

Numerical Methods Summary

October 8, 2020

Chapter 1

Computing with Matrices and Vectors

Elementary Operations: $+, -, *, \backslash$, lowest level of real arithmetic available on computers usually implemented in hardware

Elementary Linear Algebra operations: The next level real arithmetic which is the computation on finite arrays of real numbers

Complex Algorithms: involves iterations and approximations

1.1 Fundamentals

\mathbb{K} notation for a generic field of numbers i.e from \mathbb{R} or \mathbb{C}

Vectors: one-dimensional array of real/complex numbers, the default for this lecture are column vectors:

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{K}^n \quad \text{column vector} \qquad [x_1 \cdots x_n] \in \mathbb{K}^{1,n} \quad \text{row vector}$$

Unless stated otherwise, in mathematical formulas vector components are indexed from 1

Notations:

- column vectors: bold small roman letters e.g $\mathbf{x}, \mathbf{y}, \mathbf{z}$
- row vectors: x^T, y^T, z^T
- Addressing vector components:

$$x = [x_1, \dots, x_n]^T \rightarrow x_i, i = 1, \dots, n$$
$$x \in \mathbb{K}^n \rightarrow (x)_i, i = 1, \dots, n$$

- Selecting sub-vectors: $(x)_{k:l} = [x_k, \dots, x_l]^T, 1 \leq k \leq l \leq n$
- j-th unit vector: $(e_j)_i = \sigma_{ij}$ where $\sigma_{ij} := 1$ if $i = j$ 0 else (Kronecker symbol)

1.2 Software and Libraries

Eigen: A header-only C++ template library designed to enable easy, natural and efficient numerical linear algebra

Header-Only A library is called header-only if the full definitions of all macros functions and classes comprising the library are visible to the compiler in a header file form. Header-only libraries do not need to be separately compiled, packaged and installed in order to be used. All that is required is to point the compiler at the location of the headers, and then include the header files into the application source

Header File form: Many programming languages and other computer files have a directive, often called `include` (or `copy, import`) that causes the contents of a second file to be inserted into the original file. These included files are called copybooks or header files.

Eigen Cheet Sheet:

```

// A simple quickref for Eigen. Add anything that's missing.
// Main author: Keir Mierle

#include <Eigen/Dense>

Matrix<double, 3, 3> A; // Fixed rows and cols. Same as Matrix3d.
Matrix<double, 3, Dynamic> B; // Fixed rows, dynamic cols.
Matrix<double, Dynamic, Dynamic> C; // Full dynamic. Same as MatrixXd.
Matrix<double, 3, 3, RowMajor> E; // Row major; default is column-major.
Matrix3f P, Q, R; // 3x3 float matrix.
Vector3f x, y, z; // 3x1 float matrix.
RowVector3f a, b, c; // 1x3 float matrix.
VectorXd v; // Dynamic column vector of doubles
double s;

// Basic usage
// Eigen // Matlab // comments
x.size() // length(x) // vector size
C.rows() // size(C,1) // number of rows
C.cols() // size(C,2) // number of columns
x(i) // x(i+1) // Matlab is 1-based
C(i,j) // C(i+1,j+1) //

A.resize(4, 4); // Runtime error if assertions are on.
B.resize(4, 9); // Runtime error if assertions are on.
A.resize(3, 3); // Ok; size didn't change.
B.resize(3, 9); // Ok; only dynamic cols changed.

A << 1, 2, 3, // Initialize A. The elements can also be
4, 5, 6, // matrices, which are stacked along cols
7, 8, 9; // and then the rows are stacked.
B << A, A, A; // B is three horizontally stacked A's.
A.fill(10); // Fill A with all 10's.

// Eigen // Matlab
MatrixXd::Identity(rows,cols) // eye(rows,cols)
C.setIdentity(rows,cols) // C = eye(rows,cols)
MatrixXd::Zero(rows,cols) // zeros(rows,cols)
C.setZero(rows,cols) // C = zeros(rows,cols)
MatrixXd::Ones(rows,cols) // ones(rows,cols)
C.setOnes(rows,cols) // C = ones(rows,cols)
MatrixXd::Random(rows,cols) // rand(rows,cols)*2-1 // MatrixXd::Random returns uniform random numbers in (-1, 1).
C.setRandom(rows,cols) // C = rand(rows,cols)*2-1
VectorXd::LinSpaced(size,low,high) // linspace(low,high,size)
v.setLinSpaced(size,low,high) // v = linspace(low,high,size)
VectorXd::LinSpaced((hi-low)/step)+1, // low:step:hi
low,low+step*(size-1)) // 

// Matrix slicing and blocks. All expressions listed here are read/write.
// Templated size versions are faster. Note that Matlab is 1-based (a size N
// vector is x(1)...x(N)).
// Eigen // Matlab
x.head(n) // x(1:n)
x.head<n>() // x(1:n)
x.tail(n) // x(end - n + 1: end)
x.tail<n>() // x(end - n + 1: end)
x.segment(i, n) // x(i+1 : i+n)
x.segment<n>(i) // x(i+1 : i+n)
P.block(i, j, rows, cols) // P(i+1 : i+rows, j+1 : j+cols)
P.block<rows, cols>(i, j) // P(i+1 : i+rows, j+1 : j+cols)
P.row(i) // P(i,:)
P.col(j) // P(:, j+1)
P.leftCols<cols>() // P(:, 1:cols)
P.leftCols<cols> // P(:, 1:cols)
P.middleCols<cols>(j) // P(:, j+1:j+cols)
P.middleCols(j, cols) // P(:, j+1:j+cols)
P.rightCols<cols>() // P(:, end-cols+1:end)
P.rightCols<cols> // P(:, end-cols+1:end)
P.topRows<rows>() // P(1:rows, :)
P.topRows<rows> // P(1:rows, :)
P.middleRows<rows>(i) // P(i+1:rows, :)
P.middleRows(i, rows) // P(i+1:rows, :)
P.bottomRows<rows>() // P(end-rows+1:end, :)
P.bottomRows<rows> // P(end-rows+1:end, :)
P.bottomRows<rows>(rows) // P(end-rows+1:end, :)
P.topLeftCorner<rows, cols> // P(1:rows, 1:cols)
P.topRightCorner<rows, cols> // P(1:rows, end-cols+1:end)
P.bottomLeftCorner<rows, cols> // P(end-rows+1:end, 1:cols)
P.bottomRightCorner<rows, cols> // P(end-rows+1:end, end-cols+1:end)
P.topLeftCorner<rows, cols>() // P(1:rows, 1:cols)
P.topRightCorner<rows, cols>() // P(1:rows, end-cols+1:end)
P.bottomLeftCorner<rows, cols>() // P(end-rows+1:end, 1:cols)
P.bottomRightCorner<rows, cols>() // P(end-rows+1:end, end-cols+1:end)
P.bottomRightCorner<rows, cols>() // P(end-rows+1:end, end-cols+1:end)

// Of particular note is Eigen's swap function which is highly optimized.
// Eigen // Matlab
R.row(i) = P.col(j); // R(i, :) = P(:, j)
R.col(j).swap(mat1.col(j2)); // R(:, [j1 j2]) = R(:, [j2, j1])

// Views, transpose, etc;
// Eigen // Matlab
R.adjoint() // R'
R.transpose() // R.' or conj(R') // Read-write
R.diagonal() // diag(R) // Read-write
x.asDiagonal() // diag(x)
R.transpose().colwise().reverse() // rot90(R) // Read-write
R.rowwise().reverse() // fliplr(R)
R.colwise().reverse() // flipud(R)
R.replicate(i,j) // repmat(P,i,j)

// All the same as Matlab, but matlab doesn't have *= style operators.
// Matrix-vector. Matrix-matrix. Matrix-scalar.
Y = M*x; // M*x;
R = P^Q; // P^s;
a = b*M; // R = P - Q; // R = s^4P;
a *= M; // R = P + Q; // R = P/s;
R *= Q; // R = Q; // R = s^2P;
R += Q; // R = Q; // R *= s;
R -= Q; // R = Q; // R /= s;

// Vectorized operations on each element independently
// Eigen // Matlab
R = P.cwiseProduct(Q); // R = P .* Q
R = P.array() ^ s.array(); // R = P.^ s
R = P.cwiseQuotient(Q); // R = P ./ Q
R = P.array() / Q.array(); // R = P ./ Q
R = P.array() + s.array(); // R = P + s
R = P.array() - s.array(); // R = P - s
R.array() += s; // R = R + s
R.array() -= s; // R = R - s
R.array() < Q.array(); // R < Q
R.array() <= Q.array(); // R <= Q
R.cwiseInverse(); // 1 ./ P
R.array().inverse(); // 1 ./ P
R.array().sin(); // sin(P)
R.array().cos(); // cos(P)

```

The Fundamental type of Eigen is the Matrix. There are two types of matrices:

- **fixed size:** size known at compile time
- **dynamic:** size known only at run time

Tensor product A column vector multiplied with a row vector

Dot Product Row vector multiplied with a column vector

1.3 1.2.3 Matrix Storage Formats

The entries of a (generic, dense i.e every entry matters) $A \in \mathbb{K}^{m,n}$ are stored in a contiguous linear array of size $m \cdot n$. An exception is structured/sparse matrices i.e matrices which have a certain structure or few non zero entries.

By Default Eigen stores matrices in Column major format as all the elements in this format are contiguous in memory. In Row major format the elements of the Matrix are scattered which results in cache misses. Hence Column major format is more efficient

Accessing Arrays in Eigen:

- $A(i)$ → reference to the i -th element of the array
- $A.data()$ → raw pointer

We declare a $n \times n$ matrix in Eigen like:

```
Eigen:: MatrixXd A = Eigen:: MatrixXd::Random(n,n)
```

Row/Col Access (j -th row):

```
A.row(j)  
A.col(j)
```

1.3.1 Raw Pointers

A pointer is a type of variable. It stores the address of an object in memory and is used to access that object in memory and is used to access that object.

A **Raw Pointer** is a pointer whose lifetime is not controlled by an encapsulating object . A raw pointer can be assigned the address of another non-pointer variable, or it can be assigned a value of `nullptr`.

```
int* p = nullptr; // declare pointer and initialize it  
                  // so that it doesn't store a random address  
int i = 5;  
p = &i; // assign pointer to address of object  
int j = *p; // dereference p to retrieve the value at its address
```

1.4 1.4 Computational Effort

Traditional: number of elementary operations

Modern: The computational effort involved in a run of a numerical code is only loosely related to the overall execution time on modern computers. It is mainly determined by the memory access pattern and vectorization/pipelining.

1.4.1 1.4.1 Asymptotic complexity

Characterises the worst-case dependence of its computational effort on one or more problem size parameters when these tend to ∞

Notation: Landau-O notation

We define the computational effort as $\text{Cost}(n) = (O)(n^\alpha), \alpha > 0 \iff \exists C > 0, n_0 \in \mathbb{N} : \text{cost}(n) \leq Cn^\alpha \forall n > n_0$

Tacit Sharpness assumption: $\text{cost}(n) \neq \mathcal{O}(n^\beta), \forall \beta < \alpha$

Asymptotic complexity predicts the dependence of runtime on problem size for large problems, because for small problem sizes we can use the caches and hence we do not have the memory access bottleneck.

1.4.2 Computational Cost of basic numerical LA operations

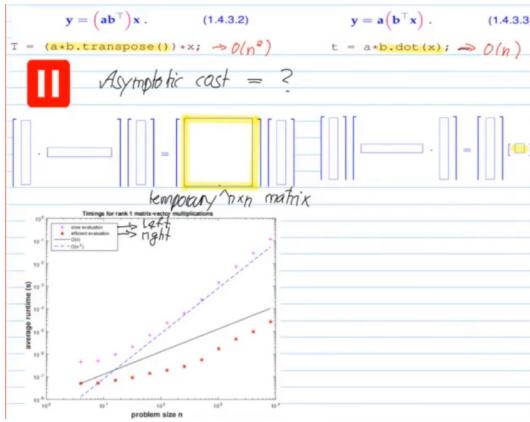
operation	description	#mul/div	#add/sub	asym. complexity
dot product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^\top \mathbf{y}$	n	$n - 1$	$O(n)$
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^\top$	nm	0	$O(mn)$
Matrix \times vector	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m,n}) \mapsto \mathbf{Ax}$	mn	$(n-1)m$	$O(mn)$
matrix product ^(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{AB}$	mnk	$mk(n-1)$	$O(mnk)$

The matrix multiplication is implemented with a triple loop and hence not optimal.

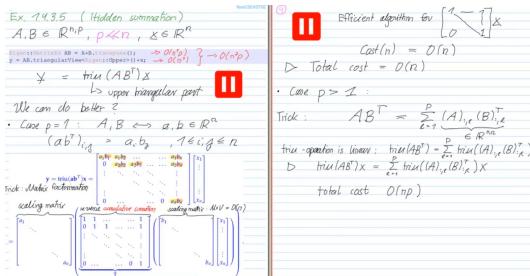
A doubly logarithmic plot is scaled logarithmically on the x and y axis. If $\text{Cost}(n) = N(n^\alpha)$ then the data points aligned in a doubly logarithmic plot correspond to a straight line.

1.4.3 Tricks to improve complexity

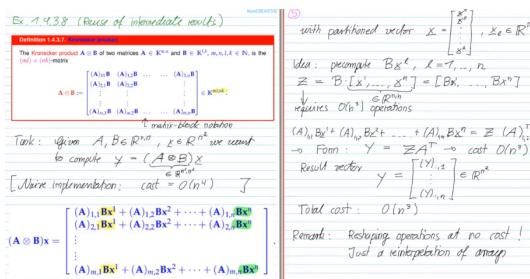
- Exploit associativity



- Hidden summation:



- Reuse of intermediate results



1.5 1.5 Machine Arithmetic and Consequences

Computers can only calculate with finite numbers and hence cannot calculate with real numbers. Computers compute with machine numbers denoted: M The set M has two properties:

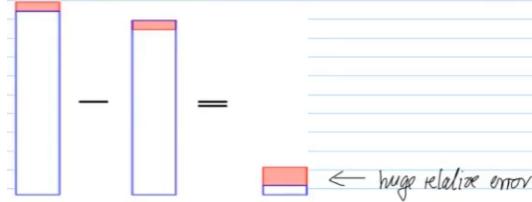
- finite i.e there is a lowest and a highest machine number any numbers larger or smaller than these bounds result in overflow/underflow
- discrete in ℝ hence there are gaps in the Machine numbers. When working in ℝ our computer must round the numbers such that it can work with them.

Roundoff will effect any result of a numerical calculation. Hence a result which should be 0 might not be zero. Therefore we cannot test $== 0$ we must test for relative smallness

1.6 1.5.4 Cancellation

EPS- Errors: Max error you can have in a Machine set of numbers

EPS-sized errors are troublesome because of error amplification.



Cancellation: Extreme amplification of relative errors during the subtraction of numbers of equal size.

1.6.1 Avoiding Cancellation:

Rewrite expressions inequivalent form immune to cancellation. For x close to 1 cancellation is harmless.

1.7 Numerical Stability

Problem A function/mapping $F : X \rightarrow Y$ where X is the data space (input) and Y is the result space (output), in this course we usually consider normed Vector space. On \mathbb{R}^n we have vector norms:

- $\|\cdot\|_2$ Euclidean Norm
- $\|\cdot\|_1$ One Norm (sum of the absolute value)
- $\|\cdot\|_\infty$ Maximum Norm (max value)

The vector norms induce **Matrix Norms**. **D1.5.5.10 Matrix Norm:** Given vector norms $\|\cdot\|_x$ and $\|\cdot\|_y$ on \mathbb{K}^n and \mathbb{K}^m , respectively, the associated matrix norm is defined by

$$M \in \mathbb{R}^{m,n} : \|M\| := \sup_{x \in \mathbb{R}^n \setminus \{0\}} \frac{\|Mx\|_y}{\|x\|_x}$$

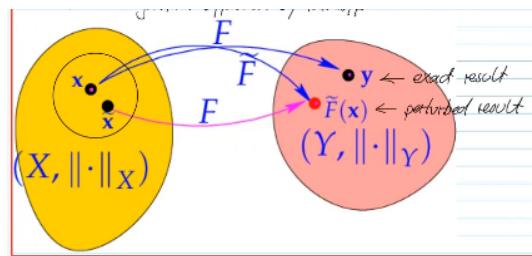
Norms are used to understand the size of perturbations.

Stable Algorithm: An algorithm \tilde{F} for solving a problem $F : X \mapsto Y$ is **numerically stable** if for all $x \in X$ its result $\tilde{F}(x)$ (possibly affected by roundoff) is the exact result for "slightly perturbed" data:

$$\exists C \approx 1 : \forall x \in X : \exists \tilde{x} \in X : \|x - \tilde{x}\|_X \leq C w(x) EPS \|x\|_X \wedge \tilde{F}(x) = F(\tilde{x})$$

Where:

- \tilde{F} : Algorithm affected by roundoff
- \tilde{x} : perturbed data
- $C w(x)$: number of operations during the execution of the algorithm
- EPS: machine precision



The impact of roundoff on a stable algorithm is of the same order of magnitude as the effect of the inevitable perturbations due to rounding the input data. When talking about perturbations we mean relative perturbations.

Sensitive Dependence: Given a slight perturbation the result varies largely:

$$\|F(x) - F(\tilde{x})\| \text{ can be large for tiny } \|x - \tilde{x}\|_X$$

Chapter 2

Chapter 2: Direct Methods for Linear Systems of Equations

2.1 Intro and Theory of Linear Systems of Equations (LSE's)

LSE:

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

where:

- $A \in \mathbb{K}^{n,n}$ a square system/coefficient matrix
- $\underline{x} \in \mathbb{K}^n$
- $\underline{b} \in \mathbb{K}^n$

A being a square matrix is important, because this ensures that the number of equations is equal to the number of unknowns

2.2 Square LSE

2.2.1 Existence and uniqueness of solutions

1. If A is invertible (regular) :

$$Ax = b \iff \underline{x} = A^{-1}\underline{b}$$

Do not use matrix inversion to solve LSE with numerical libraries

A matrix $A \in \mathbb{K}^{n,n}$ is regular \iff :

- $\det A \neq 0$
- columns of A are linearly independent
- rows of A are linearly independent
- $N(A) = \{\underline{z} \in \mathbb{K}^n : A\underline{z} = \underline{0}\} = \{\underline{0}\}$

2.3 Theory: Linear Systems of Equations

2.3.1 Sensitivity/conditioning of LSE

Sensitivity: Quantities how small (relative) perturbation of data lead to changes of the output

Vector Norms & (induced) Matrix norms $\|\cdot\|$

- $\|M\| = \sup_{\underline{x} \neq 0} \frac{\|M\underline{x}\|}{\|\underline{x}\|} \Rightarrow \|M\underline{x}\| \leq \|M\| \cdot \|\underline{x}\|$
- $\|\underline{x} + \underline{y}\| \leq \|\underline{x}\| + \|\underline{y}\|$
- $\|\underline{y} - \underline{y}'\| \geq \|\underline{x}\| - \|\underline{y}\|$

Sensitivity strongly depends on the choice of norms

Lemma 2.2.2.5 Perturbation lemma

$$B \in \mathbb{R}^{n,n}, \|B\| < 1 \Rightarrow I + B \text{ regular} \wedge \|(I + B)^{-1}\| \leq \frac{1}{1 - \|B\|}$$

T 2.2.2.4 Conditioning of LSE's If A is regular, $\|\Delta A\| < \|A^{-1}\|^{-1}$ and (2.2.2.3), then

- $A + \Delta A$ is regular/invertible

$$\bullet \frac{\|\underline{x} - \Delta \underline{x}\|}{\|\underline{x}\|} \leq \frac{\|A^{-1}\| \cdot \|A\|}{1 - \|A^{-1}\| \|A\| \cdot \frac{\|\Delta A\|}{\|A\|}} \cdot \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right)$$

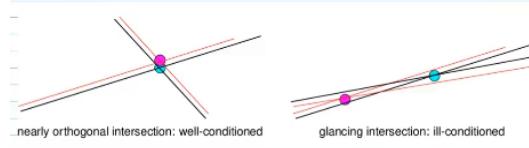
where:

- $\frac{\|x - \Delta x\|}{\|x\|}$ is the relative error of the result
- $(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|})$ are the relative perturbations

D 2.2.2.7 Condition number of a matrix

$$\text{Cond}(A) := \|A^{-1}\| \cdot \|A\|$$

- **LSE: Well conditioned** $\text{Cond}(A) \approx 1$
- **LSE: Ill-conditioned** $\text{Cond}(A) \gg 1$



$\text{cond}(A) \gg 1 \iff \text{columns/rows of } A \text{ 'almost linearly dependent'}$

2.4 2.3-2.5: Gaussian Elimination

Gauss Elim: Successive row manipulation of LSE $A\underline{x} = \underline{b}$ to convert it to triangular form. It consists of 2 steps:

1. **forward elimination:** \mathcal{O}^3
2. **back substitution:** \mathcal{O}^2

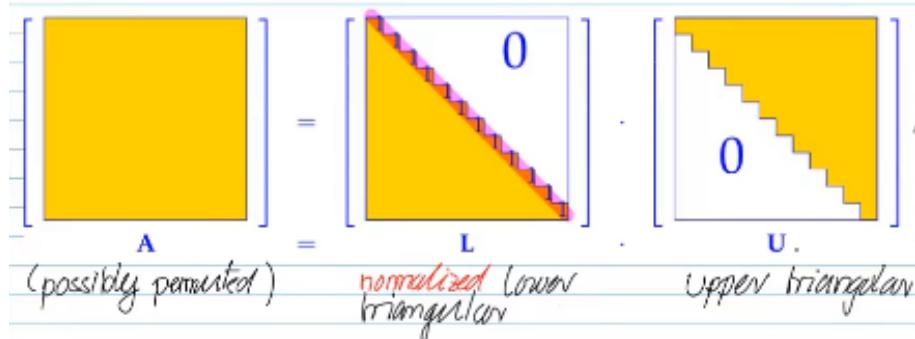
Block Gauss Elim:

Reminder: Matrix multiplication is not commutative!

Given the LSE $A\underline{x} = \underline{b}$, where $A \in \mathbb{R}^{n,n}$ we split A into blocks where $A_{1,1} \in \mathbb{R}^{n,n}$ and regular. We can then apply Gauss like for scalars with the exception of commutivity

LU-Decomposition

equivalent to GE i.e $\text{cost(LU)} = \mathcal{O}(n^3)$



Definition 2.3.2.3. LU-decomposition/LU-factorization

Given a square matrix $\mathbf{A} \in \mathbb{K}^{n,n}$, an **upper triangular matrix** $\mathbf{U} \in \mathbb{K}^{n,n}$ and a **normalized lower triangular matrix** (\rightarrow Def. 1.1.2.3) form an LU-decomposition/LU-factorization of \mathbf{A} , if $\mathbf{A} = \mathbf{LU}$.

We can solve a LSE with LU-decomposition using the following steps:

$$\begin{array}{l} \text{Ax} = \mathbf{b} : \quad \begin{array}{l} \text{① LU-decomposition } \mathbf{A} = \mathbf{LU}, \# \text{elementary operations} = \frac{1}{2}n(n-1)(n+1) \Rightarrow \mathcal{O}(n^3) \\ \text{② forward substitution, solve } \mathbf{Lz} = \mathbf{b}, \# \text{elementary operations} = \frac{1}{2}n(n-1) \\ \text{③ backward substitution, solve } \mathbf{Ux} = \mathbf{z}, \# \text{elementary operations} = \frac{1}{2}n(n+1) \end{array} \end{array} \Rightarrow \mathcal{O}(n^3)$$

With eigen we can solve LSE with multiple right hand sides i.e we are solving the equation:

$$AX = B \text{ with } X, B \in \mathbb{R}^{n,l}$$

Eigen: $X = \mathbf{A}.\text{lu}().\text{solve}(\mathbf{B})$

Where $\text{lu}()$ calculates the LU-decomposition and $\text{solve}(\mathbf{B})$ computes the forward and backward substitution for the columns of \mathbf{B}

C++ code 2.5.0.11: Wasteful approach! → GITLAB <pre> 1 // Setting: N ≫ 1, 2 // large matrix A ∈ ℝ^{n,n} 3 for(int j = 0; j < N; ++j){ 4 x = A.lu().solve(b); 5 b = some_function(x); 6 } </pre> <p style="text-align: center;">computational effort $\mathcal{O}(Nn^3)$</p>	C++ code 2.5.0.12: Smart approach! → GITLAB <pre> 1 // Setting: N ≫ 1, 2 // large matrix A ∈ ℝ^{n,n} 3 auto A_lu_dec = A.lu(); 4 for(int j = 0; j < N; ++j){ 5 x = A_lu_dec.solve(b); ↗ 6 b = some_function(x); 7 } </pre> <p style="text-align: center;">computational effort $\mathcal{O}(n^3 + Nn^2)$</p>
 setup phase (factorization) Cost: $\mathcal{O}(n^3)$	 elimination phase (forward/backward substitution) Cost: $\mathcal{O}(n^2)$

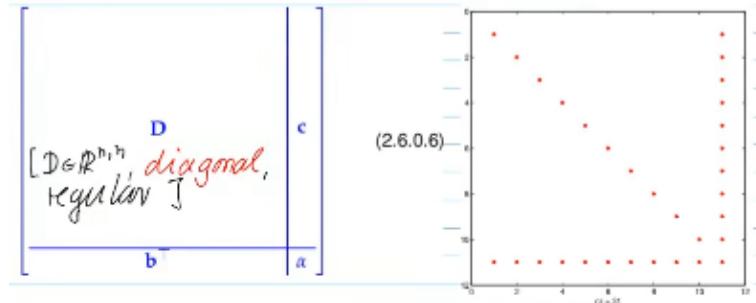
Block LU-Decomposition

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{bmatrix} \quad \text{with Schur complement } \mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}.$$

block LU-factorization

2.5 Exploiting Structure when solving Linear Systems

LSE with arrow matrix: An arrow matrix is a matrix with the following form:



We can solve this LSE with either:

- LU-decomposition resulting in a runtime of $\mathcal{O}(n^3)$
- Block GE resulting in a runtime of $\mathcal{O}(n)$

The second method is faster but more vulnerable to round-off errors i.e it is more instable

LSE subject to low-rank modification: Assume $\mathbf{Ax} = \mathbf{b}$ is easy to solve, because :

1. \mathbf{A} has a special structure
2. LU-dec is available

Our goal is to solve $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \mathbf{b}$ where $\tilde{\mathbf{A}}$ arises from \mathbf{A} by changing a single entry $(\mathbf{A})_{i^*j^*}$

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{u}\mathbf{v}^T, \mathbf{u}, \mathbf{v} \in \mathbb{R}^n$$

Using a modified Block elimination we get the equation:

$$\tilde{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b} - \frac{\mathbf{A}^{-1}\mathbf{u}(\mathbf{v}^H(\mathbf{A}^{-1}\mathbf{b}))}{1 + \mathbf{v}^H(\mathbf{A}^{-1}\mathbf{u})}$$

The generalization of this formula is given by:

Lemma 2.6.0.21. Sherman-Morrison-Woodbury formula

For regular $\mathbf{A} \in \mathbb{K}^{n,n}$, and $\mathbf{U}, \mathbf{V} \in \mathbb{K}^{n,k}$, $n, k \in \mathbb{N}$, $k \leq n$, holds

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^H)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1},$$

If $\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U}$ is regular.

General rank k-matrix

2.6 Sparse Linear Systems

A sparse matrix is characterized by:

- "almost all" entries = 0
- for which it is worth while exploiting this fact i.e a 2x2 matrix is not sparse

Sparse LSE arise in models of large networks where each node is connected to only a few other nodes.

2.6.1 Sparse Matrix storage formats

A storage format should fulfill the following conditions:

- Memory $\text{nnz}(\mathbf{A})$ (number of non zero elements in \mathbf{A})
- cost ($\mathbf{A} \cdot \mathbf{vector}$) $\text{nnz}(\mathbf{A})$
- provide easily accessible information about the location of non zero entries

[COO/triplet format]: Stores \mathbf{A} as a list/sequence of tuples (i,j,value)

```
struct Triplet {
    size_t i; // row index
    size_t j; // column index
    scalar_t a; // additive contribution to matrix entry
};
using TripletMatrix = std::vector<Triplet>;
```

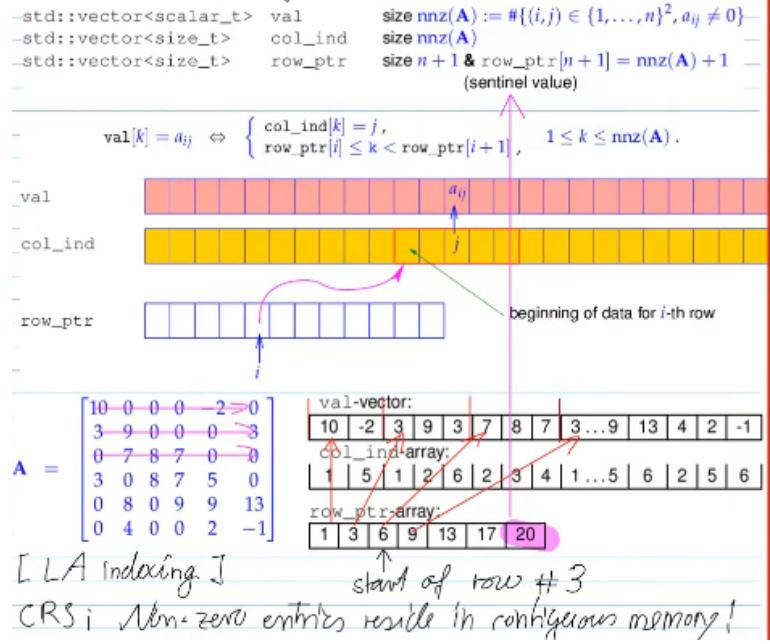
C++-code 2.7.1.2: Matrix×vector product $\mathbf{y} = \mathbf{Ax} + \mathbf{y}$ in triplet format

```
1 void multTriplMatvec(const TripletMatrix &A,
2                         const vector<scalar_t> &x,
3                         vector<scalar_t> &y)
4     for (size_t l=0; l<A.size(); l++) {
5         y[A[l].i] += A[l].a*x[A[l].j];
6     }
```

Allowed: repeated index pairs \rightarrow values summed up

$$(A)_{ij} = \sum_{k: A[k].i=i, A[k].j=j} A[k].a$$

Compressed row storage (CRS)



2.6.2 Sparse Matrices in Eigen

default format is CRS

```
#include <Eigen/Sparse>
Eigen::SparseMatrix<int, Eigen::ColMajor> Asp(rows, cols); // CCS
format
Eigen::SparseMatrix<double, Eigen::RowMajor> Bsp(rows, cols); // CRS
format
```

The challenge is to efficiently initialize the matrix.

Setting random entries is inefficient because it causes massive data movement. Setting an element which was 0 before means we have to insert it in `val`, update `colInd` and `row_ptr` arrays. There are two possibilities to fix this:

1. Use intermediate COO/triplet format i.e we initialize with triplets and then convert to CRS format

```
std::vector <Eigen::Triplet <double>> triplets;
// ... fill the std::vector triplets ...
Eigen::SparseMatrix<double, Eigen::RowMajor> spMat(rows, cols);
spMat.setFromTriplets(triplets.begin(), triplets.end());
O(#triplets) - complexity
```

2. Use `reserve()` & `insert()`, if nnz per row/column is known ahead of time

C++-code 2.7.2.1: Accessing entries of a sparse matrix: potentially inefficient!

```
1 unsigned int rows, cols, max_no_nnz_per_row;
2 ...
3 SparseMatrix<double, RowMajor> mat(rows, cols);
4 mat.reserve(RowVectorXi::Constant(cols, max_no_nnz_per_row)); -> Allocation of
5 // do many (incremental) initializations enough space
6 for ( ) {
7     mat.insert(i,j) = value_ij;
8     mat.coeffRef(i,j) += increment_ij;
9 }
10 mat.makeCompressed(); } O(1), if enough space
                                reserve() d
                                squeeze out zeros
```

2.6.3 Direct Solution of Sparse LSE's

Assume: System matrix is in sparse matrix format i.e tells the location of the zero entries. Hence can be exploited by sparse elimination techniques.

Solving a sparse LSE with Eigen

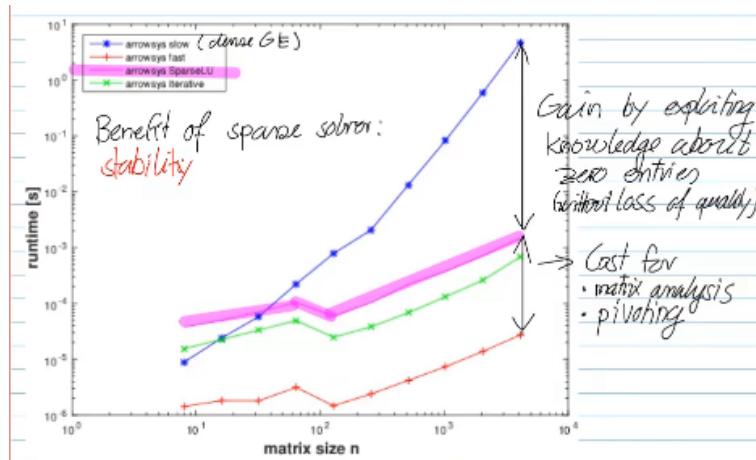
C++-code 2.7.3.1: Function for solving a sparse LSE with EIGEN [→ GITLAB](#)

```

1 using SparseMatrix = Eigen::SparseMatrix<double>;
2 // Perform sparse elimination
3 void sparse_solve(const SparseMatrix &A, const VectorXd &b, VectorXd &x) {
4     Eigen::SparseLU<SparseMatrix> solver(A);
5     if (solver.info() != Eigen::Success) → check this → the expensive part
6         throw "Matrix factorization failed";
7     x = solver.solve(b); → triangular solves cheap
8 }
```

Sparse elimination can fail because:

- The matrix is not singular
- The matrix has an awkward structure



In practice when using sophisticated sparse matrix solver libraries the cost of solving $Ax = b$ is $\mathcal{O}((nnz(A))^\alpha)$ where $\alpha \approx 1.5 - 2.5$

Chapter 3

Direct Methods for Linear Least Squares Problems

Overdetermined LSE: An LSE with more equations than unknowns i.e $m \geq n$

3.1 Overdetermined LiSE's

Linear parameter estimation in 1D We measure 2 quantities connected by a linear physical law

$$y = \alpha x + \beta \text{ for some unknown parameters } \alpha, \beta \in \mathbb{R}$$

We make n measured pairs i.e $(x_i, y_i) \in \mathbb{R}^2, i = 1, \dots, n, n > 2$ in order to determine α, β
if the measurements were precise the following linear equations would be satisfied:

$$y_i = \alpha x_i + \beta \quad \forall i$$

this results in an LSE. However the measured values are affected by measurement errors e.g the y 's are affected

\Rightarrow The RHS might be perturbed

\Rightarrow The RHS is no longer a solution for the matrix $A \Rightarrow$ No solution may exist

Linear regression: linear parameter estimation We are now given the following linear law:

$$y = \underline{a}^T \underline{x} + \beta \text{ with } x, a \in \mathbb{R}^n, \beta \in \mathbb{R}$$

Measurements:

$$(\underline{x}_i, y_i) \in \mathbb{R}^n \times \mathbb{R}, i = 1, \dots, m$$

The system is Overdetermined if $m > n+1$. As in the above example the result might be perturbed and become unsolvable.

Principle in data science: You cannot afford not to use every piece of information available

Angles in triangulations

Triangulation: Covering an area in triangles to determine the distance of points.

We have the following pieces of information available:

- Each angle is supposed to be equal to its measured value
- The sum of interior angles is π for every triangle $\rightarrow +\# \text{triangles}$ extra equations
- The sum of angles at an interior node is $2\pi \rightarrow +\# \text{ interior nodes}$ extra equations

Hence we have an OD LSE

3.2 Least Squares Solution Concepts

Setting: OD-LSE $\underline{A}\underline{x} = \underline{b}$

3.2.1 Least Squares Solutions: Definition

A LSQ solution of $\underline{A}\underline{x} = \underline{b}$ is a vector $x \in \mathbb{R}^n$ that makes the **residual**:

$$r := \underline{b} - \underline{A}\underline{x}$$

as small as possible with respect to the euclidean norm.

Definition 3.1.1.1. Least squares solution

For given $A \in \mathbb{R}^{m,n}$, $b \in \mathbb{R}^m$ the vector $x \in \mathbb{R}^n$ is a **least squares solution** of the linear system of equations $Ax = b$, if

$$x \in \underset{y \in \mathbb{R}^n}{\operatorname{argmin}} \|Ay - b\|_2^2, =: lsg(A, b)$$

↑
need not be unique

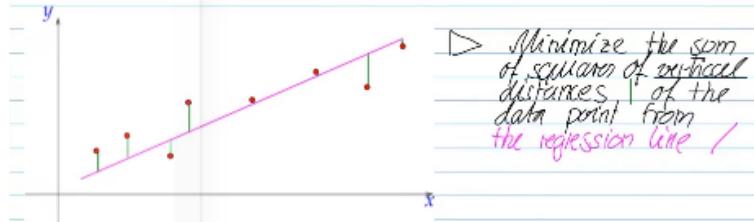
$$\|Ax - b\|_2^2 = \min_{y \in \mathbb{R}^n} \|Ay - b\|_2^2 = \min_{y_1, \dots, y_n \in \mathbb{R}} \sum_{i=1}^m \left(\sum_{j=1}^n (A)_{ij} y_j - (b)_i \right)^2.$$

Generalization: $A \in \mathbb{R}^{n,n}$ regular $\Rightarrow lsg(A, b) := \{A^{-1}, \underline{x}\}$

Ex 3.1.7.5 : 1D linear regression , Ex 3.0.1.1 cont'd

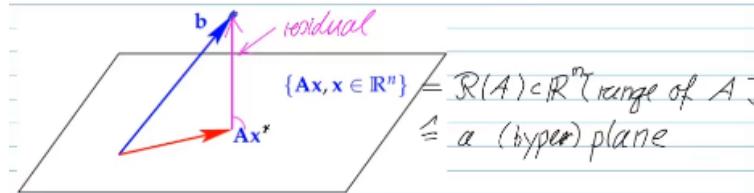
OD-LSE : $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \Leftrightarrow Ax = b, A \in \mathbb{R}^{m,2}, b \in \mathbb{R}^m, x \in \mathbb{R}^2.$ (3.0.1.3)

LSE sol : $\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \in \underset{\alpha, \beta}{\operatorname{argmin}} \sum_{i=1}^m (y_i - \tilde{x}_i \cdot \tilde{\beta})^2$



▷ Minimize the sum of squares of vertical distances of the data points from the regression line!

Geometric interpretation of least squares solution of $Ax = b$



Ax^* is closest to b in $R(A)$, $x^* := LSQ$ sol
 $\Rightarrow Ax^* = b_0$ = orthogonal projection of b onto $R(A)$

Geometric intuition: Since there will always be a point closest to b a least squares solution always exists:

$$\forall A \in \mathbb{R}^{m,n}, b \in \mathbb{R}^m : lsg(A, b) \neq \emptyset$$

3.2.2 Normal Equations

The vector $x \in \mathbb{R}^n$ is a least squares solution of the linear system of equations:

$$Ax = b, A \in \mathbb{R}^{m,n}, b \in \mathbb{R}^m$$

if and only if it solves the **normal equations (NEQ)**

$$A^T Ax = A^T b$$

From this follows:

$$x \in lsg(A, b) \iff A^T Ax = A^T b$$

Proof is as follows:

The residual r is defined as follow: $r := b - Ax^* \perp R(A) = \{Ay : y \in \mathbb{R}^n\}$
 \Rightarrow by the definition of Orthogonality we have:

$$(Ay)^T(b - Ax^*) = 0 \quad \forall y \in \mathbb{R}^n$$

$$\Rightarrow A^T(b - Ax^*) = 0$$

The system matrix of a NEQ is symmetric

Kernel and range of $A^T A$ For $A \in \mathbb{R}^{m,n}, m \geq n$ holds:

$$N(A^T A) = N(A)$$

$$R(A^T A) = R(A^T)$$

Uniqueness of least squares solutions: If $m \geq n$ and $N(A) = \{0\}$ then the linear system of equations

$Ax = b, A \in \mathbb{R}^{m,n}, b \in \mathbb{R}$. has a unique least squares solution

$$x = (A^T A)^{-1} A^T b$$

That can be obtained by solving the normal equations. Two things follow from the above Corollary:

- $N(A) = \{0\} \Rightarrow N(A^T A) = \{0\} \Rightarrow A^T A$ is regular
- $N(A) = \{0\} \Rightarrow A^T A$ is symmetric positive definite

We know x is a LSQ solution if A fulfills the Full Rank Condition (FRC):

$$N(A) = \{0\} \iff \text{rank}(A) = n$$

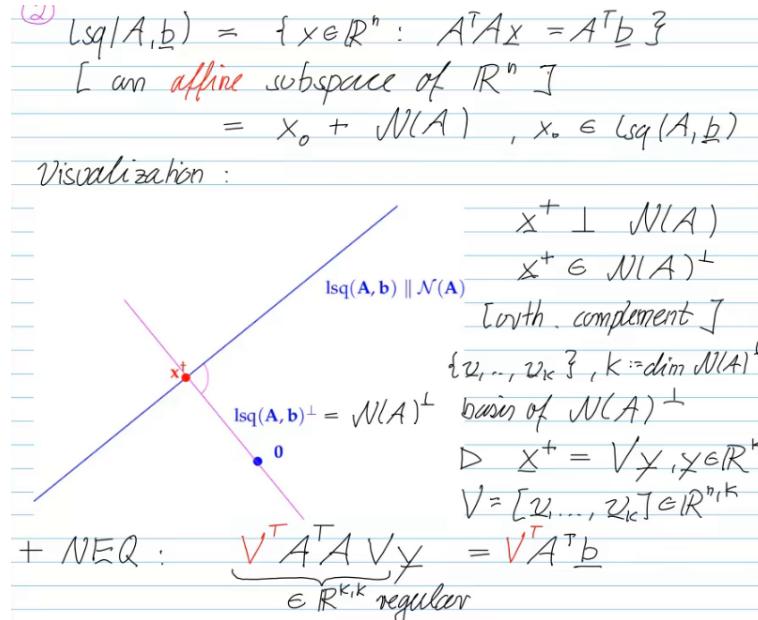
A failure of FRC indicates to:

- Flawed mathematical model
- Useless data

3.2.3 Moore-Penrose Pseudoinverse

Generalized solution of a linear system of equations: The generalized solution $x^\dagger \in \mathbb{R}^n$ of a linear system of equations $Ax = b, A \in \mathbb{R}^{m,n}, b \in \mathbb{R}^m$ is defined as:

$$x^\dagger := \underset{x \in \text{lsq}(A, b)}{\text{argmin}} \{ \|x\|_2 : x \in \text{lsq}(A, b) \}$$



T3.1.3.6 Formula for generalized solution: Given $A \in \mathbb{R}^{m,n}, b \in \mathbb{R}^m$ the generalized solution x^\dagger of the linear system of equations $Ax = b$ is given by:

$$x^\dagger = V(V^T A^T A V)^{-1}(V^T A^T b)$$

where V is any matrix whose columns form a basis of $N(A)^\perp$. The Matrix multiplication $V(V^T A^T A V)^{-1}(V^T A^T)$ is called the Moore-Penrose pseudoinverse A^\dagger and is independent of the choice of V .

3.3 Normal Equation Methods

Symmetric positive definite matrices: $M \in \mathbb{K}^{n,n}$ is symmetric (Hermitian) positive definite (s.p.d) if:

$$M = M^H \text{ and } \forall k \in \mathbb{K}^n : x^H M x > 0 \iff x \neq 0$$

If $x^H M x \geq 0 \quad \forall x \in \mathbb{K}^n$ then M is called **positive semi-definite**

Algorithm1: Normal equation method to solve full-rank least squares problem $Ax = b$

1. Compute regular matrix $C := A^T A \in \mathbb{R}^{n,n}$
2. Compute right hand side vector $c := A^T b$
3. Solve s.p.d linear system of equations: $Cx = C$

C++ code 3.2.0.1: Solving a linear least squares problem via normal equations

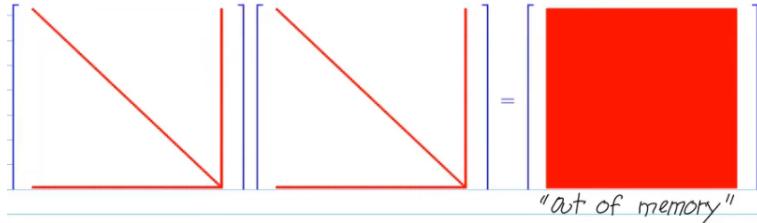
```

2  ///! Solving the overdetermined linear system of equations
3  ///!  $\mathbf{Ax} = \mathbf{b}$  by solving normal equations (3.1.2.2)
4  ///! The least squares solution is returned by value
5  VectorXd normeqsolve(const MatrixXd &A, const VectorXd &b) {
6      if (b.size() != A.rows()) throw runtime_error("Dimension mismatch");
7      // Use Cholesky factorization for s.p.d. system matrix, § 2.8.0.13
8      VectorXd x = (A.transpose() * A).llt().solve(A.transpose() * b);
9      return x;
10 }    ①  $O(mn^2)$  ③  $O(n)$  ②  $O(mn)$ 

```

Special LU-type factorization for s.p.d. Matrices : $O(n^3)$

Loss of sparsity: Only a concern for large m, n . A matrix A being sparse does not imply that $A^T A$ will also be sparse.
e.g.



We can avoid computing $A^T A$ using the Extended normal equations: Idea is to use the residual vector r as new unknown and split the equations

$$A^T \underline{A}x = A^T \underline{b} \iff A^T(\underline{b} - A\underline{x}) = 0$$

this gives us the following split:

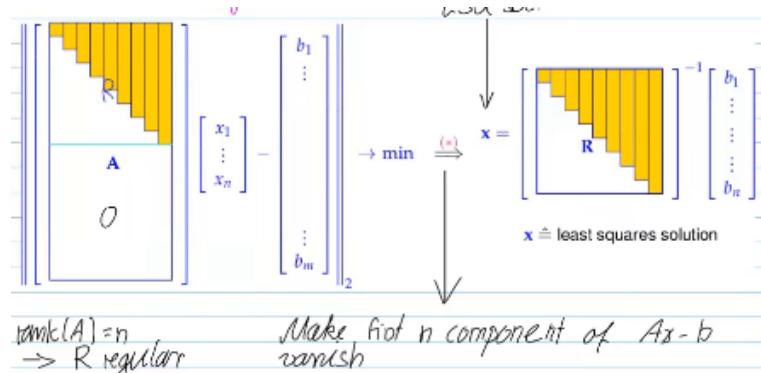
$$\begin{aligned} r &= \underline{b} - A\underline{x} \\ A^T r &= 0 \end{aligned} \Rightarrow \underbrace{\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix}}_{\in \mathbb{R}^{m+n, m+n}, \text{but sparse, if } A \text{ sparse}} \begin{bmatrix} \underline{x} \\ r \end{bmatrix} = \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix}$$

3.4 Orthogonal Transformation Methods

Like in the GE we want to transform $A\underline{x} = \underline{b} \rightarrow \tilde{A}\underline{x} = \tilde{\underline{b}}$ such that

1. LSQ solution of $\tilde{A}\underline{x} = \tilde{\underline{b}}$ is "easy to compute"
2. $lsq(A, \underline{b}) = lsq(\tilde{A}, \tilde{\underline{b}})$

Regarding 1: An easy to compute system for GE is a triangular system, we can use the same approach for this purpose



Regarding 2: We want a linear transformation which preserves the 2-norm

Idea: If we have a (transformation) matrix $\mathbf{T} \in \mathbb{R}^{m,m}$ satisfying

$$\|\mathbf{T}\mathbf{y}\|_2 = \|\mathbf{y}\|_2 \quad \forall \mathbf{y} \in \mathbb{R}^m, \quad (3.3.1.3)$$

$$\text{then } \underset{\mathbf{y} \in \mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 = \underset{\mathbf{y} \in \mathbb{R}^n}{\operatorname{argmin}} \|\tilde{\mathbf{A}}\mathbf{y} - \tilde{\mathbf{b}}\|_2, = \text{Lsq}(\mathbf{A}, \mathbf{b})$$

where $\tilde{\mathbf{A}} = \mathbf{T}\mathbf{A}$ and $\tilde{\mathbf{b}} = \mathbf{T}\mathbf{b}$.

3.4.1 Orthogonal Matrices

- $\mathbf{Q} \in \mathbb{K}^{n,n}, n \in \mathbb{N}$ is unitary if $\mathbf{Q}^{-1} = \mathbf{Q}^H$
- $\mathbf{Q} \in \mathbb{R}^{n,n}, n \in \mathbb{N}$ is orthogonal if $\mathbf{Q}^{-1} = \mathbf{Q}^T \iff \|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \forall \mathbf{x} \in \mathbb{R}^n$

The columns and rows form an Orthonormal base of \mathbb{R}^n

3.4.2 QR-Decomposition

Our goal here is to answer whether and how it is possible to find orthogonal transformations that convert any given matrix $\mathbf{A} \in \mathbb{R}^{m,n}, m \geq n, \text{rank}(\mathbf{A}) = n$, to upper triangular form, as required for the application of the "equivalence transformation idea" to full-rank linear least squares problems.

Gram-Schmidt orthogonalisation:

Input: $\{\mathbf{a}^1, \dots, \mathbf{a}^n\} \subset \mathbb{K}^m$
Output: $\{\mathbf{q}^1, \dots, \mathbf{q}^n\}$ (assuming no premature termination!)

```

1:  $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|_2}; \quad \text{\% 1st output vector}$ 
2: for  $j = 2, \dots, n$  do
   { % Orthogonal projection
3:    $\mathbf{q}^j := \mathbf{a}^j;$ 
4:   for  $\ell = 1, 2, \dots, j-1$  do (GS)
5:     {  $\mathbf{q}^j \leftarrow \mathbf{q}^j - \langle \mathbf{a}^\ell, \mathbf{q}^\ell \rangle \mathbf{q}^\ell;$  }
6:   if ( $\mathbf{q}^j = 0$ ) then STOP
7:   else {  $\mathbf{q}^j \leftarrow \frac{\mathbf{q}^j}{\|\mathbf{q}^j\|_2};$  }
8:   }

```

Theorem 3.3.3.2. Span property of G.S. vectors

If $\{\mathbf{a}^1, \dots, \mathbf{a}^n\} \subset \mathbb{R}^m$ is linearly independent, then Algorithm (GS) computes orthonormal vectors $\mathbf{q}^1, \dots, \mathbf{q}^n \in \mathbb{R}^m$ satisfying

$$\text{Span}\{\mathbf{q}^1, \dots, \mathbf{q}^n\} = \text{Span}\{\mathbf{a}^1, \dots, \mathbf{a}^n\}, \quad (1.5.1.2)$$

for all $\ell \in \{1, \dots, n\}$.

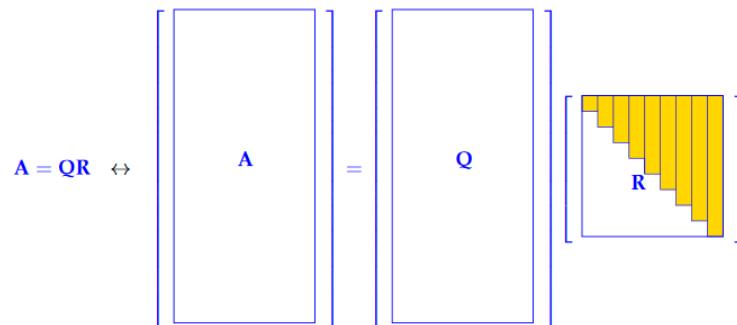
From the span property in the above image it follows that:

$$\exists \mathbf{T} \in \mathbb{R}^{n,n} \text{ Upper Triangular : } \mathbf{Q} = \mathbf{AT}$$

The inverse of a regular upper triangular matrix is also upper triangular. Hence we have found an upper triangular matrix \mathbf{R} defined as:

$$\mathbf{R} := \mathbf{T}^{-1} \in \mathbb{R}^{n,n}$$

Such that $\mathbf{A} = \mathbf{QR}$.



We then add $m-n$ zero rows to the bottom of \mathbf{R} and complement columns of \mathbf{Q} to form an orthonormal basis of \mathbb{R}^m , which yields an orthogonal matrix $\tilde{\mathbf{Q}} \in \mathbb{R}^{m,m}$

$$\begin{array}{c}
 \blacktriangleright \quad \mathbf{A} = \tilde{\mathbf{Q}} \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \\
 \Downarrow \\
 \left[\begin{array}{|c|} \hline \mathbf{A} \\ \hline \end{array} \right] = \left[\begin{array}{|c|} \hline \tilde{\mathbf{Q}} \\ \hline \end{array} \right] \left[\begin{array}{|c|} \hline \mathbf{R} \\ \hline 0 \\ \hline \end{array} \right] \\
 \Leftrightarrow \quad \tilde{\mathbf{Q}}^T \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix}.
 \end{array}$$

From this follows:

Theorem 3.3.3.4. QR-decomposition → [NS02, Satz 5.2], [Gut07, Sect. 7.3]

For any matrix $\mathbf{A} \in \mathbb{K}^{n,k}$ with $\text{rank}(\mathbf{A}) = k$ there exists

- (i) a unique Matrix $\mathbf{Q}_0 \in \mathbb{R}^{n,k}$ that satisfies $\mathbf{Q}_0^H \mathbf{Q}_0 = \mathbf{I}_k$, and a unique **upper triangular Matrix** $\mathbf{R}_0 \in \mathbb{K}^{k,k}$ with $(\mathbf{R}_0)_{i,i} > 0$, $i \in \{1, \dots, k\}$, such that

$$\mathbf{A} = \mathbf{Q}_0 \cdot \mathbf{R}_0 \quad (\text{"economical" QR-decomposition}),$$

- (ii) a **unitary Matrix** $\mathbf{Q} \in \mathbb{K}^{n,n}$ and a unique **upper triangular** $\mathbf{R} \in \mathbb{K}^{n,k}$ with $(\mathbf{R})_{i,i} > 0$, $i \in \{1, \dots, n\}$, such that

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R} \quad (\text{full QR-decomposition}).$$

If $\mathbb{K} = \mathbb{R}$ all matrices will be real and \mathbf{Q} is then orthogonal.

The economical QR-factorization of $A \in \mathbb{K}^{m,n}$, $m \geq n$ with $\text{Rank}(A) = n$ is unique if we demand $(R_0)_{ii} > 0$, $i = 1, \dots, n$

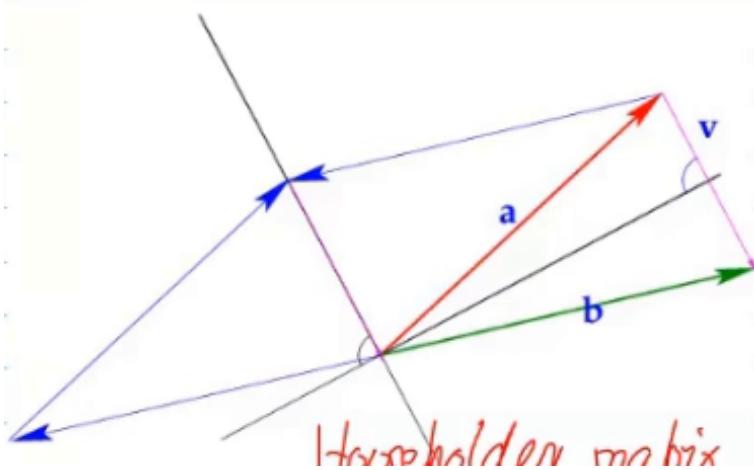
3.4.3 Computation of QR Decompositions

Gram-Schmidt is unstable i.e affected by roundoff errors, hence we need better ways to compute the QR Decomposition.

Idea: Similar to GE we want to find simple unitary/orthogonal (row) transformations rendering certain matrix elements to zero

Corollary 3.3.3.9 The product of two orthogonal/unitary matrices of the same size is again orthogonal/unitary.

Householder Reflections: A Householder transformation is a linear transformation that describes a reflection about a plane or hyperplane containing the origin. In our case we are using the Householder matrix as orthogonal transformations which step by step convert a matrix into upper triangular form.



The householder matrix is given by:

$$H(v) = (I - 2 \frac{vv^T}{v^Tv})$$

$H(v)$ is orthogonal and symmetric. Special for this case is:

$$\underline{v} = \underline{a} + \text{sgn}(a_1) \|a\|_2 e_1$$

We get the following outline for the QR-decomposition using householder matrices:

$Q_{n-1} Q_{n-2} \dots Q_1 A = R$ We know the invers of Q_i is $Q_i^H \Rightarrow$ we define $Q := Q_1^H \dots Q_{n-1}^H$ which is orthogonal (C 3.3.3.9) $\Rightarrow A = QR$ HH-reflections can be applied to any matrices. In practice we do not store the Q matrices, but the v vector, because $H(v)$ can be computed from v alone this allows us to be more efficient.

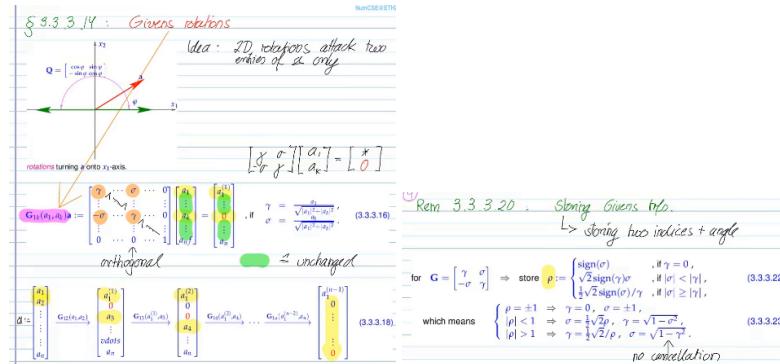
The computational cost of QR decomposition using Householder matrices is given by:

$$\text{cost}(R\text{-factor of } A \text{ by Householder transformation}) = \sum_{k=1}^{n-1} 2(m-k+1)(n-k) = \mathcal{O}(mn^2)$$

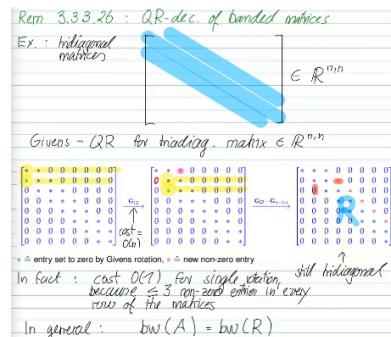
Householder QR-decomposition in EIGEN:

```
C++-code 3.3.3.34: QR-decompositions in EIGEN
1 // Include <Eigen/QR>
2
3 // Computation of full QR-decomposition (3.3.1),
4 // dense matrices built for both QR-factors (expensive!)
5 // std::pair<MatrixX<double>, MatrixX<double>> qr::decompose_full(const MatrixX<double> A)
6 Eigen::HouseholderQR<MatrixX<double> > qr(A);
7 MatrixX<double> Q = qr.householderQ(); // multiply HMMs
8 MatrixX<double> R = qr.matrixR(); template<typename T> triangularView<Eigen::Upper> R_t() { return std::pair<MatrixX<double>, MatrixX<double>>(Q,R); }
9
10 // Computation of economical QR-decomposition (3.3.1),
11 // dense matrix built for Q-factor (possibly expensive)
12 std::pair<MatrixX<double>, MatrixX<double>> qr::decompose_econ(const MatrixX<double> A)
13 using index_t = MatrixX<double>::Index;
14 const index_t m = A.rows(); const index_t n = A.cols();
15 Eigen::HouseholderQR<MatrixX<double> > qr(A);
16 MatrixX<double> Q = qr.householderQ(); // m x m identity(m,m);
17 MatrixX<double> R = qr.matrixR(); template<typename T> triangularView<Eigen::Upper> R_t() { return std::pair<MatrixX<double>, MatrixX<double>>(Q,R); }
18
19
20 // Computed QR-dec.: HMMs stored in the lower triangular part
21
22 Cost = O(mn2)
23 for m,n → ∞
24 contains 2x pent →
25 for HMMs H(2)
```

Givens Rotations



The advantage of Givens Rotations over Householder reflections is that with Givens we can specifically choose which entries we want to set to zero. QR-dec of banded matrices:



Chapter 4

C++

4.1 Templates and Template Classes in C++

Templates: Let you define the behavior of the class without actually knowing what datatype will be handled by the operations of the class i,e a templated class does not depend on the datatype it deals with. The basic syntax for declaring a templated class:

```
1 | template <class a_type> class a_class {...};
```

a_type is not a keyword, its an identifier that during the execution of the program will represent a single datatype

When defining a function as a member of a templated class, it is necessary to define it as a templated function

```
1 | template<class a_type> void a_class<a_type>::a_function() {...}
```

specialization: An instantiated object of a templated class

Example:

```
1 | class calc
2 | {
3 |     public:
4 |         int multiply(int x, int y);
5 |         int add(int x, int y);
6 |     };
7 |     int calc::multiply(int x, int y)
8 |     {
9 |         return x*y;
10 |    }
11 |    int calc::add(int x, int y)
12 |    {
13 |        return x+y;
14 |    }
1 | template <class A_Type> class calc
2 | {
3 |     public:
4 |         A_Type multiply(A_Type x, A_Type y);
5 |         A_Type add(A_Type x, A_Type y);
6 |     };
7 |     template <class A_Type> A_Type calc<A_Type>::multiply(A_Type x,A_Type y)
8 |     {
9 |         return x*y;
10 |    }
11 |    template <class A_Type> A_Type calc<A_Type>::add(A_Type x, A_Type y)
12 |    {
13 |        return x+y;
14 |    }

```

Keyword: typename In a template declaration "typename" can be used as an alternative to class to declare type template parameters

4.2 Scope Resolution Operator ":"

The scope Resolution Operator is used for one of the following:

- Access the global variable, if there is a local variable with the same name
- Define a function outside the class
- Access a class's static variable
- Multiple Inheritance
- Refer to a class inside another class

4.3 Namespaces

Namespaces provide a method for preventing name conflicts in large projects. Symbols declared inside a namespace block are placed in a named scope that prevents them from being mistaken for identically-named symbols in other scopes

using-declaration:

```
using ns_name::name
```

Makes the symbol name from the namespace *ns_name* accessible for unqualified lookup as if declared in the same class scope, block scope or namespace as where this using-declaration appears

unqualified lookup

4.4 Name Lookup

The procedure by which a name, when encountered in a program, is associated with the declaration that introduced

Unqualified name lookup An unqualified name, is a name that does not appear to the right of a scope resolution operator. Name lookup examines the scopes as described below, until it finds at least one declaration of any kind at which time the lookup stops and no further scopes are examined.