present

```
html"<button onclick='present()'>present</button>"
```

QR Algorithm for Eigenvalue Problems with Hessenberg/Givens Tricks

- Mathe 3 (CES)
- WS21
- Lambert Theisen (theisen@acom.rwth-aachen.de)


```
PlotlyBackend()
```

```
begin
    using LinearAlgebra, PlutoUI, Random, SparseArrays, Plots
    plotly()
end
```

QR Algorithm Native

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

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

```
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
```

Check Validity of Implementation

```
A = 3×3 Matrix{Float64}:
 3.0  2.0   3.0
 4.0  7.0   6.0
 7.0  8.0  11.0
```

```
A = 1. * [
    1 2 3
    4 5 6
    7 8 9
] + 2I(3)
```

```
([18.1168, 2.0, 0.883156], 3×3 Matrix{Float64}:
 0.231971  -0.408248  -0.882906
 0.525322   0.816497  -0.23952
 0.818673  -0.408248   0.403865)
```

```
gra_general(A, 100)
```

```
[-1.77636e-15, -2.22045e-16, -7.10543e-15]
```

```
eigen(A).values - sort(gra_general(A, 100)[1], rev=false)
```

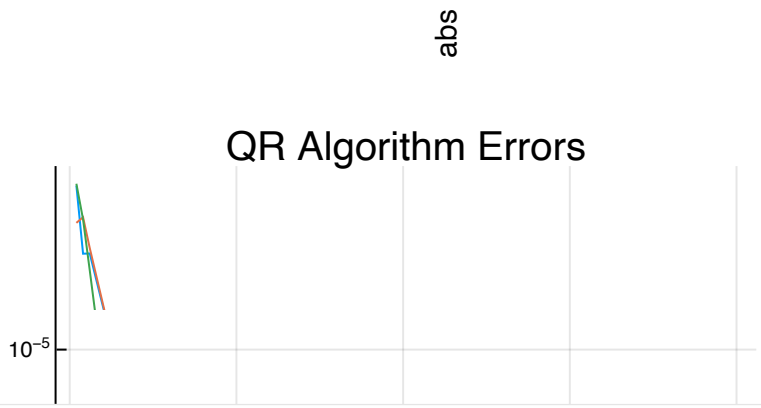
Check Convergence

```
begin
    N = 100
    tape = Array[]
    for k=1:N
        push!(tape, sort(gra_general(A, k)[1], rev=false))
        # *dont do this, very inefficient*
    end
end
```

```
errors =
```

```
[[0.137636, 0.00263023, 0.00266808, 0.000628078, 0.00012773, 2.51622e-5, 4.91885e-6, 9.5
```

```
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]),
```




```
plot([errors[1],errors[2],errors[3]], yaxis=:log, label=["lambda_1" "lambda_2"
"lambda_3"], title="QR Algorithm Errors", xlabel="itertation", ylabel="abs error")
```

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 matix 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

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

```
    n = size(A)[1]
    for k = 1:n-2
        v, β = householdervec(A[k+1:n,k])
        A[k+1:n, k:n] = (I(n-k) - β * v * v') * A[k+1:n, k:n]
        A[1:n, k+1:n] = A[1:n, k+1:n] * (I(n-k) - β * v * v')
    end
    return A
end
```

```
householdervec (generic function with 1 method)
```

```
function householdervec(x)
    @assert size(x)[1]>0 && length(size(x))==1
    n = size(x)[1]
    e1 = I(n)[:,1]
    v = x + norm(x, 2) * e1
    β = 2 / (v' * v)
    return v, β
end
```

Check if Householder ad Upperhessenberg Transformations work

```
md"""
### Check if Householder ad Upperhessenberg Transformations work
"""
```

```
BB = 3×3 Matrix{Float64}:
 1.0  2.0  3.0
 4.0  5.0  6.0
 7.0  8.0  9.0
```

```
BB = 1. * [
    1 2 3
```

matrices

Symmertric 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 $A_{\text{new}} = G(A, i, j)A$ where

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

with

```
begin
    AA =    1. * [
        1 2 3
        4 5 6
        7 8 9
    ]
    AA = givens_rotation_matrix(AA, 2, 1) * AA
end
```

Implement QR Decomposition for Symmetric Hessenberg Matrices

- Just iterate over sub-diagonal entries and make them zero to get R. Store all Givens rotations to get Q.

```
 6.0  5.0  4.44089e-16
 5.0  1.0  4.0
 0.0  4.0  3.0
```

```
qr(EE).Q * qr(EE).R
```

```
3×3 Matrix{Float64}:
 6.0          5.0  0.0
 5.0          1.0  4.0
 4.20473e-16  4.0  3.0
```

```
qr_symm_hess(EE)[1] * qr_symm_hess(EE)[2]
```

QR Algorithm for Tridiagonal Matrices

```
([9.84316, 4.02176, -3.86492], 3×3 Matrix{Float64}:
 0.746882  -0.530327  -0.401149
-0.574078  -0.209823  -0.79146
 0.335563   0.821418  -0.461162)
```

```
gra_general(DD, 1000)
```