 0

Singular Value Decomposition

- $A = U \Sigma V^H$, with
  - $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$ and $\sigma_i = \sqrt{\lambda_i}$
  - $V =$ Eigenvectormatrix from A
  - $U = (U_r \mid U_\perp)$ with $U_r = A V_r \Sigma^{-1}$, where V_r = first r cols of V
- Also there is:
  - V = right singular vectors = EV of $A^H A$
  - U = left singular vectors = EV of $A A^H$
  - U & V are unitary $\Rightarrow$ EV of $A^H A$ / $A A^H$ are orthonormal and $U^H = U^{-1}$ / $V^H = V^{-1}$

Injective: Each element of target set is max. once in function

Surjective: Each element of target set is hit at least once

Bijective: Both above

Cholesky Decomposition: If A is hermitian (symmetric) positive-definite, it has a decomposition $A = LL^T$ (L = lower triang.)

Injective Matrix: If A has full rank, it is injective. This means that each value $x = Ab$ is only „hit" once. Because $0 = A0$ (trivial solution), no other b exists such that $Ab = 0$ $\Rightarrow$ A is positive definite

Calculus / NumCSE Basics

Jacobi Matrix is the matrix of all partial differentials of a function $f := (f_1, f_2, f_3, \dots, f_m)$ with $x := (x_1, x_2, \dots, x_n)$ as the coordinates (variables):

$$J_f(a) := \left(\frac{\partial f_i}{\partial x_j}(a)\right) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \frac{\partial f_1}{\partial x_2}(a) & \cdots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & & & \\ \frac{\partial f_m}{\partial x_1}(a) & \frac{\partial f_m}{\partial x_2}(a) & \cdots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix}$$

Hornerschema for evaluation of Polynoms

- A polynom $p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$ can be evaluated using $p(x) = ((a_n x + a_{n-1})x + a_{n-2})x + \dots)x + a_0$

Newton Interpolation: Easy Matrix for calculat. of coeff.

- Consider Newton interpolant $p(x) := \sum_{i=0}^{n} a_i N_i(x)$. The coeff. a_i can be found by solving $Ma = y$ (y = Interpolants)

with:
$$M := \begin{bmatrix} 1 & 0 & \cdot & \cdot \\ 1 & N_1(x_1) & 0 & \\ 1 & N_1(x_2) & N_2(x_2) & \\ \vdots & & \vdots & \\ 1 & N_1(x_t) & N_2(x_t) & \cdots & N_n(t_n) \end{bmatrix} = \begin{bmatrix} 1 & & & \\ 1 & (x_1 - x_0) & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & \\ \vdots & \vdots & \vdots & \\ 1 & (x_n - x_0) & \cdots & \prod_{i=0}^{n-1}(x_j - x_i) \end{bmatrix}$$

$\Rightarrow$ i-th column is the same as $i-1$, except it gets mult. by $(x_i - x_{j-1})$

Code:
```
· Eigen::MatrixXd A = Eigen::MatrixXd::Zero(n+1, n+1);
A.col(0) = Eigen::VectorXd::Ones(n+1);
for (int j=1; j<=n; j++)
    for (int i=j; i<=n; i++)
        A(i,j) = A(i, j-1) * (_x(i) - _x(j-1));    // _x = Nodes
_a = A.triangularView<Eigen::Lower>().solve(y);  // get coeff.
```

Systems of ODEs

- Given a lin. homogenous System of ODEs with $\dot{y} = Ay$, the solution is given by $y(t) = e^{Ax} \cdot \vec{C}$ with C = Vector of integration constants.
- For the system $\dot{y} = Ay$, $y(0) = y_0$, the solution is given by $y(x) = e^{Ax} \cdot \vec{y_0}$
- e^{Ax} is called the **exponential / Fundamental matrix** and in the following it's written how it is calculated.

• Case 1: A is diagonal

- For $A = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$, the matrix e^A is calculated by $e^A = \text{diag}(e^{\lambda_1}, e^{\lambda_2}, \dots, e^{\lambda_n})$ $[\lambda_i \Rightarrow e^{\lambda_i}]$

• Case 2: A can be diagonalized

- If A can be diagonalized, take the EW-decomposition of $A = V \Lambda V^{-1}$, with $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$
- Set $B = \text{diag}(e^{\lambda_1}, e^{\lambda_2}, \dots, e^{\lambda_n})$ (same as prev. case) and get

$$e^A = V \cdot B \cdot V^{-1}$$

Fourier Transform

- Spatial domain → Frequency domain
- In the frequency domain are „peaks" at the frequencies that make up the signal in the spatial domain
  e.g. Wave of 440 Hz + 600 Hz ⇒ 2 peaks, one at 440 and one at 600.

Inverse FFT via FFT:

1) Take complex input and conjugate it (swap sign of imaginary part
2) FFT on that number
3) Swap sign of imaginary part
4) Divide real and imaginary by n (n = „size" of FFT)

C++ Stuff

Size of Eps: Single Precision: $2^{-24} \approx 6 \cdot 10^{-8}$ / Double P.: $2^{-53} \approx 1.1 \cdot 10^{-16}$

Sorting in C++:

- Use std::sort(v.begin(), v.end()); for vector/array v when „way of sorting" is trivially clear (e.g. Int Array)
- Use std::sort(v.begin(), v.end(), sorter); when you want to use your own sorting function (no () at sorting function!).
  ↳ bool sorter(int i, int j){ return i>j } would sort desc.
- Use the std::sort with a lambda function, e.g. to sort asc.:
  std::sort(v.begin(), v.end, [](auto x, auto y){ return x< y;});

Lambda Expressions in C++:

- standalone: std::function<a (b) > f = [captures](args){ code };
  where <a (b)> : a = return type, b = argument types (<void(void)>) for lambdas

Compile Templates

- g++ filename.cpp / g++ -I /path/to/eigen/ filename.cpp

Initializer Lists in C++

The constructor Classname(int i, int j) : x(i), y(j) {} would take 2 arguments (i, j) and set the member variables x and y to their respective value.

Std::setprecision: Use std::setprecision(d) to set cout to print up to d decimal points (useful e.g. for debug/printing)

Eigen Code Stuff

- **Vector** of size n: VectorXd vec(n);
- **Matrix** of size m×n: MatrixXd mat(m, n); (m = rows; n = cols)
- **Identity matrix**: MatrixXd::Identity(rows, cols); or M.setIdentity(rows, cols)
- **Zero matrix**: MatrixXd::Zero(rows, cols) or M.setZero(rows, cols)
- **Ones matrix**: MatrixXd::Ones(rows, cols) or M.setOnes(rows, cols)
- **Linear Spaced Vector**: VectorXd::LinSpaced(size, low, high)
- **First n elements in Vector**: vec.head(n)
- **Last n elements in Vector**: vec.tail(n)
- **n elements, starting at pos i**: vec.segment(i, n)
- **Block of matrix**: M.block(i, j, rows, cols)
- **i-th row of Matrix**: M.row(i)
- **j-th col of Matrix**: M.col(j)
- **Transpose a Matrix**: M.transpose()

- Adjoint Matrix: M.adjoint()
- Elementwise Multiplication: M.cwiseProduct(Q)
  → Works for Matrix / Matrix and Vector / Vector (same size!)
- Elementwise Division: M.cwiseQuotient(Q) (same as Mult.)
- Max Element: M.minCoeff()
- Min Element: M.maxCoeff()
- Sum: M.sum()
- Norm: v.norm()
- Dot Product $(a \cdot b)$: a.dot(b) ⇒ scalar
- Cross Product $(a \times b)$: a.cross(b) ⇒ Vector

Eigen Decompositions

Let A be Matrix. Want to solve $Ax = b$

LU - Decomposition

Compute $= O(n^3)$, Solve $= O(n^2)$

- FullPivLU solver: x = A.fullPivLU().solve(b)
  ↳ Works for all matrices / slow
- PartialPivLU solver: x = A.partialPivLU().solve(b)
  ↳ Only works for invertible matrices / fast

QR - Decomposition

Compute $= O(mn^2)$, Solve $= O(n^2)$

- Householder QR solver: x = A.householderQr().solve(b)
  ↳ Works for all matrices / fast
- FullPivHouseholder QR solver: x = A.fullPivHouseholderQr().solve(b)
  ↳ Also works for all matrices / slow

Triangular - View Solvers

- Upper: x = A.triangularView<Upper>().solve(b)
- Lower: x = A.triangularView<Lower>().solve(b)

Cholesky Decomposition

Comp $= O(\frac{1}{6}n^3 + n^2)$ Solve $= O(n^2)$

- Normal Cholesky decomposition $(A = LL^T)$:
  ↳ x = A.llt().solve(b)
    ↳ Only positive definite / fast
- Cholesky decomposition with pivoting $(A = P^T L D L^T P)$
  ↳ x = A.ldlt().solve(b)
    ↳ Only pos/neg semidefinite / fast

Eigen SVD

Decompose A in $U \Sigma V^T$

Jacobi SVD

- Compute: JacobiSVD<MatrixXd> svd(A, ComputeThinU | ComputeThinV);
  ↳ Reduced SVD of A
  - Σ (Sing. Values) = svd.singularValues()
  - U = svd.matrixU()
  - V = svd.matrixV()
- LSQ Solver: x = A.jacobiSvd(ComputeThinU | ComputeThinV).solve(b)

BDC SVD

- Compute: BDCSVD<MatrixXd> svd(A, ComputeThinU | ComputeThinV);
  ↳ Also reduced SVD
- Rest same as Jacobi SVD

Eigen Eigenvalues / Eigenvectors

#include <Eigen/Eigenvalues>

- Compute: EigenSolver<MatrixXd> eig(A)
  ↳ For self-adjoint, use SelfAdjointEigenSolver<>
- Eigenvalues: eig.eigenvalues()
- Eigenvectors: eig.eigenvectors()

Notes about ODEs / Solving ODE of second Order

- If we have a differential equation of second order, defined by:

  1) $\ddot{x} + b\dot{x} + cx = 0$

  2) $x(0) = x_0 \;,\; \dot{x}(0) = v_0$

  we can write it as a system of first order ODEs

  $\dot{y} = Ay \quad y(0) = y_0 = (x_0, v_0)$

  - Set $y_1 = x$ & $y_2 = \dot{x}$. We now have:

    $$\dot{y} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = A \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \text{ with } \dot{y}_1 = y_2 \Rightarrow \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = A \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

  - The upper row is trivially $(0, 1)$, to get $\dot{x} = \dot{x}$, and the second row is given by $\ddot{x} = -cx - b\dot{x} \Rightarrow (-c, -b)$

  $\rightarrow A = (0, 1; -c, -b)$

Solving an ODE of n-th Order with euler

- Given: a linear, homogen DE of n-th order:

  $a_n y^{(n)} + a_{n-1} y^{(n-1)} + \dots + a_1 y^{(1)} + a_0 y = 0$

- This can be solved using

  $y(x) = e^{\lambda x}$

  where $\lambda \in \mathbb{C}$ is a param which we want to find.

- By some arithmetic operations we get:

  $a_n \lambda^n + a_{n-1} \lambda^{n-1} + \dots + a_1 \lambda + a_0 = 0$

  This is the characteristic polynom of the diff. equation

- Calculate all r different zeroes λ_k with multiplicity m_k.

  A m-fold zero λ has the m solutions:

  $e^{\lambda x}, x \cdot e^{\lambda x}, \dots, x^{m-1} \cdot e^{\lambda x}$

- The solution of the ODE then is given by a linear comb. of these solutions, e.g:

  $y(x) = Ae^{\lambda_1 x} + Bx \cdot e^{\lambda_1 x} + Ce^{\lambda_2 x}$

Implicit / Explicit Euler for ODEs

- The explicit euler method $y_{k+1} = y_k + h \cdot A \cdot y_k$ can be written as

  $y_{k+1} = (\mathbb{I} + hA) \cdot y_k$

  or in C++ with Eigen:

```
MatrixXd B = MatrixXd::Identity(n,n) + h * A;
y_new = B * y_old;
```

- The implicit euler method $y_{k+1} = y_k + h \cdot A \cdot y_{k+1}$ can be written as:

  $B = \mathbb{I} - hA \Rightarrow B \cdot y_{k+1} = y_k$

  or in C++ with Eigen:

```
MatrixXd B = MatrixXd::Identity(n,n) - h * A;
y_new = B.FullPivLu().solve(y_old);
```

Implicit Mid-Point for ODEs

- The implicit midpoint method

  $y_{k+1} = y_k + h \cdot A \cdot (\frac{1}{2}(t_k + t_{k+1}), \frac{1}{2}(y_k + y_{k+1}))$ can be written as:

  $y_{k+1} = (I - \frac{h}{2}A)^{-1} \cdot (y_k + \frac{h}{2} A y_k)$

  or in C++ with Eigen:

```
MatrixXd B = MatrixXd::Identity(n,n) - h*0.5*A;
y_new = B.FullPivLu().solve(y_old + h*0.5*A*y_old);
```

Kahan Sum (summing up numbers)

```
float kahan_sum(vector<float> &v)
    float sum = 0, e = 0;
    for (float x : v)
        e += x;
        float tmp = sum + e;
        e += sum - tmp;
        sum = tmp;
    return sum;
```

Sherman-Morrison-Woodbury for inverting Matrices

- Let $A \in \mathbb{R}^{n \times n}$ be a square, invertible matrix and $u, v \in \mathbb{R}^n$ be column vectors.

  If $A + uv^T$ is invertible, its inverse is given by:

$$(A + uv)^T = A^{-1} - \frac{A^{-1} uv^T A^{-1}}{1 + v^T A^{-1} u}$$

- This version is useful if a matrix can be made up of another matrix plus some matrix constructed by two vectors.

Gradient & Hessian Matrix

Hessian Matrix $H_f(x) := \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right)$ is the matrix containing all second-order derivatives of f at point x.

$$H_f(x) := \begin{pmatrix} \frac{\partial f}{\partial x_1 \partial x_1}(x) & \frac{\partial f}{\partial x_1 \partial x_2}(x) & \cdots & \frac{\partial f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial f}{\partial x_2 \partial x_1}(x) & \frac{\partial f}{\partial x_2 \partial x_2}(x) & \cdots & \frac{\partial f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f}{\partial x_n \partial x_1}(x) & \frac{\partial f}{\partial x_n \partial x_2}(x) & \cdots & \frac{\partial f}{\partial x_n \partial x_n}(x) \end{pmatrix}$$

Gradient

The gradient $grad(f) = \nabla f$ is the vector of all partial derivatives of f:

$$\nabla(f) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

Derivatives of Vectors & Matrices

Let $x, y \in \mathbb{R}^n$ be vectors; $\alpha, \beta \in \mathbb{R}$ scalars; $A, B \in \mathbb{R}^{m \times n}$ be matrices and $f: \mathbb{R}^n \to \mathbb{R}^m$ be a function:

- Linearity: $(\alpha f(x) + \beta g(x))' = \alpha f'(x) + \beta g'(x)$
- Chain rule: $(f(g(x)))' = f'(g(x)) \cdot g'(x)$
- Product rule: $(f(x)^T g(x)) = f(x) g'(x) + g(x)^T f'(x)$

- Ax $dx = A$
- $x^T A x$ $dx = x^T(A + A^T)$
- A^{-1} $dA = ((A^{-1})^T \otimes A^{-1})$
- $\|x\|^2$ $dx = 2x^T$
- $\|x\|$ $dx = x^T / \|x\|$

Using the tests provided

- Run tests with:

  ./a.out < tests.txt