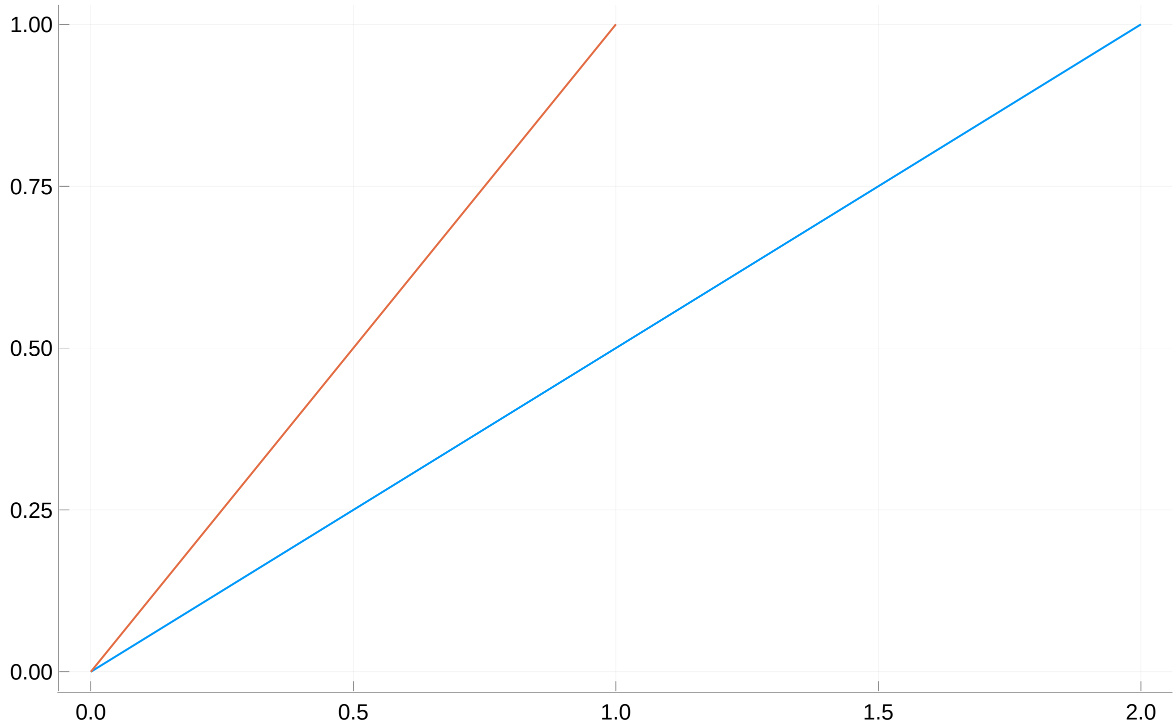
```
PlotlyBackend()
```

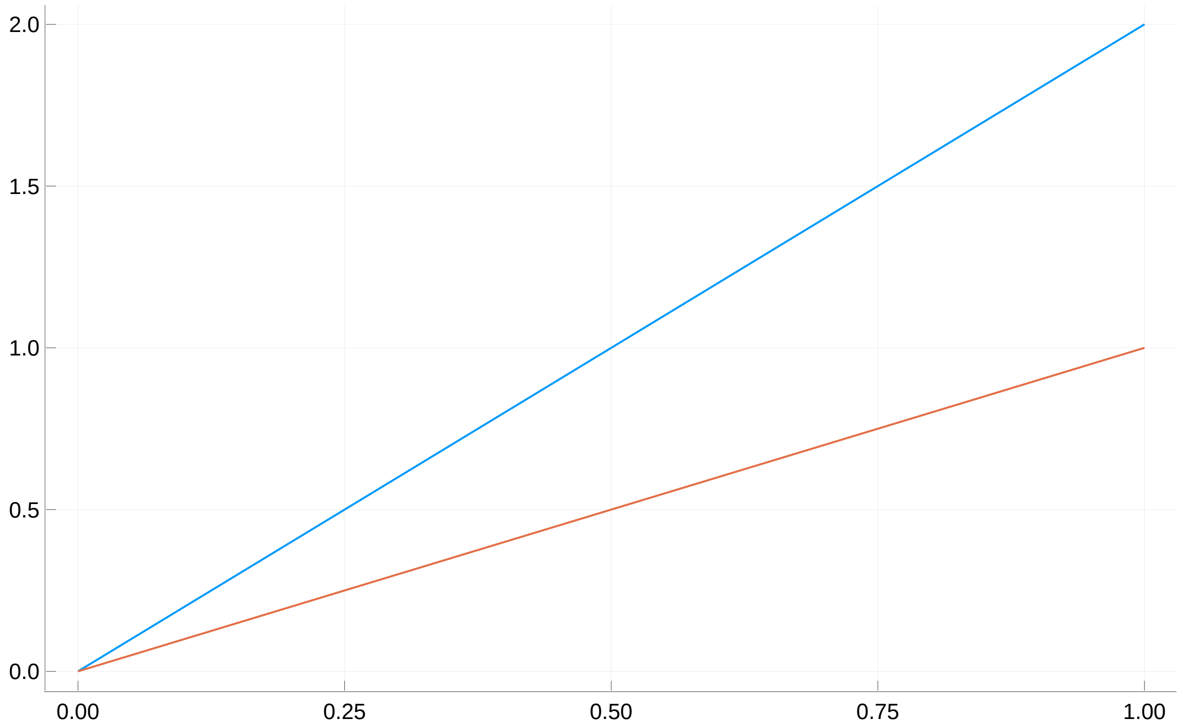
```julia
begin
    using LinearAlgebra
    include("./LA.jl")
    using .LA
    using RowEchelon
    using InvertedIndices

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

Scaling

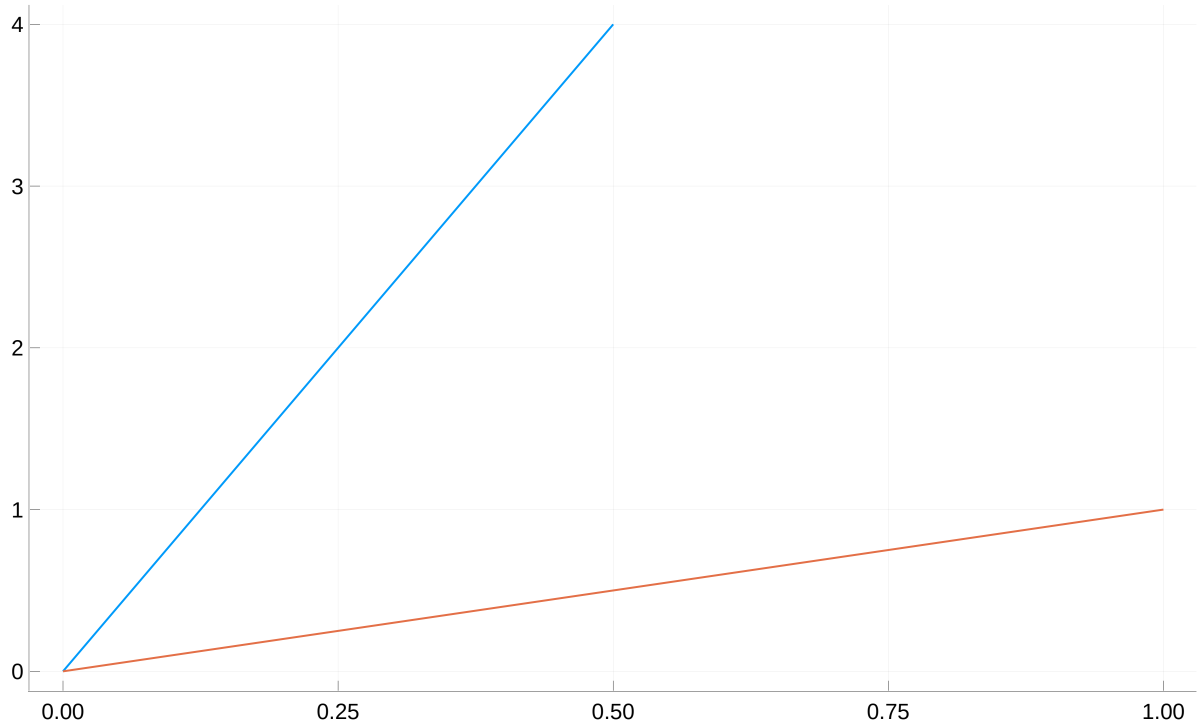
```julia
md"# Scaling"
```




```julia
begin
    k = 2 #scale
    A_x = [
        k 0;
        0 1;
    ]

    X = [
        1;
        1
    ]

    LA.graph_vectors([(A_x * X) X])
end
```




```julia
begin
    k_y = 2 #scale
    A_y = [
        1 0;
        0 k_y;
    ]

    X_y = [
        1;
        1
    ]

    LA.graph_vectors([(A_y * X_y) X_y])
end
```



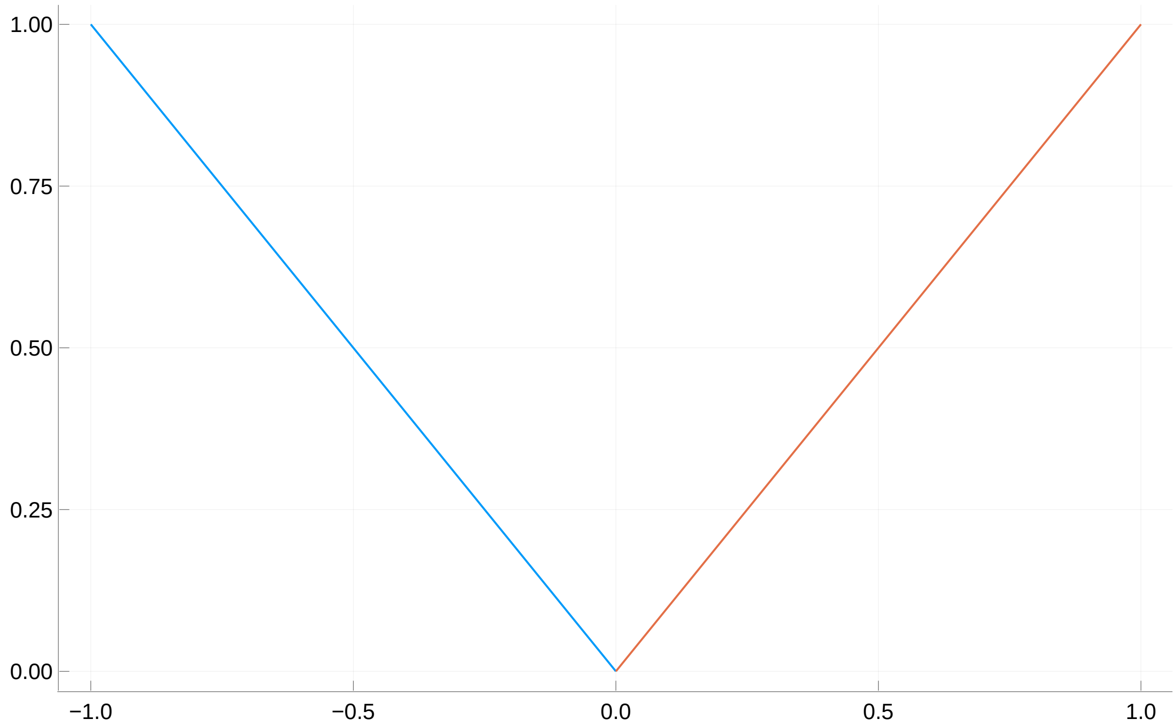

```
begin
    k_y_ = 4 #xscale
    k_x_ = 1//2 #yscale
    A_xy = [
        k_x_ 0;
        0 k_y_;
    ]

    X_xy = [
        1;
        1
    ]

    LA.graph_vectors([(A_xy * X_xy) X_xy])
end
```

Rotating

```
md"# Rotating"
```

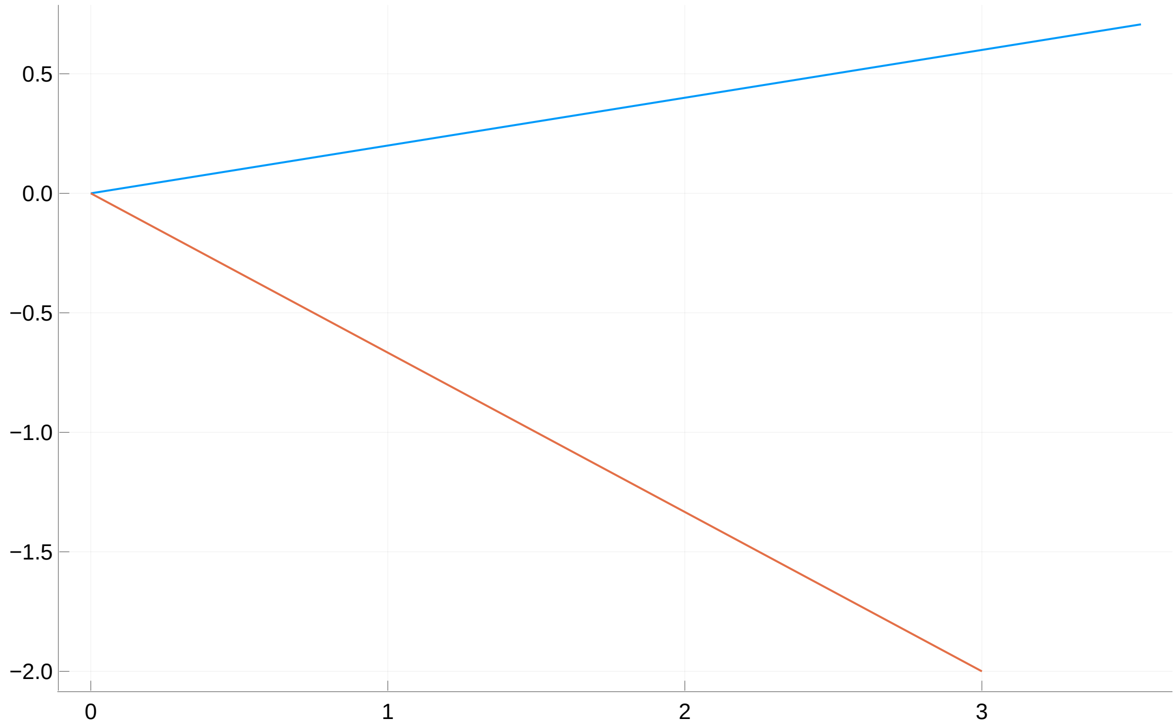
Always counter clockwise

```
md"### Always counter clockwise"
```




```
begin
    θ = pi/2
    B = [
        cos(θ) -sin(θ);
        sin(θ) cos(θ)
    ]

    X_rot = [
        1;
        1
    ]

    println(B)

    LA.graph_vectors([(B * X_rot) X_rot])
end
```

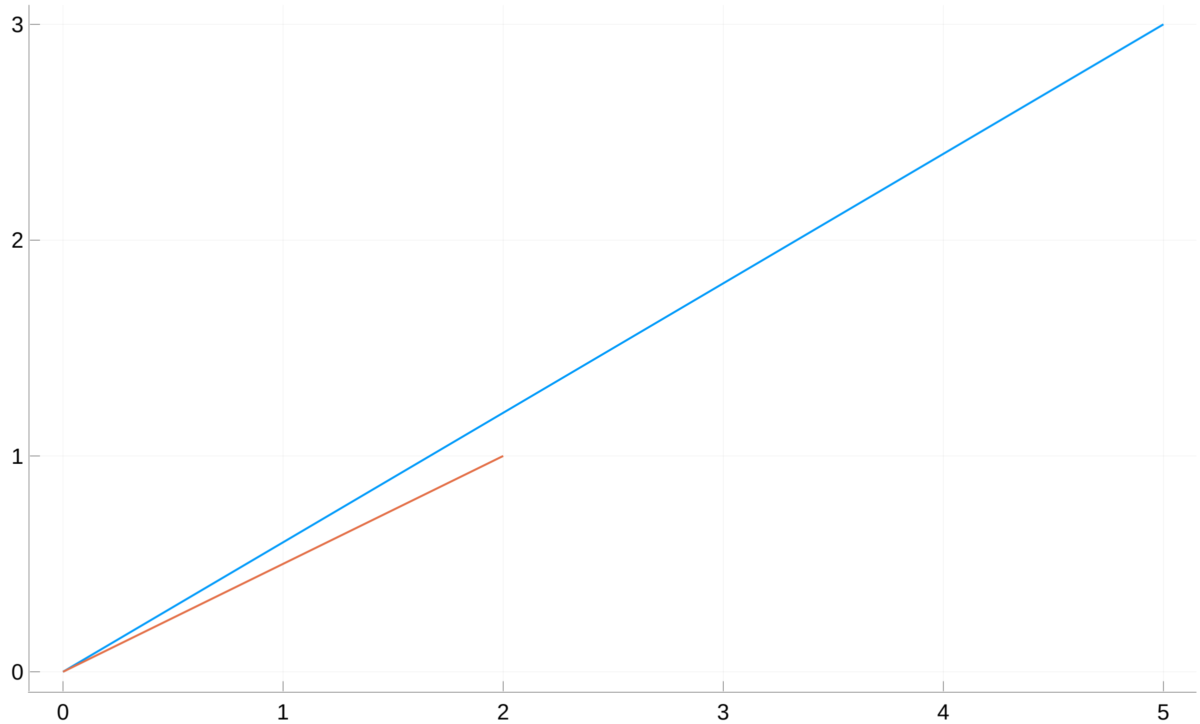
```
[6.123233995736766e-17 -1.0; 1.0 6.123233995736766e-17]
```



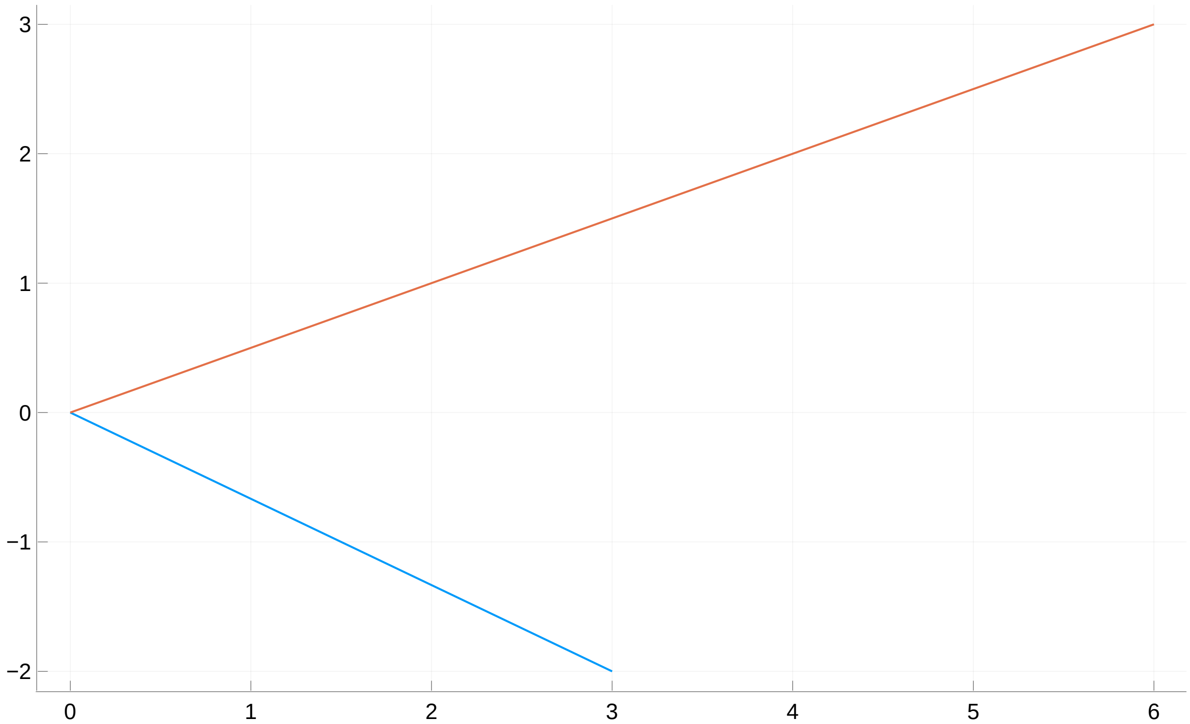

```
begin
    θ_ = pi/4
    X_rot_2 = [3 -2]'
    B_ = [
        cos(θ_) -sin(θ_);
        sin(θ_) cos(θ_)
    ]

    LA.graph_vectors([(B_ * X_rot_2) X_rot_2])
end
```

Translation

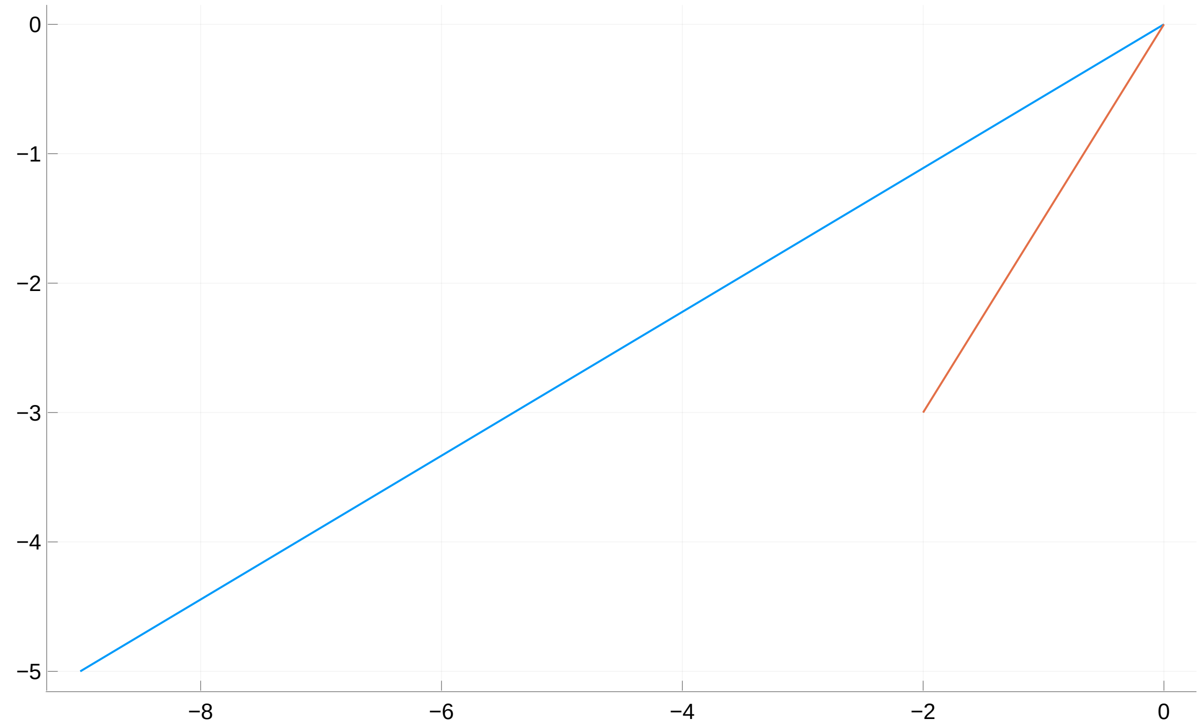
```
md"# Translation"
```




```
begin
    # Addition
    δ_x = 3
    δ_y = 2
    X_trans = [ 2 1]'

    result = X_trans + [δ_x δ_y]'

    LA.graph_vectors([result X_trans])

end
```




```julia
begin
    # Multiplication
    α_x = 3 # right 3
    α_y = 5 # up 5
    placeholder = [3 -2 1]'
    A_mult = [
        1 0 α_x;
        0 1 α_y;
        0 0 1
    ]
    result_mult = A_mult * placeholder
    result_ = result_mult[1:end-1, :]
    LA.graph_vectors([placeholder[1:end-1, :] result_])
end
```




```
begin
    γ_x = -7 # left 7
    γ_y = -2 # down 2
    B_mult = [
        1 0 γ_x;
        0 1 γ_y;
        0 0 1
    ]
    X_mult = [-2 -3 1]'
    result_mult_2 = (B_mult * X_mult)[1:end-1, :]
    LA.graph_vectors([result_mult_2 X_mult[1:end-1, :]])
    # -9, -5
end
```

Rank

```
md"# Rank"
```

The number of nonzero rows in the row echelon form of the matrix.

The number of independent rows.

If transformation squishes matrix into a line, it is rank 1.

If transformation squishes matrix into a plane, it is rank 2.

Rank = Number of Dimensions of the output matrix.

```
md"""### If transformation squishes matrix into a line, it is rank 1.

### If transformation squishes matrix into a plane, it is rank 2.

### Rank = Number of Dimensions of the output matrix.

"""
```

Set of all possible outputs after transformation = Column Space of A = Span of Columns in Matrix.

Rank = Number of dimensions of column space.

Full Rank = Rank(A) == Dim(A)

Zero vector always included in column space.

For full rank, only <0,0> lands on <0,0> after transformation.

For non-full rank, many vectors can get squished into <0,0>.

2d squished into a line (rank 1) => Full line of vectors that land on origin.

3d Transformation gets squished into a line (rank 1) => Full plane that lands on <0, 0, 0>

3d transformation squished into plane (rank 2) => Full line of vectors that land on <0, 0, 0>.

Set of vectors that land on origin = null space = kernel of matrix = space of all vectors that become null (land on zero vector).

When b is zero vector (Ax = b), the null space gives you all the possible solutions to the equation

```
md"### Set of all possible outputs after transformation = Column Space of A = Span of Columns in Matrix.

### Rank = Number of dimensions of column space.

### Full Rank = Rank(A) == Dim(A)

Zero vector always included in column space.

For full rank, only <0,0> lands on <0,0> after transformation.

For non-full rank, many vectors can get squished into <0,0>.

2d squished into a line (rank 1) => Full line of vectors that land on origin.

3d Transformation gets squished into a line (rank 1) => Full plane that lands on <0, 0, 0>

3d transformation squished into plane (rank 2) => Full line of vectors that land on <0, 0, 0>.


```

```
Set of vectors that land on origin = null space = kernel of matrix = space of all
vectors that become null (land on zero vector).


### When b is zero vector (Ax = b), the null space gives you all the possible
```

Nonsquare Matrices

3x2 matrix

3x2 matrix = transformation of 2D vectors into 3D vectors. Where i and j land.

Column space of 3x2 matrix (space where all vectors land after transformation) = 2D plane slicing through origin of 3D space.

But matrix is still full rank since dim(Column Space) == dim(input space).

2x3 Matrix

2x3 matrix = transformation of 3d Vectors into 2 dimensions. Where i, j, and k land.

3 columns == starting in space with 3 basis vectors (i, j, k) (or 3D).

2 rows == landing spot of 3d Vectors only land on space described by 2 coordinates => 2 dimensions.

1x2 Matrix

2D => 1D (number line).

2 columns == starting in space with 2 basis vectors (i, j) (or 2D)

1 row == landing space of 2d vectors only land on space described by 1 coordinate => number line.

Line of evenly spaced dots before transformation (not necessarily passing through origin) == line of evenly spaced dots after transformation (visual understand of what linearity means)

Same as dot product.

```
md" ### Nonsquare Matrices

#### 3x2 matrix

3x2 matrix = transformation of 2D vectors into 3D vectors. Where i and j land.

Column space of 3x2 matrix (space where all vectors land after transformation) = 2D plane slicing through origin of 3D space.

But matrix is still full rank since dim(Column Space) == dim(input space).

----


#### 2x3 Matrix

2x3 matrix = transformation of 3d Vectors into 2 dimensions. Where i, j, and k land.
```

```

3 columns == starting in space with 3 basis vectors (i, j, k) (or 3D).

2 rows == landing spot of 3d Vectors only land on space described by 2 coordinates => 2 dimensions.


#### 1x2 Matrix

2D => 1D (number line).

2 columns == starting in space with 2 basis vectors (i, j) (or 2D)

1 row == landing space of 2d vectors only land on space described by 1 coordinate => number line.

Line of evenly spaced dots before transformation (not necessarily passing through origin) == line of evenly spaced dots after transformation (visual understand of what linearity means)

Same as dot product.





"
```

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

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

```
begin
    C = [
        1 2;
        3 4
    ]
    rref(C)
end
```

```
2
```

```
rank(C)
```

```
D = 2×2 Matrix{Int64}:
     1  2
     2  4
```

```
D = [
    1 2;
    2 4
]
```

```
2×2 Matrix{Float64}:
 1.0  2.0
 0.0  0.0
```

```
rref(D)
```

```
1
```

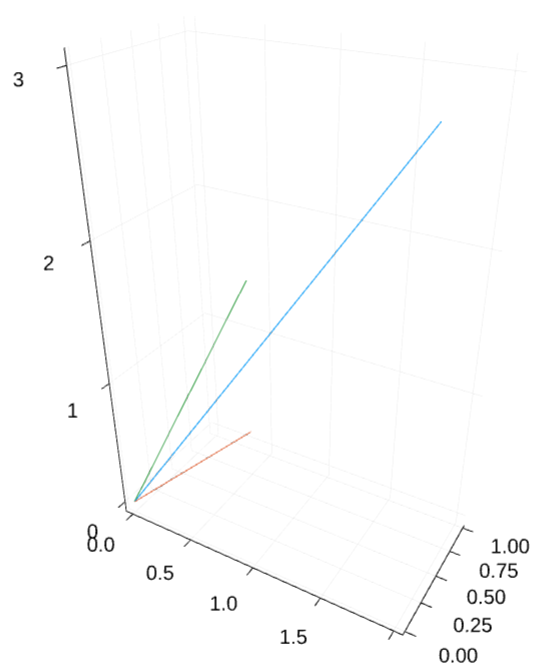
```
rank(D)
```

```
E = 3×3 Matrix{Int64}:
     2  1  1
     0  0  0
     3  1  2
```

```
E = [
    2 1 1;
    0 0 0;
    3 1 2
]
```

```
3×3 Matrix{Float64}:
 1.0  0.0   1.0
 0.0  1.0  -1.0
 0.0  0.0   0.0
```

```
rref(E)
```




```
LA.graph_vectors(E)
```

```
2
```

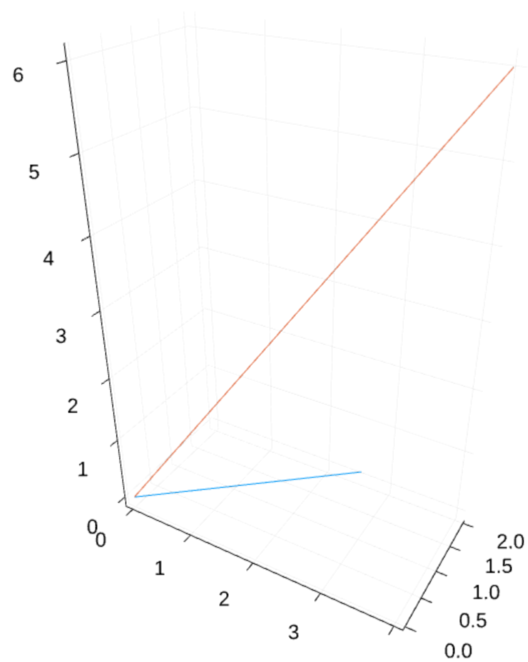
```
rank(E)
```

```
F = 3×2 Matrix{Int64}:
 3  4
 1  2
 1  6
```

```
F = [
    3 4;
    1 2;
    1 6
]
```

```
3×2 Matrix{Float64}:
 1.0  0.0
 0.0  1.0
 0.0  0.0
```

```
rref(F)
```




```
LA.graph_vectors(F)
```

```
2
```

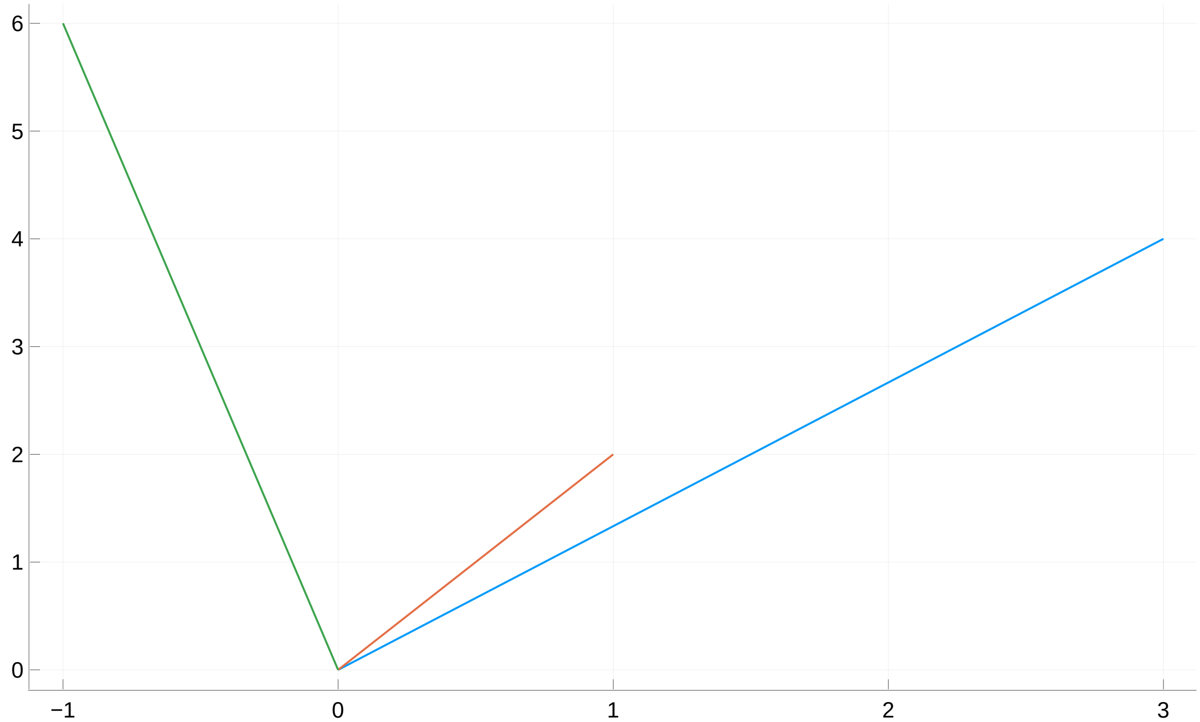
```
rank(F)
```

```
G = 2×3 Matrix{Int64}:
 3  1  -1
 4  2   6
```

```
G = [
    3 1 -1;
    4 2 6
]
```

```
2×3 Matrix{Float64}:
 1.0  0.0  -4.0
 0.0  1.0  11.0
```

```
rref(G)
```




```
LA.graph_vectors(G)
```

2

```
rank(G)
```

Determinants using row operations

```
md"# Determinants using row operations"
```

1. If a multiple of row A is added to another row to make matrix B:

$$det(A) = det(B)$$

R2 = 3R3 + R2

2. If the matrix B is formed by interchanging rows of A:

$$det(A) = -det(B)$$

R1 <-> R2

3. If matrix B is formed by multiplying a row in matrix A by a number k:

$$det(A) = (1/k)det(B)$$

R3 = (1/2)R3

GOAL:

Get matrix to triangular form, then det of matrix A is equal to:

$$det(A) = a_{11} * a_{22} * a_{33}$$

```
true
```

```
begin
    I = [
        1 2 1;
        0 2 1;
        0 0 4
    ]

    # in triangular form
    # multiply diagonals
    1 * 2 * 4 == det(I)
end
```

```
true
```

```
begin
    J = [
        2 3;
        0 9
    ]

    2 * 9 == det(J)
end
```

8

```
prod(diag(UpperTriangular(I)))
```

18

```
prod(diag(UpperTriangular(J)))
```

Cofactors

There is a cofactor located at every position in a matrix

$$C_{ij} = (-1)^{i+j} det(A_{ij})$$

(A_ij is submatrix (row i and column j crossed out))

Finding det with cofactors

about ith row:

$$det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + \dots$$

about jth col:

$$det(A) = a_{1j}C_{1j} + a_{2j}C_{2j} + \dots$$

```
md"""
about ith row:

$det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + \dots$

about jth col:

$det(A) = a_{1j}C_{1j} + a_{2j}C_{2j} + \dots$
"""
```

Adjoints and Inverses

```
K = 3×3 Matrix{Int64}:
     -1  1  2
      0  6  3
      4  7  5
```

```
K = [
    -1 1 2;
    0 6 3;
    4 7 5
]
```

```
3×3 Matrix{Float64}:
  9.0   12.0  -24.0
  9.0  -13.0   11.0
 -9.0    3.0   -6.0
```

```
LA.cofactor(K)
```

```
adj = 3×3 transpose(::Matrix{Float64}) with eltype Float64:
         9.0    9.0  -9.0
        12.0  -13.0   3.0
       -24.0   11.0  -6.0
```

```
adj = transpose(LA.cofactor(K))
```

```
3×3 transpose(::Matrix{Float64}) with eltype Float64:
   9.0    9.0  -9.0
  12.0  -13.0   3.0
 -24.0   11.0  -6.0
```

```
LA.adjugate(K)
```

```
true
```

```
LA.adjugate(K) == adj

```

Properties of adjoints

$$A * adjugate(A) = det(A) * I$$

```
3×3 Matrix{Float64}:
 -45.0    0.0    0.0
   0.0  -45.0    0.0
   0.0    0.0  -45.0
```

```
K * LA.adjugate(K)
```

$$inv(A) = (1/det(A)) * adjugate(A)$$

```
inv_ = 3×3 Matrix{Float64}:
        -0.2        -0.2         0.2
        -0.266667    0.288889  -0.0666667
         0.533333  -0.244444    0.133333
```

```
inv_ = 1/det(K) * LA.adjugate(K)
```

```
3×3 Matrix{Float64}:
 -0.2        -0.2         0.2
 -0.266667    0.288889  -0.0666667
  0.533333  -0.244444    0.133333
```

```
inv(K)
```

```
3×3 Matrix{Rational{Int64}}:
 -1//5    -1//5    1//5
 -4//15   13//45  -1//15
  8//15  -11//45   2//15
```

```
inv(Rational.(K))
```

```
3×3 transpose(::Matrix{Float64}) with eltype Float64:
   9.0    9.0  -9.0
  12.0  -13.0   3.0
 -24.0   11.0  -6.0
```

```
LA.adjugate(K)
```

As long as det(M) != 0, there will be an inv(M)

```
md"### As long as det(M) != 0, there will be an inv(M)"
```

If det(M) == 0, then there may be an inv(M), but you have to be lucky. Solution must live on that line.

```
md"### If det(M) == 0, then there may be an inv(M), but you have to be lucky.
Solution must live on that line."
```

Properties of Determinants

If any row or col of a matrix is all zeros, then the determinant is also zero

```
L = 3×3 Matrix{Int64}:
 1  2  3
 0  0  0
 4  5  6
```

```julia
L = [
    1 2 3;
    0 0 0;
    4 5 6
]
```

```
0.0
```

```julia
det(L)
```

If any two rows or cols are proportional to each other, then the determinant is also zero

```julia
md"### If any two rows or cols are proportional to each other, then the determinant is also zero"
```

```
M = 2×2 Matrix{Int64}:
 1  2
 2  4
```

```julia
M = [
    1 2;
    2 4
]
```

```
0.0
```

```julia
det(M)
```

```
false
```

```julia
LA.is_LI(M)
```

```
2×2 Matrix{Float64}:
 1.0  2.0
 0.0  0.0
```

```julia
rref(M)
```

```
0.0
```

```julia
det(rref(M))
```

If row/col is linear combo of other rows/cols, then determinant is zero

Det of transpose of matrix is equal to the determinant of the matrix

$$det(A^T) = det(A)$$

The determinant of two matrices multiplied together is equal to the det(A)*det(B)

$$det(AB) = det(A) * det(B)$$

Homogeneous System of Equations (square matrices)

$$AX = 0$$

Solution is non-trivial iff det(A) == 0

If det(A) != 0, then X = 0 is only solution (trivial case)

Linear Independence and Basis

- Solution to homogeneous equation's free variables' basis vectors are linearly independent

Vectors are LI if:

$$c_1u_1 + c_2u_2 + c_3u_3 + \cdots + c_ku_k = 0$$

only if

$$c_1, c_2, c_3, \ldots, c_k$$

are all zero.

Not LI example: $u_1 = 3u_2 + 2u_3$

Test:

$$\langle c_1 u_1, c_2 u_2, \ldots, c_k u_k \rangle$$

as matrix A.

If det(A) != 0, then LI.

$$u_1 = \langle 1, 0, 0 \rangle, u_2 = \langle 0, 1, 0 \rangle, u_3 = \langle 1, 1, 0 \rangle$$

```
md"$u_1 = \langle 1, 0, 0 \rangle, u_2 = \langle 0, 1, 0 \rangle, u_3 = \langle 1, 1, 0 \rangle$"
```

But $u_3 = u_1 + u_2$ => Not LI

```
md"But $u_3 = u_1 + u_2$ => Not LI"
```

```
O = 3×3 Matrix{Int64}:
 1  0  1
 0  1  1
 0  0  0
```

```
O = [
    1 0 1;
    0 1 1;
    0 0 0
]
```

```
0.0
```

```
det(O)
```

```
false
```

```
LA.is_LI(O)
```

$$u_3 = u_1 + u_2$$

New vectors:

$$u_1 = \langle 1, 1, -1 \rangle, u_2 = \langle 0, 1, 1 \rangle, u_3 = \langle 1, 2, 3 \rangle$$

```
P = 3×3 Matrix{Int64}:
  1  0  1
  1  1  2
 -1  1  3
```

```
P = [
    1 0 1;
    1 1 2;
    -1 1 3
]
```

```
3.0
```

```
det(P)
```

```
true
```

```
LA.is_LI(P)
```

Eigenvalues, Eigenvectors

```
md"# Eigenvalues, Eigenvectors"
```

The Eigenvalue Problem

```
md"## The Eigenvalue Problem"
```

$$Ax = \lambda x$$

λ = eigenvalue (scalar)

x = eigenvector

For every eigenvalue λ there is a set of nontrivial vectors x called the eigenvectors

```
2×1 Matrix{Int64}:
 -24
  20
```

```
begin
    Q = [
        1 6;
        5 2
    ]

    x_1 = [6 -5]'

    # Is x an eigenvector?

    #Ax = λx

    result_eig = Q * x_1
end
```

```
2×1 Matrix{Float64}:
 -4.0
 -4.0
```

```
result_eig ./ x_1
```

$$\lambda = -4$$

```
md"$\lambda = -4$"
```

```
[-4.0, 7.0]
```

```
eigvals(Q)
```

```
2×2 Matrix{Float64}:
 -0.768221  -0.707107
  0.640184  -0.707107
```

```
eigvecs(Q)
```

```
val = 2×1 Matrix{Float64}:
       -7.810249675906654
       -7.810249675906655
```

```
val = [6 -5]' ./ eigvecs(Q)[:, 1]
```

Example 2

```
2×1 Matrix{Int64}:
 -9
 11
```

```julia
begin
    R = [
        1 6;
        5 2
    ]

    x_R = [3 -2]'

    result_R = R * x_R
end
```

```
2×1 Matrix{Float64}:
 -3.0
 -5.5
```

```julia
result_R ./ x_R
```

NOT an eigenvector!

```julia
md"NOT an eigenvector!"
```

```
[-4.0, 7.0]
```

```julia
eigvals(R)
```

```
2×2 Matrix{Float64}:
 -0.768221  -0.707107
  0.640184  -0.707107
```

```julia
eigvecs(R)
```

Example 3: 3x3 matrix

```julia
md"Example 3: 3x3 matrix"
```

```
3×1 Matrix{Int64}:
 3
 5
 7
```

```julia
begin
    S = [
        2 2 1;
        1 3 1;
        1 2 2
    ]

    x_S = [-1 1 3]'

    result_S = S * x_S
end
```

```
3×1 Matrix{Float64}:
 -3.0
  5.0
  2.3333333333333335
```

```
result_S ./x_S
```

NOT an eigenvector!

```
md"NOT an eigenvector!"
```

```
3×1 Matrix{Int64}:
 5
 5
 5
```

```
begin
    T = [
        2 2 1;
        1 3 1;
        1 2 2
    ]

    x_T = [1 1 1]'

    result_T = T * x_T
end
```

```
3×1 Matrix{Float64}:
 5.0
 5.0
 5.0
```

```
result_T ./ x_T
```

$$\lambda = 5$$

Find Eigenvalues and eigenvectors from scratch

Find eigenvalue

$$Ax = \lambda x$$

$$Ax = \lambda I x$$

$$Ax - \lambda I x = 0$$

$$(A - \lambda I)x = 0$$

This equation has nontrivial solns when det(A - λI) == 0

Example 1

```
md"Example 1"
```

```
U = 2×2 Matrix{Int64}:
 1  2
 2  1
```

```
U = [
        1 2;
        2 1
    ]
```

```
char_U = #1 (generic function with 1 method)
```

```
char_U = λ -> (1 - λ)*(1 - λ) - (2)*(2)
```

```
[-1.0, 3.0]
```

```
find_zeros(char_U, -100, 100)
```

$A - \lambda I$ is the characteristic matrix

```
md"### $A-\lambda I$ is the characteristic matrix"
```

```
[-1.0, 3.0]
```

```
eigvals(U)
```

Example 2

```
md"Example 2"
```

```
V = 2×2 Matrix{Int64}:
 2  7
 7  2
```

```
V = [
    2 7;
    7 2
]
```

```
char_V = #3 (generic function with 1 method)
```

```
char_V = λ -> (2 - λ)*(2 - λ) - (7)(7)
```

```
[-5.0, 9.0]
```

```
find_zeros(char_V, -100, 100)
```

```
[-5.0, 9.0]
```

```
eigvals(V)
```

Example 3

```
md"Example 3"
```

```
W = 2×2 Matrix{Int64}:
     3  -2
     1  -1
```

```
W = [
    3 -2;
    1 -1
]
```

```
char = #5 (generic function with 1 method)
```

```
char = λ -> (3 - λ)*(-1 - λ) - (-2)
```

```
[-0.414214, 2.41421]
```

```
find_zeros(char, -100, 100)
```

```
[-0.414214, 2.41421]
```

```
eigvals(W)
```

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
md"$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$"
```

```
2.414213562373095
```

```
1 + sqrt(2)
```

```
-0.41421356237309515
```

```
1 - sqrt(2)
```

Example 4

```
md"Example 4"
```

```
Y = 3×3 Matrix{Int64}:
  5   4   4
 -7  -3  -1
  7   4   2
```

```julia
Y = [
    5 4 4;
    -7 -3 -1;
    7 4 2
]
```

```
char_Y = #7 (generic function with 1 method)
```

```julia
char_Y = λ -> (5 - λ)*(λ^2 + λ - 2) - 4*(7*λ -7) + 4*(7*λ - 7)
```

```
[-2.0, 1.0, 5.0]
```

```julia
find_zeros(char_Y, -100, 100)
```

```
[-2.0, 1.0, 5.0]
```

```julia
eigvals(Y)
```

Example 5

```julia
md"Example 5"
```

```
Z = 4×4 Matrix{Int64}:
 5  -2   6  -1
 0   3  -8   0
 0   0   5   4
 0   0   0   1
```

```julia
Z = [
    5 -2 6 -1;
    0 3 -8 0;
    0 0 5 4;
    0 0 0 1
]
```

```
[1.0, 3.0, 5.0, 5.0]
```

```julia
eigvals(Z)
```

```julia
# Upper Triangular Matrix! Use shortcut of diagonal prods
```

```
char_Z = #9 (generic function with 1 method)
```

```julia
char_Z = λ -> (5 - λ)*(3 - λ)*(5 - λ)*(1-λ)
```

```
[1.0, 3.0, 5.0]
```

```julia
find_zeros(char_Z, -100, 100)
```

Finding eigenvectors

```
2×2 Matrix{Int64}:
 1  2
 2  1
```

```
U
```

```
[-1.0, 3.0]
```

```
eigvals(U)
```

```
2×2 Matrix{Float64}:
 -0.707107  0.707107
  0.707107  0.707107
```

```
eigvecs(U)
```

```
2×2 Matrix{Int64}:
 2  2
 2  2
```

```
begin
    λ_U = -1
    matrix_U = U - λ_U * LA.identity(2)
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: 1* [inlined]

```
LA.aug_solve([matrix_U [0 0]'])
```

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

```
rref(matrix_U)
```

for lambda = -1

$$X = \langle -k, k \rangle$$

or

$$X = k\langle -1, 1 \rangle$$

```
md"""$X = \langle -k, k \rangle$

or

$X = k\langle -1, 1 \rangle$"""
```

Eigenvector: for lambda = -1: k <-1, 1>

```
2×2 Matrix{Int64}:
 -2   2
  2  -2
```

```
begin
    λ_U_3 = 3
    matrix_U_3 = U - λ_U_3 * LA.identity(2)
end
```

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

```
rref(matrix_U_3)
```

$$x_2 = k$$

$$x_1 - k = 0, x_1 = k$$

$$X = \langle k, k \rangle = k\langle 1, 1 \rangle$$

eigenbasis: [-1, 1], [1, 1]

```
true
```

```
begin
    eig_neg1 = [-1 1]'
    eig_3 = [1 1]'
    LA.is_LI([eig_neg1 eig_3])
end
```

3x3 example

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

```
function eig_result(M::AbstractMatrix, lambda::Real)
    return M - lambda * LA.identity(size(M, 1))
end
```

```
3×3 Matrix{Int64}:
  5   4   4
 -7  -3  -1
  7   4   2
```

```
Y
```

```
result_Y = 3×3 Matrix{Int64}:
             0   4   4
            -7  -8  -1
             7   4  -3
```

```
result_Y = eig_result(Y, 5)
```

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

```
function eig_solve(M::AbstractMatrix)
    return rref([M [0 for i=1:Integer(size(M, 1))]])
end
```

```
3×4 Matrix{Float64}:
 1.0  0.0  -1.0  -0.0
 0.0  1.0   1.0   0.0
 0.0  0.0   0.0   0.0
```

```
eig_solve(result_Y)
```

for lambda = 5, X = [k, -k, k] = k[1, -1, 1] for (x_3 = k)

```
3×4 Matrix{Float64}:
 1.0  0.0  -1.0  -0.0
 0.0  1.0   2.0   0.0
 0.0  0.0   0.0   0.0
```

```
eig_solve(eig_result(Y, 1))
```

for lambda = 1, X = k[1, -2, 1] for x_3 = k

```
3×4 Matrix{Float64}:
 1.0  0.0  0.0  0.0
 0.0  1.0  1.0  0.0
 0.0  0.0  0.0  0.0
```

```
eig_solve(eig_result(Y, -2))
```

for lambda = -2, X = k[0, -1, 1] for x_3 = k

Basis Vectors: <1, -1, 1>, <1, -2, 1>, <0, -1, 1>

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

```
function bluebrown_eigvals(M::AbstractMatrix)
    # Must be 2x2 matrix
    if Integer(size(M, 1)) != 2 && Integer(size(M, 2)) != 2
        throw("Not 2x2 Matrix")
    end
    # trace: sum of diags
    m = tr(M)/2
    p = det(M)
    square = sqrt(m^2 - p)
    return [m + square, m - square]
end


```

[7.0, -4.0]

```
bluebrown_eigvals(Q)
```

[7.0, -4.0]

```
bluebrown_eigvals(R)
```

[3.0, -1.0]

```
bluebrown_eigvals(U)
```

[9.0, -5.0]

```
bluebrown_eigvals(V)
```

[2.41421, -0.414214]

```
bluebrown_eigvals(W)
```