```
PlotlyBackend()
```

```julia
begin
    using LinearAlgebra
    include("./LA.jl")
    using .LA
    using RowEchelon
    using InvertedIndices

    using Plots
    using PlutoUI
    plotly();
end
```

9

```julia
begin
    a = (1, 3, -1)
    b = (-1, 4, 2)
    dot(a, b)
end
```

9

```julia
a ⋅ b #cdot
```

When vectors point in same direction (<90deg): dot(a, b) is positive (a_proj and b are on same line in same quadrant.).

When vectors are perpendicular: dot(a,b) is zero (no a_projection)

When vectors are pointing in different directions (>90deg), dot(a,b) is negative.

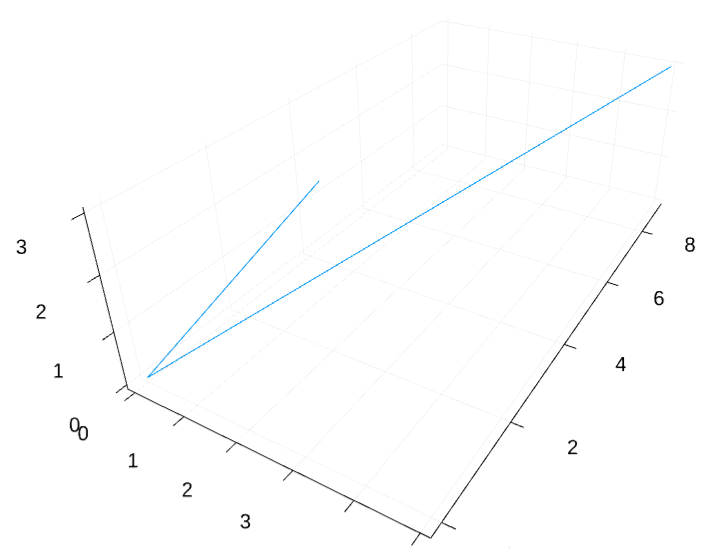
dot(a,b) = ||proj*a*on_b|| * ||b||

```julia
md"""

When vectors point in same direction (<90deg): dot(a, b) is positive (a_proj and b are on same line in same quadrant.).

When vectors are perpendicular: dot(a,b) is zero (no a_projection)

When vectors are pointing in different directions (>90deg), dot(a,b) is negative.

dot(a,b) = ||proj_a_on_b|| * ||b||

"""
```

true

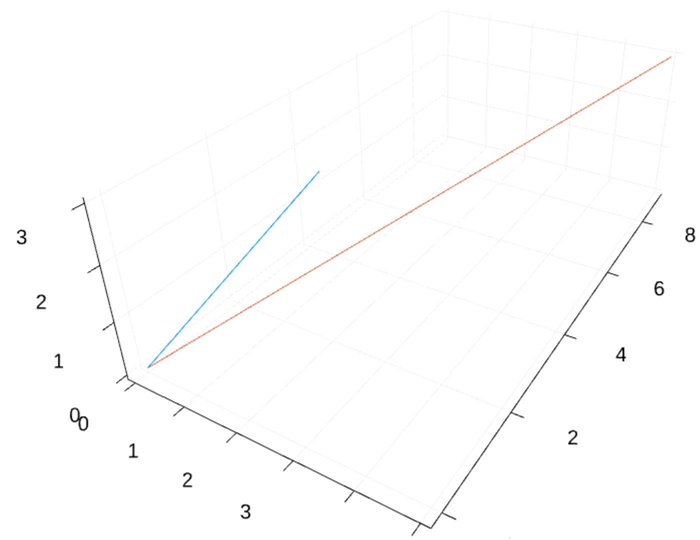
```
begin
    # Skew Symmetric
    A = [
        0 2 -1;
        -2 0 -4;
        1 4 0
    ]

    A == -transpose(A)
end
```

Norm of a Vector




```
begin
    u = [(2, 2, 3), (5, 9, 3)]
    quiver([(0, 0, 0), (0, 0, 0)], gradient=(u), arrowscale=0.3, headsize=0.2)
    # w = [(5, 9, 3)]
    # quiver!((0, 0, 0), gradient=(w), arrowscale=0.3, headsize=0.2)

end
```

```
Enter cell code...
```




```
begin
    a1 = [2 2 3]
    a2 = [5 9 3]
    LA.graph_vectors([a1, a2])
end
```

```
3.3166247903554
```

```
norm([1 -1 3]')
```

```
5.0
```

```
norm([2 1 -2 4]')
```

```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor:
2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.948683  -0.316228
 -0.316228   0.948683
R factor:
1×1 Matrix{Float64}:
 -3.1622776601683795
```

```
qr([3 1]')
```

```
5.0
```

```
begin
    C = [3 4 0]'

    norm(C)
end
```

```
F = LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
    Q factor:
    3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
     -0.6  -0.8  0.0
     -0.8   0.6  0.0
      0.0   0.0  1.0
    R factor:
    1×1 Matrix{Float64}:
     -5.0
```

```julia
F = qr(C)

```

```
3×1 Matrix{Float64}:
 3.0000000000000004
 4.0
 0.0
```

```julia
F.Q[:, 1] * F.R
```

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

```julia
function qr_unit(vector)
    Q,R = qr((vector))
    return (Q[:, 1] * R) ./ abs.(R)
end
```

```
3×1 Matrix{Float64}:
 0.6000000000000001
 0.8
 0.0
```

```julia
qr_unit(C)
```

```
D = 3×1 adjoint(::Matrix{Int64}) with eltype Int64:
     -1
     -1
     -5
```

```julia
D = [-1 -1 -5]'
```

```
3×1 Matrix{Float64}:
 -0.19245008972987535
 -0.19245008972987526
 -0.9622504486493763
```

```julia
qr_unit(D)
```

```
E = 4×1 adjoint(::Matrix{Int64}) with eltype Int64:
     -1
      4
     -2
      2
```

```julia
E = [-1 4 -2 2]'
```

```
4×1 Matrix{Float64}:
 -0.19999999999999996
  0.7999999999999999
 -0.39999999999999997
  0.39999999999999997
```

```julia
qr_unit(E)
```

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

```
function unit_vector(vector::AbstractMatrix)
    return vector/norm(vector)
end
```

```
3×1 Matrix{Float64}:
 0.6
 0.8
 0.0
```

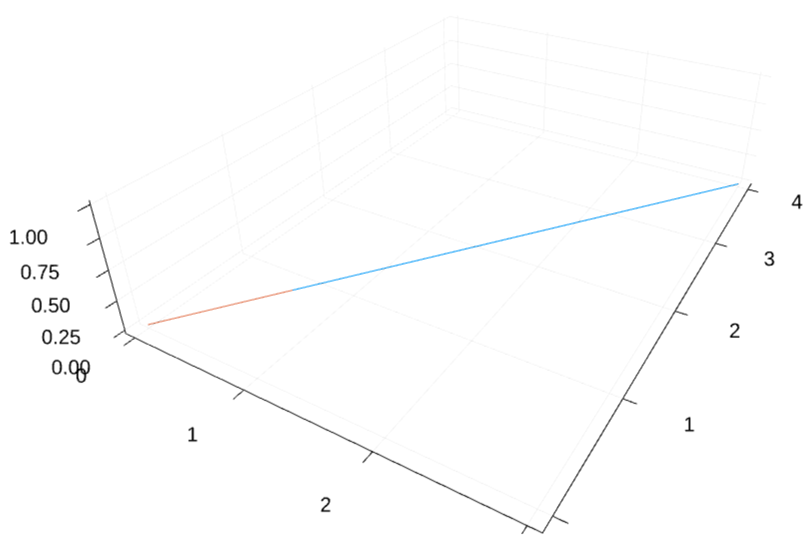
```
unit_vector(C)
```

```
3×1 Matrix{Float64}:
 -0.19245008972987526
 -0.19245008972987526
 -0.9622504486493763
```

```
unit_vector(D)
```

```
4×1 Matrix{Float64}:
 -0.2
  0.8
 -0.4
  0.4
```

```
unit_vector(E)
```




```
LA.graph_vectors([C unit_vector(C)])
```

Linear Independence

```
3×1 Matrix{Float64}:
 0.0
 0.0
 0.0
```

```
begin
    v_1 = [-1 0 2]'
    v_2 = [3 -2 2]'
    v_3 = [5 2 -6]'

    A_ = [v_1 v_2 v_3]
    b_ = [0 0 0]'
    A_ \ b_
end
```

```
A__ = 3×3 Matrix{Int64}:
       -1   3   5
        0  -2   2
        2   2  -6
```

```
# Lin indep
A__ = [ -1 3 5; 0 -2 2; 2 2 -6]
```

```
3×1 Matrix{Float64}:
 0.0
 0.0
 0.0
```

```
 A__ \ [0 0 0]'
```

```
[0.0, 0.0, 0.0]
```

```
# lin indep returns 0_n
LA.aug_solve([A_ b_])
```

```
@assert false == LA.is_LI([1 0 2; 1 3 2; 2 4 4])
```

```
@assert true == LA.is_LI([1 0 2; 2 3 2; 2 4 4])
```

```
@assert true == LA.is_LI([0 1; 1 0])
```

```
@assert false == LA.is_LI([1 2; 1 2])
```

```
@assert false == LA.is_LI([1 1; 1 1])
```

Additon

```
3×1 Matrix{Int64}:
  0
 -1
  3
```

```
[-1 -3 2]' + [1 2 1]'
```

$$\overrightarrow{a} \cdot \overrightarrow{b} = \lVert a\rVert \lVert b\rVert \cos(\theta)$$

```
md"""

$\newcommand{\norm}[1]{\lVert#1\rVert}$
$\newcommand{\abs}[1]{\lvert#1\rvert}$
$\overrightarrow{a} \cdot \overrightarrow{b} = \norm{a} \norm{b} \cos(\theta)$

"""
```

Cauchy Schwarz Inequality

$$\lvert\overrightarrow{a} \cdot \overrightarrow{b}\rvert \leq \lVert\overrightarrow{a}\rVert \cdot \lVert\overrightarrow{b}\rVert$$

```
md"""
$\abs{\overrightarrow{a} \cdot \overrightarrow{b}} \leq \norm{\overrightarrow{a}}
\cdot \norm{\overrightarrow{b}}$
"""
```

```
begin
    # Triangle Inequality
    w = [1 -1 3 4]'
    x = [2 0 3 1]'

    lhs = LA.magnitude(w + x)
    rhs = LA.magnitude(w) + LA.magnitude(x)
    println(lhs)
    println(rhs)
    @assert lhs <= rhs
end
```

```
8.426149773176359
8.937809809480573
```

```
Enter cell code...
```

```
begin
    # Cauchy Schwarz
    lhs_cs = abs(dot(w, x))
    rhs_cs = dot(LA.magnitude(w), LA.magnitude(x))
    println(lhs_cs)
    println(rhs_cs)
    @assert lhs <= rhs
end
```

```
15
19.44222209522358
```

Orthonormal

angle (generic function with 2 methods)

```
begin
    function angle(w::AbstractMatrix, v::AbstractMatrix)
        return acos((dot(w, v)/(LA.magnitude(w) * LA.magnitude(v))))
    end

    function angle(w::AbstractVector, v::AbstractVector)
        return angle(hcat(w), hcat(v))
    end
end
```

39.50971228619393

```
angle(w, x) * (180/pi)
```

90.0

```
angle([0 1]', [1 0]') * (180/pi)
```

60.00000000000001

```
angle([1; 0; 1], [1; 1; 0]) * (180/pi)
```

45.00000000000001

```
angle([2 1 2]', [1 1 0]') * (180/pi)
```

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

```julia
begin
    # Orthonormal set?
    v1 = [2//3; -2//3; 1//3]
    v2 = [1//3; 2//3; 2//3]
    v3 = [2//3; 1//3; -2//3]

    function magnitude(vector::AbstractVector)
        return magnitude(hcat(vector))
    end


end
```

```
1.0
```

```julia
LA.magnitude(hcat(v1))
```

```
1.0
```

```julia
LA.magnitude(hcat(v2))
```

```
1.0
```

```julia
LA.magnitude(hcat(v3))
```

```
0//1
```

```julia
dot(v1, v2)
```

```
0//1
```

```julia
dot(v2, v3)
```

```
0//1
```

```julia
dot(v1, v3)
```

```julia
# Yes, orthonormal
```

Solving Systems of Equations

```
3×1 Matrix{Float64}:
 11.0
 -4.0
  3.0
```

```
begin
    AA = [
        2 5 1;
        1 4 2;
        4 10 -1
    ]

    BB = [5 1 1]'

    AA \ BB
    # B / A <=> A \ B
end
```

```
[11.0, -4.0, 3.0]
```

```
LA.aug_solve([AA BB])
```

```
[1.33333, 2.0]
```

```
begin
    # Error in code: Too many rows/Inconsistent System:
    LA.aug_solve([[1 3; 2 -1; -1 4] [5 3 9]'])
end
```

Dependent system! Infinite solutions! [0 0]

1. **error**(::String) @ *error.jl:35*
2. **aug_solve**(::Matrix{Int64}, ::Float64) @ *LA.jl:56*
3. **aug_solve**(::Matrix{Int64}) @ *LA.jl:33*
4. **top-level scope** @ *Local: 4* [inlined]

```
begin
    CC = [1 2 -1; 1 0 1; 1 1 0]
    DD = [1 3 2]'
    LA.aug_solve([CC DD]) # Dependent System
end
```

Inverse

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

```
function inverse(matrix::AbstractMatrix)
    return (1/det(matrix))*adjugate(matrix)
end
```

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

```julia

function adjugate(M::AbstractMatrix)
    return transpose(LA.cofactor(M))
end
```

```
2×2 Matrix{Float64}:
 -0.0   1.0
  0.5  -0.5
```

```julia
inverse([1 2; 1 0])

```

Nonsingular - Inverse matrix Exists

Singular - Inverse Doesn't exist

```
2×2 Matrix{Float64}:
  Inf  -Inf
 -Inf   Inf
```

```julia
inverse([1 2; 1 2]) # Singular
```

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

```julia
function inv_ident(M::AbstractMatrix)
    # DOESNT WORK!!!
    rows = Integer(size(M)[1])
    return LA.aug_solve([M LA.identity(rows)])
end

```

```
[0.428571, 0.0952381, 0.047619, -0.571429, 0.047619]
```

```julia
begin
    HH = [1 1 1; -1 3 2; 2 1 1]
    inv_ident(HH)
end
```

```
4×4 Matrix{Bool}:
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
```

```julia
LA.identity(4)
```

inv_solve (generic function with 1 method)

```julia

function inv_solve(aug_M::AbstractMatrix)
    coeffs_m = aug_M[:, 1:end-1]
    consts_m = aug_M[:, end]
    _inv = LinearAlgebra.inv(coeffs_m)
    return _inv * consts_m
end
```

[-4.0, -20.0, 29.0]

```julia
begin
    FF = [
        1 1 1;
        -1 3 2;
        2 1 1;
    ]

    GG = [5 2 1]'

    inv_solve([FF GG])
end
```

[2.0, -1.0]

```julia
begin
    II = [
        3 -1;
        2 3
    ]

    JJ = [7 1]'

    inv_solve([II JJ])
end
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```