

A

Kurven: \rightarrow stetige Abbildung $\gamma: [a, b] \rightarrow \mathbb{R}^n$.

• Weg: $\hat{=} \text{ Bild } \Gamma = \gamma([a, b])$ $\xrightarrow{\text{L}} \Gamma$ [$\gamma = \text{Parametrisierung}$ von Γ]

• Einfachheit: Falls γ injektiv ("doppel punktfrei") $\xrightarrow{\text{L}} \gamma$ $\xrightarrow{\text{nicht einfach}} \gamma$ ✓

• Geschlossenheit: Falls $\gamma(a) = \gamma(b)$ $\xrightarrow{\text{L}} \gamma$ ✓

• C^1 -Kurve: Falls γ stetig diff'bar $\xrightarrow{\text{L}} \gamma$ ✓

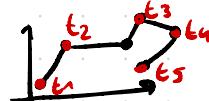
• Tangentenvektor im Pkt. $\gamma(t)$: $\gamma'(t) = (\gamma_1'(t), \dots, \gamma_n'(t))^T$

• Glättigkeit: Falls γ eine C^1 -Kurve & $\|\gamma'(t)\| \neq 0 \forall t \in [a, b]$

• Tangentialerheitsvektor für glatte Kurven: $T: [a, b] \rightarrow \mathbb{R}^n$: $T(t) := \frac{\gamma'(t)}{\|\gamma'(t)\|} \Rightarrow \|T(t)\| = 1$ ✓

• Umparametrisierung: $\tilde{\gamma} := \gamma \circ \phi: [c, d] \rightarrow [a, b] \rightarrow \mathbb{R}^n$, $t \mapsto \gamma(\phi(t))$ $\xrightarrow{\text{Skalar!}}$ geht nur für glatte Kurven!

• Stückweise Eigenschaften:



\rightarrow z.B. Polygonzug stückweise glatt

• Bogenlänge für C^1 -Kurven:

$$\rightarrow L(\gamma) = \int_a^b \|\gamma'(t)\| dt$$



"Quasi Maßband an Kurve anlegen"

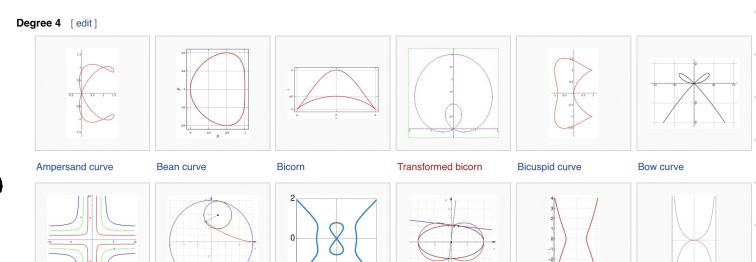
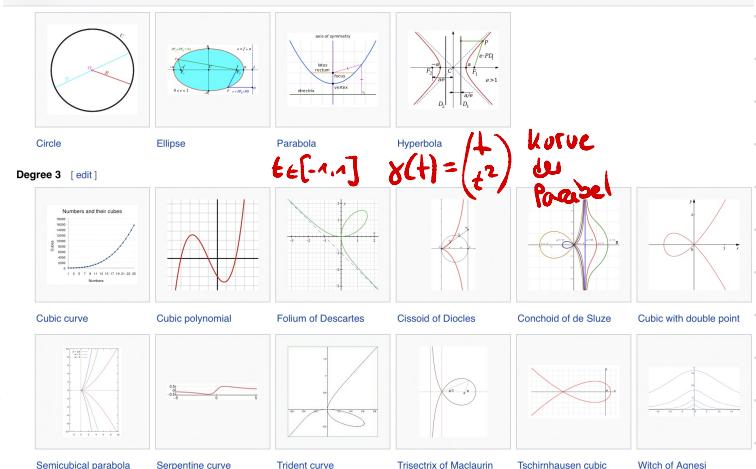
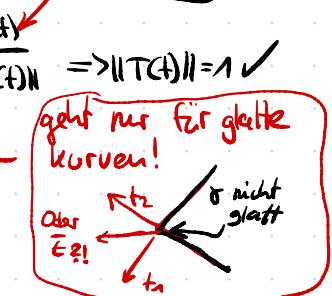
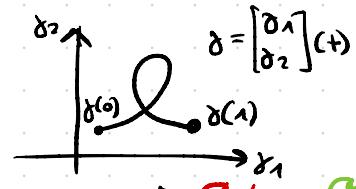
• Invarianz Bogenlänge unter Umparametrisierung:

Sei $\phi: [c, d] \rightarrow [a, b]$ Diffeomorphismus und $\tilde{\gamma} := \gamma \circ \phi$, dann

$$L(\tilde{\gamma}) = \int_c^d \|\tilde{\gamma}'(s)\| ds = \int_a^b \|\gamma'(\phi(s))\| |\phi'(s)| ds = \int_a^b \|\gamma'\| dt = L(\gamma)$$

$\|\gamma\| = 1$ wäre schön.... 😊

weil sub
 $t = \phi(s)$
 $\phi'(s) ds = dt$



Natürliche Parametrisierung: $\gamma: [0, L] \rightarrow \mathbb{R}^n$ ist nat. Param. eines Weges Γ falls $\|\gamma'(t)\|_2 = 1$. [Einheitliches Ablesen der Bogenlänge] $\gamma(t)$

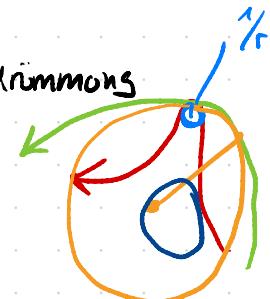
→ Konstruktion: Benutze Diffeomorphismus $\psi: [a, b] \rightarrow [0, L]$ mit $t \mapsto \psi(t) := \int_a^t \|\gamma'(s)\| ds$ und nach t umformen und einsetzen in γ

Normeneinheitsvektor: Sei $T(t) := \frac{\gamma'(t)}{\|\gamma'(t)\|}$ Tangentenvektor, dann $N(t) := \frac{T'(t)}{\|T'(t)\|}$ Normenvektor.

Krümmung: Sei γ eine natürliche Param., dann ist die Krümmung von γ in γ :

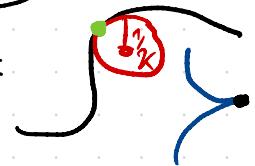
$$\kappa(\gamma) := \|T'(\gamma)\|$$

Kreiskrümmung: $\gamma: [0, 2\pi] \rightarrow \mathbb{R}$, $s \mapsto \gamma(s) = \begin{bmatrix} r \cos(\frac{s}{r}) \\ r \sin(\frac{s}{r}) \end{bmatrix}$



$$\gamma'(s) = \begin{bmatrix} -\sin(\frac{s}{r}) \\ \cos(\frac{s}{r}) \end{bmatrix} \Rightarrow T(t) = \frac{\gamma'(t)}{\|\gamma'(t)\|} = \gamma'(t) \text{ weil } \|\gamma'(t)\|_2 = \sqrt{\sin^2 + \cos^2} = 1$$

$$\Rightarrow T'(\gamma) = \begin{bmatrix} -\frac{1}{r} \cos(\frac{s}{r}) \\ -\frac{1}{r} \sin(\frac{s}{r}) \end{bmatrix} \Rightarrow \kappa(\gamma) = \sqrt{\frac{1}{r^2} (\sin^2 + \cos^2)} = \frac{1}{r}$$



Beispiel: a) Check: Bdg. muss erst parametrisiert werden!

$$x^{2/3} + y^{2/3} = a^{2/3} \text{ mit } \gamma: [0, 2\pi] \rightarrow \mathbb{R}^2$$

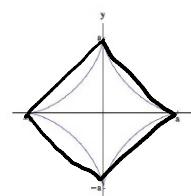
$$[a \cos^3(t)]^{2/3} + [a \sin^3(t)]^{2/3} = a^{2/3}$$

$$\Leftrightarrow a^{2/3} \cos^2(t) + a^{2/3} \sin^2(t) = a^{2/3} \quad \checkmark$$

b) Bogenlänge: $L(\gamma) = \int_0^{2\pi} \|\gamma'(t)\|_2 dt$

$$\text{Daten } \gamma'(t) = \begin{bmatrix} \frac{d}{dt}(a \cos^3(t)) \\ \frac{d}{dt}(a \sin^3(t)) \end{bmatrix} = \begin{bmatrix} -3a \cos^2(t) \sin(t) \\ 3a \sin^2(t) \cos(t) \end{bmatrix}$$

Aufgabe 68. (Bogenlänge einer Kurve)



The so-called astroid (Sternkurve) is described by the Cartesian equation

$$x^{2/3} + y^{2/3} = a^{2/3}$$

for some $a > 0$.

$$\gamma(t) = \begin{bmatrix} a \cos^3(t) \\ a \sin^3(t) \end{bmatrix}$$

a) Verify that the astroid can be expressed in parametric form $\gamma: [0, 2\pi] \rightarrow \mathbb{R}^2$ for

$$x(t) = a \cos^3 t \quad ; \quad y(t) = a \sin^3 t$$

b) Find the length of the astroid.

c) Find the area of the astroid using the change of variables $F: (x, y) \mapsto (r, \varphi)$ defined as

$$x = r \cos^3 \varphi, \quad y = r \sin^3 \varphi$$

Begin by verifying that

$$J(r, \varphi) = \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \varphi} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \varphi} \end{vmatrix} = \frac{8}{3} r (1 - \cos 4\varphi)$$

and notice that we can describe the astroid in terms of the coordinates (r, φ) as $0 \leq r \leq a$ and $0 \leq \varphi \leq 2\pi$.

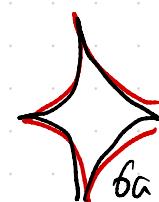
$$\Rightarrow \|g'(t)\|_2 \stackrel{a>0}{\geq} 3a \sqrt{(\cos^2 \sin)^2 + (\sin^2 \cos)^2} = 3a \sqrt{(\cos^2 + \sin^2)(\sin^2 \cos^2)} = 3a \sqrt{\cos^2 + \sin^2}$$

(3)

$$\Rightarrow L(g) = \int_0^{2\pi} \|g'(t)\| dt = 3 \int_0^{2\pi} \sqrt{\sin^2(t) \cos^2(t)} dt$$

$$= 3a \int_0^{2\pi} \left(\underbrace{\sin(t) \cos(t)}_{{= \frac{1}{2} \sin(2t)}} \right)^2 dt = 3a \int_0^{2\pi} \left| \frac{1}{2} \sin(2t) \right| dt$$

(Formelzettel)

$$= 12a \int_0^{\pi/2} \frac{1}{2} \sin(2t) dt = 12a \cdot \frac{1}{2} = 6a$$


c) Fläche (nicht so wichtig) $A = \int_0^a \int_0^{2\pi} 1 \cdot J(r, \varphi)$

$$= \int_0^a \int_0^{2\pi} \frac{3}{8}r(1 - \cos(4\varphi)) d\varphi dr$$

$$= \dots = \frac{3}{8}\pi a^2$$

- Symbolic tools (Sympy, Julia, Matlab, Wolfram Alpha) manchmal hilfreich

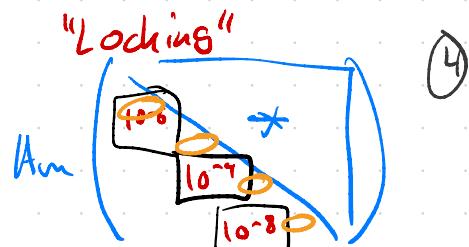
N

QR-Algorithm for Eigenvalue Problems

- Given $A \in \mathbb{R}^{n \times n}$
- Set $A_0 = A$
- for $m = 0, 1, 2, \dots$ do

$$[A] = [Q^T] [R]$$

Am (400)
10^-4
10^-8



- Compute $A_m = Q_m R_m$ with $Q_m^T = Q_{m-1}^{-1}$ ortho. & R_m upper-Δ (QR-Decomp.)
- Assign $A_{m+1} = R_m Q_m = Q_m^T A Q_m$ (similarity ratio \Leftrightarrow same Eigen)

$$\rightarrow A_m = Q_{m-n}^T A_{m-1} Q_{m-1} = Q_{m-n}^T Q_{m-2}^T A_{m-2} Q_{m-2} Q_{m-1}$$

$$= \dots = (Q^{(m)})^T A Q^{(m)} \quad \text{with } Q^{(m)} := Q_0 Q_1 \dots Q_{m-1} \text{ is ortho.}$$

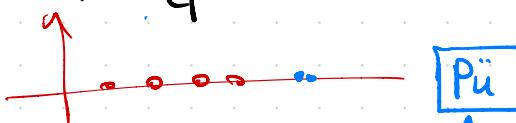
Buchi-Templates
for the Solution
of Eigenr. Probs
(HalBlaes
Cover)

Eigenvalue-Estimates: stehen auf Diagonalen von A_m (upper-Δ $\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}$)

Eigenvector-Estimates: stehen in Spalten von $Q^{(m)}$

Konvergenz: (siehe Lecture) : $A_m = \Lambda + O(\rho^m)$

$$\text{mit } \rho := \max_{i=1 \dots n} \left| \frac{\lambda_{i+1}}{\lambda_i} \right|$$



↪ Wenn zwei Elval nahe beieinander \Rightarrow nicht so gut (mit QR mit Shift)

↪ Wenn komplex-konjugierte Elvals \Rightarrow A_m konvergiert zu oberer Block A-Matrix

ZB $A_m \rightarrow \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ 0 & 0 & [*] & * \\ 0 & 0 & 0 & [*] \end{bmatrix}$ \tilde{A} hat dann Elvals λ_j & λ_{j+n}

Berechnung der QR-Zerlegungen:

Gram-Schmidt: "Wenn A vollen Rang, mache Spalten von A zu OGB $\Rightarrow Q$ "

Householder: "Bringe A auf obere Dreieckstruktur $\Rightarrow R$, $Q \stackrel{\text{def}}{=} \prod T$ rotos"

Givens-Rotationen: "Entferne alle Einträge von A unterhalb des Diag"

Implementierung: \rightarrow Problem: Jede QR-Decomp. braucht $O(n^3)$

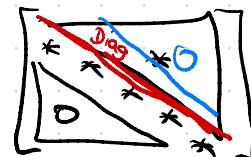
Beobachtung: QR-Decomp für Hessenberg-Matrizen braucht nur $O(n^2)$ und $\tilde{H} = RQ$ bleibt obere Hessenberg (\rightarrow siehe HA).

\Rightarrow Vorbereitung der Matrix auf obere Hessenberg Gestalt (per Ähnlichkeitstrfo)

\Rightarrow Hessenberg braucht zwar auch $O(n^3)$ immal dann aber pro Iterationsschritt nur noch $O(n^2)$ Operationen.

\Rightarrow Für symmetrische Matrizen braucht man nur $O(n)$ Givens-Rotationen mit konstantem Aufwand.

$$C A_{ij} = 0$$



(5)

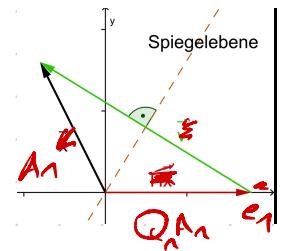
Recall: Householder Spiegelung für QR Zerlegung

→ Matrix $A = [A_1 \dots A_n]$ gegeben

→ Erste Householder Matrix $Q_1 = I - 2 \frac{v_1 v_1^T}{v_1^T v_1}$ mit $\begin{cases} \cdot v_1 = A_1 + \alpha_1 e_1 \\ \cdot \alpha_1 = \text{sgn}(A_{11}) \|A_1\|_2 \end{cases}$

→ $Q_1 A = \begin{bmatrix} * & * & * \\ 0 & * & * \\ \vdots & \ddots & * \\ 0 & * & * \end{bmatrix}$ (erste Spalte wird auf e_1 projiziert.)

→ $Q_2 Q_1 A = \begin{bmatrix} * & * & * \\ * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}$ no Nutze $A'_1 := (Q_1 A)[2:\text{end}, 2:\text{end}]$ für Konstruktion von $Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & Q_2' \end{bmatrix}$ usw...



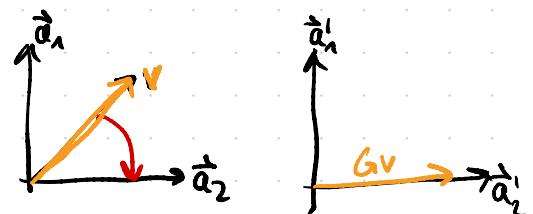
$$R = \tilde{Q} A \quad A = Q R$$

→ $R = \underbrace{Q_{n-1} \cdots Q_1}_\tilde{Q} A = \begin{bmatrix} * & * \\ 0 & * \\ \vdots & * \end{bmatrix}$ obere $n \times n$ -Matrix

→ Was ist \tilde{Q}^2 : $R = \tilde{Q} A \Leftrightarrow \tilde{Q}^{-1} R = A \Leftrightarrow \tilde{Q}^T R = A \Leftrightarrow \boxed{Q = \tilde{Q}^T}$

Recall: Givens Rotationen : Drehung in Ebene

$$G(i,j,\theta) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & c \\ 0 & -s \\ -s & c \\ 0 & 0 \end{bmatrix} \text{ mit } \begin{cases} c = \cos \theta = \frac{a_{ii}}{\rho} \\ s = \sin \theta = \frac{a_{ij}}{\rho} \\ \rho = \sqrt{a_{ii}^2 + a_{ij}^2} \end{cases} ?!$$



$$G(n,1) \cdot A = \begin{bmatrix} * & * \\ \vdots & * \\ 0 & 0 \end{bmatrix} \rightarrow \text{Kann für QR benutzt werden}$$

→ Insbesondere QR für Hessenberg Matrizen in $O(n-1)$ Givens-Rotationen [mit konst oder variablen Koeffizienten!]
Eintrag (i,j) wird zu Null

6

- QR Algorithmus
- Konvergenz
- Householder Hessenberg Trafo
- Givens QR für tridiagonale Matrizen

QR Algorithm Native

We use Julia's standard `qr()` function and implement:

- Given $A \in \mathbb{R}^{n \times n}$
 - Initialize $Q^{(m+1)} = I$
 - For $k = 1, \dots, m$:
 - Calculate QR-Decomposition: $A_k = Q_k R_k$
 - Update: $A_{k+1} = R_k Q_k$
 - Return diagonal entries of A_m and $Q^{(m)} = Q_m \cdots Q_1 Q_0$

`qra_general` (generic function with 1 method)

```

function qra_general(A, m)
    %assert size(A[1]) == size(A)[2] & length(size(A))==2
    n = size(A)[1];
    Qm = I(n);
    for k=1:m
        Qk = qr(A);
        A = R * Qk;
        Qm = Qm * Qk;
    end
    return diag(A), Qm
end

```

QR Algorithm for symmetric Hessenberg matrices

Symmetric hessenberg matrices are tridiagonal (only diag plus uppe and lower sub-diagonal). For the QR decomposition, we only have to make the lower sub diagonal entries to zero to obtain the upper right triangular matrix. This can be done by using Givens roations:

Givens Rotations [1]

Given a matrix A , we can make to entry A_{ij} to zero with $A_{\text{new}} = G(A, i, j)$ where

$$G(A, i, j) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

with

- $r = \sqrt{A_M^2 + A_{ij}^2}$
 - $s = -A_{ij}/r$
 - $c = A_{jj}/r$

[1]: https://en.wikipedia.org/wiki/Givens_rotation

```

givens.rotation_matrix (generic function with 1 method)
  • Function givens.rotation_matrix(A, i, j)
    - Input A = [A[1], ..., A[n]]^T
    - Input i = index of row i
    - Input j = index of row j
    - Output G = sparse(1,01, n, n)
      G[i,i] = c
      G[i,j] = s
      G[j,i] = -s
      G[j,j] = 1
      return G
  • end

```

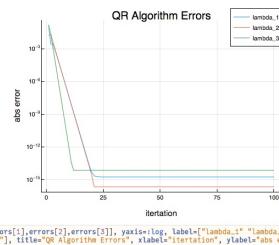
Check Convergence

```

    begin
      N = 100
      tape = Array[]
      for k in 1:N
        push!(tape, sort(qra_general(A, k)[1], rev=false)) # don't do this, very
        inefficient
      end
    end

errors =
@vars(Float64, |||) @vars(64 ||| 0.137636, 0.00293032, 0.00266808, 0.000628876, 0.00012773,
  errors = [
    abs.(map(x => x[1]), tape) .- eigen(A, values)[1],
    abs.(map(x => x[2]), tape) .- eigen(A, values)[2],
    abs.(map(x => x[3]), tape) .- eigen(A, values)[3],
  ]
  , )

```



Improve QR Algorithm with Upper Hessenberg Matrix Preconditioning

- Idea: QR decomposition needs $\mathcal{O}(n^3)$, QR for Hessenberg matrices is easier done in $\mathcal{O}(n^2)$. Linear complexity is even possible if A is symmetric. In this case, the QR decomposition only needs $\mathcal{O}(n)$ Givens rotations with constant effort.
 - Therefore transform the matrix A to upper Hessenberg form with similarity transforms in $\mathcal{O}(n^3)$ (also cubic, but only needs to be done once) and use this matrix for the QR algorithm.

Upper Hessenberg Shape

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & h_{3n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{m-1,n} & h_{mn} \end{pmatrix}$$

Algorithm [1]:

- Given: $A \in \mathbb{R}^{n \times n}$
 - For $k = 1 \dots n - 2$ do:
 - $[v, \beta] \leftarrow \text{house}(A(k+1:n, k))$
 - $A(k+1:n, k:n) \leftarrow (I - \beta vv^T)A(k+1:n, k:n)$
 - $A(1:n, k+1:n) \leftarrow A(1:n, k+1:n)(I - \beta vv^T)$

with Householder reflection vector v and weight $\beta = 2/(v^T v)$

[1]: <https://www.tu-chemnitz.de/mathematik/numa/lehre/nla-2015/Folien/nla-kapitel6.pdf>

```

upperhessenberg (generic function with 1 method)
  • function upperhessenberg(A)
    - if size(A)[1] == size(A)[2] && length(size(A))==2
      n = size(A)[1]
      for k = 1:n-2
        v, b = householderdecomp(A[k+1:n, k])
        A[k+1, k:n] = (I(n-k) - B * v'') * A[k+1:n, k:n]
        A[1:n, k+1:n] = A[1:n, k+1:n] * (I(n-k) - B * v'')
      end
    end
  end

```

Speed Check

We still loose against Julia's native `qr`-method. 😞

- Homework: Improve this

```
QRA General:  
 0.166218 seconds (38.90 k allocations: 51.228 MiB, 4.84% gc time)  
QRA Own:  
 0.417899 seconds (99.22 k allocations: 752.204 MiB, 13.56% gc time)
```

```

with_terminal() do
    M = 100
    M = 50
    C = Array(Symmetric(rand(N,N)))
    # @time qr(C)
    # @time qr_symm_hess(C)
    # @time upperhessenberg(C)

    println("QRA General:")
    @time Am1, Qm1 = qra_general(C, M)
    println("QRA Own:")
    @time Am2, Qm2 = qra_symm(C, M)
end

```