present

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

(Shifted) (Inverse) Power Method

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

```julia
using LinearAlgebra , PlutoUI , Plots , Printf
```

```
PlotlyBackend()
```

```julia
plotly()
```

Define some Matrices

```
A = 3×3 Matrix{Int64}:
   25  -89   68
  -26  148  -52
  -10   77  -29
```

```julia
A = [
    25 -89 68
    -26 148 -52
    -10 77 -29
]
```

```
B = 3×3 Matrix{Int64}:
  -139   -85  -125
   182   -64  -178
  -117  -105    79
```

```julia
B = [
    -139 -85 -125
    182 -64 -178
    -117 -105 79
]
```

Define the Rayleigh Quotient

$$\rho_A(x) := \frac{x^T A x}{x^T x}$$

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

```
function ρ(A, x) # Rayleigh quotient
    return x' * A * x / (x' * x)
end
```

Construct Error History Object

This is used to store all the errors for later plotting.

```
struct Errorhistory
    errors :: Array{Float64}
end
```

Define the Power Method Algorithm

1. Given $A \in \mathbb{R}^{n \times n}, \tau = 10^{-10}$
2. Choose start vector $x_0 \in \mathbb{R}^n \setminus \{0\}$
3. While $k < k_{\max} \wedge \text{error} > \tau$ do
   1. $$x_{k+1} = \frac{Ax_k}{||Ax_k||_2}$$
   2. $$\lambda_{k+1} = \rho_A(x_{k+1})$$
4. Return estimated eigenpair (λ_k, x_k)

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

```
function PM(A, x0; maxit = 100, tol = 1E-10) # Power Method
    x = x0
    k = 0
    residual = Inf
    λ = nothing
    eh = Errorhistory([])
    while k <= maxit && norm(residual) > tol
        x = A * x
        x = x / norm(x)
        λ = ρ(A, x)
        residual = A * x - λ * x # if λ exact => residual = zeros(n)
        push!(eh.errors, norm(residual))
        k += 1
    end
    return (λ, x, eh)
end
```

Execute PM

Will converge to largest eigenpair.

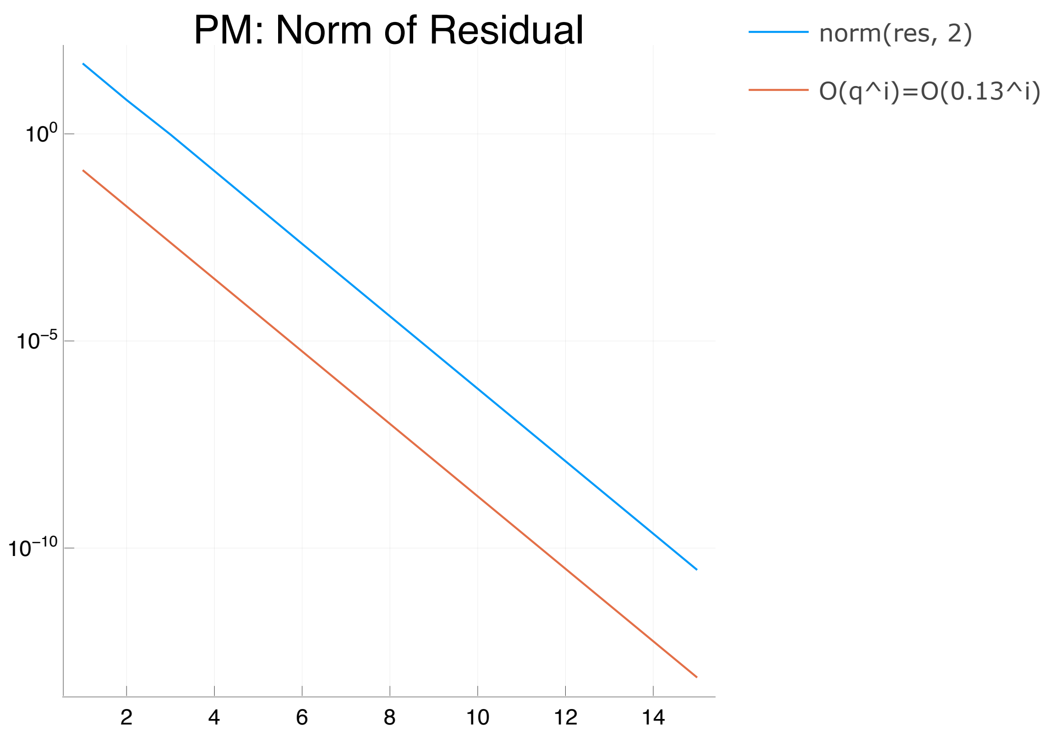
```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
  -9.0
  18.000000000000025
 135.00000000000017
vectors:
3×3 Matrix{Float64}:
 -0.894427  0.904534  -0.408248
  0.0       0.301511   0.816497
  0.447214  0.301511   0.408248
```

```
pm =
  (
      1:  135.0
      2:    Float64[
                1:  -0.408248
                2:  0.816497
                3:  0.408248
            ]
      3:    Errorhistory(Float64[50.3331, 6.59847, 0.965483, 0.123424, 0.0168063, 0.0022174
 )
```

```
pm = PM(A, ones(3))
```

Residual Error

The normed error of the residual $e = ||x_k - x^*||_2$ is in $\mathcal{O}(q^k)$ with the eigenvalue ratio $q = \lambda_1/\lambda_2$ of the considered matrix.




```
plot([pm[3].errors, [1 * abs(eigvals(A)[end-1] / eigvals(A)[end])^i for i = 1 :
size(pm[3].errors)[1]]], yaxis=:log, title="PM: Norm of Residual", label=["norm(res,
2)" "O(q^i)=O($(@sprintf("%.2f", abs(eigvals(A)[end-1] / eigvals(A)[end])))^i)"])
```

Comparison of Matrices with Different Fundamental Ratios

- Matrix A has ratio $q=$ 0.13333333333333336 = λ_2/λ_1
- Matrix B has ratio $q=$ 0.9718253158075516

Therefore, much faster converge for A.

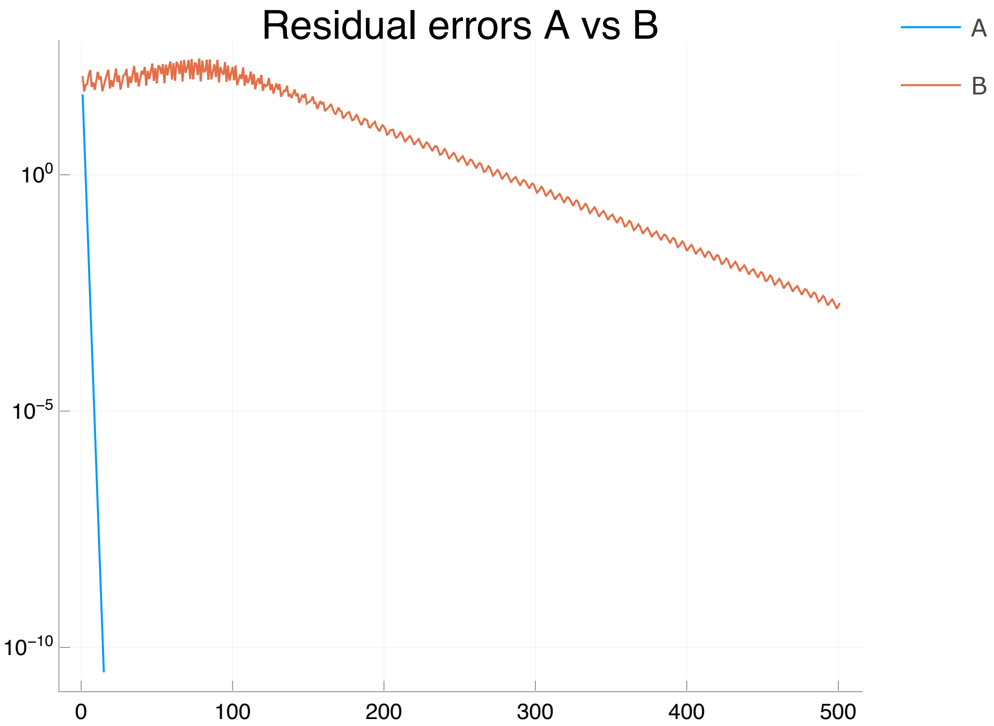
Oscillations probably due to $|\lambda_3| = |\lambda_2|$ 🤔.

```
md"""
## Comparison of Matrices with Different Fundamental Ratios

- Matrix $A$ has ratio $q=$ $(abs(eigvals(A)[end-1]/eigvals(A)[end])) = λ₂/λ₁
- Matrix $B$ has ratio $q=$ $(abs(eigvals(B)[end-1]/eigvals(B)[end]))

Therefore, much faster converge for $A$.

Oscillations probably due to $|\lambda_3| = |\lambda_2|$ 🤔.
"""
```

```
ComplexF64[
    1:  -155.0-93.0im
    2:  -155.0+93.0im
    3:  186.0+0.0im
]
```

```
eigen(B).values
```

```
3×3 Matrix{Int64}:
 -139   -85  -125
  182   -64  -178
 -117  -105    79
```




```
plot(map(X -> PM(X, ones(3), maxit=500)[3].errors, [A, B]), yaxis=:log, label=["A"
"B"], title="Residual errors A vs B")
```

Define the (Shifted) Inverse Power Method Algorithm

1. Given $A \in \mathbb{R}^n$ and eigenvalue shift μ
2. Choose start vector $x_0 \in \mathbb{R} \setminus \{0\}$
3. While $k < k_{\max} \wedge \text{error} > \text{tol}$ do
    1. Solve: $(A - \mu I)x_{k+1} = x_k$ (this is like $x_{k+1} = (A - \mu I)^{-1} x_k$)
    2. Normalize: $x_{k+1} \mapsto x_{k+1}/||x_{k+1}||_2$
    3. Update eigenvalue estimate (with initial matrix A, not A^{-1}): $\lambda_{k+1} = \rho_A(x_{k+1})$
4. Return estimated eigenpair (λ_k, x_k)

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

```
function IPM(A, x0; shift = 0, maxit = 100, tol = 1E-10) # Inverse Power Method
    x = x0
    i = 0
    residual = Inf
    λ = nothing
    eh = Errorhistory([])
    while i <= maxit && norm(residual) > tol
        x = (A - shift * I(size(A)[2])) \ x
        x = x / norm(x)
        λ = ρ(A, x)
        residual = A * x - λ * x
        push!(eh.errors, norm(residual))
        i += 1
    end
    return (λ, x, eh)
end
```

Execute IPM to find the Smallest Eigenpair

```
ipm =
  (-9.0, [-0.894427, 1.01683e-12, 0.447214], Errorhistory([8.21361, 30.0096, 5.04452, 6.23
```

```
ipm = IPM(A, ones(3))
```

Check the Error

```
6.139266872651206e-11
```

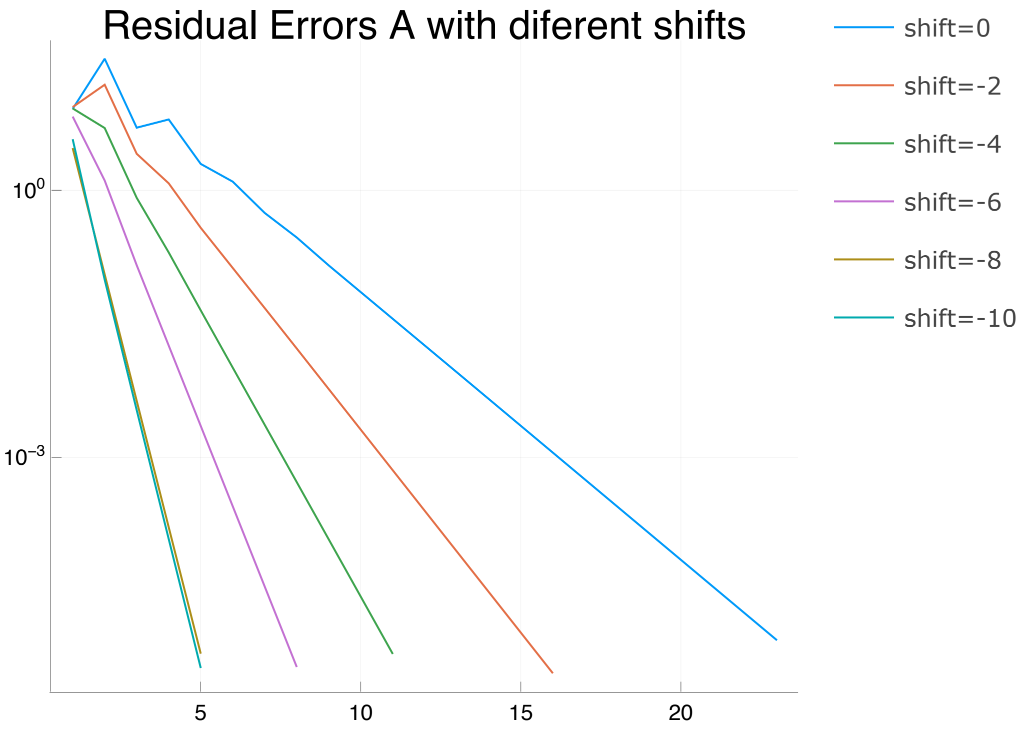
```
abs(ipm[1] - eigvals(A)[1])
```

Check the Convergence Behavior for Different Shifts

Notice that better shifts significantly improve the performance of the algorithm. Shift eight and ten are the same because they have the same absolute distance to the real eigenvalue.

```
shifts = 0:-2:-10
```

```
# Compare zeros shift with good estimation
shifts = 0:-2:-10 # real lowest eval is -9
```




```
plot(map(X -> IPM(A, ones(3), tol=1E-5, shift=X)[3].errors, shifts), yaxis=:log,
label=reshape(map(x -> string("shift=", x), shifts), 1, :), title="Residual Errors A
with diferent shifts")
```