Fondamenti di Informatica (Elettronici)

Vettori e matrici in Julia – Lezione 35

10 gennaio 2021

Matrici in Julia[1]

1. Matrices in Julia
2. Block matrices
3. Matrix operations
4. Other functions
5. Linear equation systems

[1] Tratto da: Pathak, Go, Zeng, and Boyd, EE108: Introduction to Matrix Methods, Stanford University, 2016.

Section 1

Matrices in Julia

Matrices

- matrices in Julia are repersented by 2D arrays

Matrices

- matrices in Julia are repersented by 2D arrays
- `[2 -4 8.2; -5.5 3.5 63]` creates the 2×3 matrix

$$A = \begin{bmatrix} 2 & -4 & 8.2 \\ -5.5 & 3.5 & 63 \end{bmatrix}$$

Matrices

- matrices in Julia are repersented by 2D arrays
- `[2 -4 8.2; -5.5 3.5 63]` creates the 2×3 matrix

$$A = \begin{bmatrix} 2 & -4 & 8.2 \\ -5.5 & 3.5 & 63 \end{bmatrix}$$

Matrices

- matrices in Julia are repersented by 2D arrays
- `[2 -4 8.2; -5.5 3.5 63]` creates the 2×3 matrix

$$A = \begin{bmatrix} 2 & -4 & 8.2 \\ -5.5 & 3.5 & 63 \end{bmatrix}$$

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0
```

- spaces separate entries in a row; semicolons separate rows,

Matrices

- matrices in Julia are repersented by 2D arrays
- `[2 -4 8.2; -5.5 3.5 63]` creates the 2×3 matrix

$$A = \begin{bmatrix} 2 & -4 & 8.2 \\ -5.5 & 3.5 & 63 \end{bmatrix}$$

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0
```

- spaces separate entries in a row; semicolons separate rows,

Matrices

- matrices in Julia are repersented by 2D arrays
- `[2 -4 8.2; -5.5 3.5 63]` creates the 2×3 matrix

$$A = \begin{bmatrix} 2 & -4 & 8.2 \\ -5.5 & 3.5 & 63 \end{bmatrix}$$

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0
```

- spaces separate entries in a row; semicolons separate rows, but we can also write:

```
julia> A = [2.0  -4.0   8.2
           -5.5   3.5  63.0]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0
```

Indexing and slicing

- A_{ij} is found with `A[i,j]`

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$
- operator `:` selects all elements along that dimension

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$
- operator `:` selects all elements along that dimension
  - `A[:,3]` is third column

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$
- operator `:` selects all elements along that dimension
  - `A[:,3]` is third column
  - `A[2,:]` is second row

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$
- operator `:` selects all elements along that dimension
  - `A[:,3]` is third column
  - `A[2,:]` is second row
- `A[:]` stacks the columns of A as a vector (column-major order)

Indexing and slicing

- A_{ij} is found with `A[i,j]`
- can use ranges:
  - `A[1:2,1:3]` is 2×3 submatrix or slice $A_{1:2,1:3}$
- operator `:` selects all elements along that dimension
  - `A[:,3]` is third column
  - `A[2,:]` is second row
- `A[:]` stacks the columns of A as a vector (column-major order)
- `A'[:]` stacks the rows of A as a vector (row-major order)

Indexing and slicing – Example

```
julia> A
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0

julia> A[:]
6-element Array{Float64,1}:
  2.0
 -5.5
 -4.0
  3.5
  8.2
 63.0

julia> A'[:]
6-element Array{Float64,1}:
  2.0
 -4.0
  8.2
 -5.5
  3.5
 63.0
```

Matrices

- size(A) returns the size of A as a pair, i.e.,

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0

julia> A_rows, A_cols = size(A)
(2, 3)

julia> A_rows == size(A)[1]  &&  A_cols == size(A)[2]
true
```

Matrices

- size(A) returns the size of A as a pair, i.e.,

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0

julia> A_rows, A_cols = size(A)
(2, 3)

julia> A_rows == size(A)[1]  &&  A_cols == size(A)[2]
true
```

Matrices

- size(A) returns the size of A as a pair, i.e.,

```
julia> A = [2 -4 8.2; -5.5 3.5 63]
2×3 Array{Float64,2}:
  2.0  -4.0   8.2
 -5.5   3.5  63.0

julia> A_rows, A_cols = size(A)
(2, 3)

julia> A_rows == size(A)[1]  &&  A_cols == size(A)[2]
true
```

- row vectors are $1 \times n$ matrices, e.g.:

```
julia> [4 8.7 -9]
1×3 Array{Float64,2}:
 4.0  8.7  -9.0
```

Section 2

Block matrices

Block matrices

- block matrix

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

(with A, B, C, and D matrices) is formed with

```
X = [A B; C D]
```

Block matrices

- block matrix

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

(with A, B, C, and D matrices) is formed with

```
X = [A B; C D]
```

- usual rules governing dimensions of A, B, C, and D apply

Block matrices – Examples

```
julia> A = [1 2 3 4]
1×4 Array{Int64,2}:
 1  2  3  4

julia> B = [5 6 7 8]
1×4 Array{Int64,2}:
 5  6  7  8

julia> C = [A B]
1×8 Array{Int64,2}:
 1  2  3  4  5  6  7  8

julia> D = [A; B]
2×4 Array{Int64,2}:
 1  2  3  4
 5  6  7  8

julia> E = [C; D D]
3×8 Array{Int64,2}:
 1  2  3  4  5  6  7  8
 1  2  3  4  1  2  3  4
 5  6  7  8  5  6  7  8
```

Block matrix product 1/2

```
julia> A = [0 2; 2 0]
2×2 Array{Int64,2}:
 0  2
 2  0

julia> B = [3 3 3; 3 3 3]
2×3 Array{Int64,2}:
 3  3  3
 3  3  3

julia> C = [4 4; 4 4; 4 4]
3×2 Array{Int64,2}:
 4  4
 4  4
 4  4

julia> D = [5 0 5; 0 5 0; 5 0 5]
3×3 Array{Int64,2}:
 5  0  5
 0  5  0
 5  0  5

julia> [A B
        C D]
5×5 Array{Int64,2}:
 0  2  3  3  3
 2  0  3  3  3
 4  4  5  0  5
 4  4  0  5  0
 4  4  5  0  5
```

Block matrix product 1/2

```
julia> A = [0 2; 2 0]
2×2 Array{Int64,2}:
 0  2
 2  0

julia> C = [4 4; 4 4; 4 4]
3×2 Array{Int64,2}:
 4  4
 4  4
 4  4
```

```
julia> B = [3 3 3; 3 3 3]
2×3 Array{Int64,2}:
 3  3  3
 3  3  3

julia> D = [5 0 5; 0 5 0; 5 0 5]
3×3 Array{Int64,2}:
 5  0  5
 0  5  0
 5  0  5
```

```
julia> [A B
        C D]
5×5 Array{Int64,2}:
 0  2  3  3  3
 2  0  3  3  3
 4  4  5  0  5
 4  4  0  5  0
 4  4  5  0  5
```

Remark

When two block matrices have the same shape and their diagonal blocks are square matrices, then they multiply similarly to matrix multiplication.

Block matrix product 2/2

```
julia> A_1 = A;    A_2 = 2*A;
julia> B_1 = B;    B_2 = 3*B;
julia> C_1 = 4*C;  C_2 = C;
julia> D_1 = 2*D;  D_2 = D;

julia> [A_1 B_1; C_1 D_1] * [A_2 B_2; C_2 D_2]
5×5 Array{Int64,2}:
  44   36   48   33   48
  36   44   48   33   48
 144  144  388  288  388
 104  104  288  338  288
 144  144  388  288  388

julia> [A_1*A_2+B_1*C_2   A_1*B_2+B_1*D_2;
        C_1*A_2+D_1*C_2   C_1*B_2+D_1*D_2]
5×5 Array{Int64,2}:
  44   36   48   33   48
  36   44   48   33   48
 144  144  388  288  388
 104  104  288  338  288
 144  144  388  288  388
```

Identity matrices

```
julia> Matrix(1I, 3, 3)     #Identity matrix of Int type
3×3 Array{Int64,2}:
 1  0  0
 0  1  0
 0  0  1

julia> Matrix(1.0I, 3, 3)   #Identity matrix of Float64 type
3×3 Array{Float64,2}:
 1.0  0.0  0.0
 0.0  1.0  0.0
 0.0  0.0  1.0

julia> Matrix(I, 3, 3)      #Identity matrix of Bool type
3×3 Array{Bool,2}:
 1  0  0
 0  1  0
 0  0  1

Matrix{Int}(I, 3, 3);       #Identity matrix of Int type
Matrix{Float64}(I, 3, 3);   #Identity matrix of Float64 type
Matrix{Bool}(I, 3, 3);      #Identity matrix of Bool type
```

Useful matrices

$\mathbf{0}_{m\times n}$

```
julia> zeros(3,4)
3×4 Array{Float64,2}:
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
```

$\mathbf{1}_{m\times n}$

```
julia> ones(3,4)
3×4 Array{Float64,2}:
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
```

$\mathbf{diag}(x)$

```
julia> diagm([10.,20.,30.])
3×3 Array{Float64,2}:
 10.0   0.0   0.0
  0.0  20.0   0.0
  0.0   0.0  30.0
```

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)
- +/− are used for matrix addition/subtraction

  (matrices must have the same size)

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)
- +/− are used for matrix addition/subtraction

  (matrices must have the same size)

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)
- +/- are used for matrix addition/subtraction

  (matrices must have the same size)
- for example:

```
julia> A = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 15.8  -34.5
 46.9  -81.2

julia> B = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 -42.5  30.5
  57.5  96.4
```

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)
- +/- are used for matrix addition/subtraction

  (matrices must have the same size)
- for example:

```
julia> A = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 15.8  -34.5
 46.9  -81.2

julia> B = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 -42.5  30.5
  57.5  96.4
```

Transpose and matrix addition

- $A^\top$ is written A' (single quote mark)
- +/- are used for matrix addition/subtraction

  (matrices must have the same size)
- for example:

```
julia> A = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 15.8  -34.5
 46.9  -81.2

julia> B = rand(-99:.1:99, 2,2)
2×2 Array{Float64,2}:
 -42.5  30.5
  57.5  96.4
```

- and we may write:

```
julia> A + B == [15.8  -34.5; 46.9  -81.2] + [-42.5  30.5; 57.5  96.4]
true
```

Section 3

Matrix operations

Matrix-scalar operations

- matrix-scalar operations (`+`,`-`,`*`,`\`) apply elementwise

Matrix-scalar operations

- matrix-scalar operations (`+`,`-`,`*`,`\`) apply elementwise

Matrix-scalar operations

- matrix-scalar operations (`+`,`-`,`*`,`\`) apply elementwise
- scalar-matrix multiplication:

  `10 * [1 2; 3 4]`

  gives

$$10 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

  (scalar can also appear on right of matrix)

Matrix-scalar operations

- matrix-scalar operations (`+`,`-`,`*`,`\`) apply elementwise
- scalar-matrix multiplication:

  `10 * [1 2; 3 4]`

  gives

$$10 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

  (scalar can also appear on right of matrix)

Matrix-scalar operations

- matrix-scalar operations (`+`,`-`,`*`,`\`) apply elementwise
- scalar-matrix multiplication:

  `10 * [1 2; 3 4]`

  gives

$$10 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

  (scalar can also appear on right of matrix)
- matrix-scalar addition:

  `[1 2; 3 4] .+ 10`

  gives

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix} = \begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}$$

  (which is not standard mathematical notation)

Matrix-vector multiplication

- the $*$ operator is used for matrix-vector multiplication

Matrix-vector multiplication

- the * operator is used for matrix-vector multiplication
- for example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

is written `[1 2; 3 4] * [5, 6]`

Matrix multiplication

- the * operator is also used for matrix-matrix multiplication:

$$\begin{bmatrix} 2 & 4 & 3 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} 3 & 10 \\ 4 & 2 \\ 1 & 7 \end{bmatrix}$$

is written

```
[2 4 3; 3 1 5] * [3 10; 4 2; 1 7]
```

Matrix multiplication

- the * operator is also used for matrix-matrix multiplication:

$$\begin{bmatrix} 2 & 4 & 3 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} 3 & 10 \\ 4 & 2 \\ 1 & 7 \end{bmatrix}$$

  is written

```
[2 4 3; 3 1 5] * [3 10; 4 2; 1 7]
```

- `A^k` is Julia notation for A^k

  (for square matrix A)

Section 4

Other functions

Other functions

- sum of entries of a matrix: `sum(A)`

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`
- `max(A,B)` and `min(A,B)` finds the element-wise max and min respectively

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`
- `max(A,B)` and `min(A,B)` finds the element-wise max and min respectively
  - the arguments must have the same size unless one is a scalar

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`
- `max(A,B)` and `min(A,B)` finds the element-wise max and min respectively
  - the arguments must have the same size unless one is a scalar
- `norm(A, p::Real=2)` computes the p-norm (defaulting to `p=2`) as if A were a vector of the corresponding length.

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`
- `max(A,B)` and `min(A,B)` finds the element-wise max and min respectively
  - the arguments must have the same size unless one is a scalar
- `norm(A, p::Real=2)` computes the p-norm (defaulting to `p=2`) as if A were a vector of the corresponding length.

Other functions

- sum of entries of a matrix: `sum(A)`
- average of entries of a matrix: `mean(A)`
- `max(A,B)` and `min(A,B)` finds the element-wise max and min respectively
  - the arguments must have the same size unless one is a scalar
- `norm(A, p::Real=2)` computes the p-norm (defaulting to `p=2`) as if A were a vector of the corresponding length.

The p-norm is defined as

$$\|A\|_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{1/p}$$

with a_i the entries of A, $|a_i|$ the `abs()` of a_i, and n the length of A.

Matrix eigenvalues and eigevectors

```
julia> using LinearAlgebra

julia> A = [1  2 3; 0 7 2; 4 5 -1]
3×3 Array{Int64,2}:
 1  2   3
 0  7   2
 4  5  -1

julia> val, vec = eigen(A)
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
values:
3-element Array{Float64,1}:
 -4.0123101098426925
  2.484079559108717
  8.528230550733978
vectors:
3×3 Array{Float64,2}:
  0.459669  -0.793209  -0.41267
  0.158695   0.246589  -0.723771
 -0.873797  -0.556788  -0.553045
```

Rank of a Matrix

```
julia> A = rand(3,3)
3×3 Array{Float64,2}:
 0.9018     0.143928  0.58286
 0.0494715  0.191993  0.600828
 0.88409    0.371667  0.273628

julia> rank(A)
3

julia> B = rand(3,4)
3×4 Array{Float64,2}:
 0.876633  0.458907   0.78194   0.654843
 0.139805  0.0171215  0.666739  0.252584
 0.662399  0.153036   0.20327   0.942724

julia> rank(rand(3,3))
3
```

```
julia> C = rand(8,2)
8×2 Array{Float64,2}:
 0.361777   0.698763
 0.29189    0.663315
 0.0518133  0.878524
 0.489656   0.685433
 0.467383   0.524349
 0.184916   0.536337
 0.703423   0.715112
 0.258969   0.884622

julia> rank(C)
2
```

Section 5

Linear equation systems

Example with Cramer rule

```
function cramersolve(A::Matrix, b::Vector)
return collect(begin B = copy(A); B[:, i] = b; det(B) end
    for i in eachindex(b)) ./det(A)
end

julia> A = [2 -1  5  1
            3  2  2 -6
            1  3  3 -1
            5 -2 -3  3];

julia> b = [-3, -32, -47, 49];

julia> @show cramersolve(A, b);
   2.0000000000000004
 -11.999999999999998
  -4.0
   1.0000000000000009

julia> x = inv(A) * b
4-element Array{Float64,1}:
   1.9999999999999996
 -12.0
  -3.9999999999999996
```

Rouché-Capelli theorem

Let us consider a linear system of m equations in n unknowns with coefficients in a field $\mathbb{K}$. Write the system in matrix form

$$A\mathbf{x} = \mathbf{b}, \quad \text{where} \quad A \in \mathbb{K}^{m,n}, \mathbf{x} \in \mathbb{K}^{n,1}, \mathbf{b} \in \mathbb{K}^{m,1}.$$

The Rouché-Capelli theorem states that

1. If $\mathbf{rk}(A) < \mathbf{rk}(A \,|\, b)$, the system is impossible, and no solution exists.

Rouché-Capelli theorem

Let us consider a linear system of m equations in n unknowns with coefficients in a field $\mathbb{K}$. Write the system in matrix form

$$A\mathbf{x} = \mathbf{b}, \quad \text{where} \quad A \in \mathbb{K}^{m,n}, \mathbf{x} \in \mathbb{K}^{n,1}, \mathbf{b} \in \mathbb{K}^{m,1}.$$

The Rouché-Capelli theorem states that

1. If $\mathbf{rk}(A) < \mathbf{rk}(A \,|\, b)$, the system is impossible, and no solution exists.
2. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b)$, the system is compatible, and solutions do exist.

Rouché-Capelli theorem

Let us consider a linear system of m equations in n unknowns with coefficients in a field $\mathbb{K}$. Write the system in matrix form

$$A\mathbf{x} = \mathbf{b}, \quad \text{where} \quad A \in \mathbb{K}^{m,n}, \mathbf{x} \in \mathbb{K}^{n,1}, \mathbf{b} \in \mathbb{K}^{m,1}.$$

The Rouché-Capelli theorem states that

1. If $\mathbf{rk}(A) < \mathbf{rk}(A \,|\, b)$, the system is impossible, and no solution exists.
2. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b)$, the system is compatible, and solutions do exist.
3. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b) = n$, then only one solution exists.

Rouché-Capelli theorem

Let us consider a linear system of m equations in n unknowns with coefficients in a field $\mathbb{K}$. Write the system in matrix form

$$A\mathbf{x} = \mathbf{b}, \quad \text{where} \quad A \in \mathbb{K}^{m,n}, \mathbf{x} \in \mathbb{K}^{n,1}, \mathbf{b} \in \mathbb{K}^{m,1}.$$

The Rouché-Capelli theorem states that

1. If $\mathbf{rk}(A) < \mathbf{rk}(A \,|\, b)$, the system is impossible, and no solution exists.
2. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b)$, the system is compatible, and solutions do exist.
3. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b) = n$, then only one solution exists.
4. If $\mathbf{rk}(A) = \mathbf{rk}(A \,|\, b) < n$, then there are $\infty^{n-\mathbf{rk}(A)}$ solutions.

The left Matrix divide

The left matrix divide is roughly the same as

$$A \setminus b = A^{-1} b$$

This technique can be used to quickly compute the solution of the equation

$$Ax = b \Rightarrow x = A \setminus b$$

The left Matrix divide

The left matrix divide is roughly the same as

$$A \setminus b = A^{-1} b$$

This technique can be used to quickly compute the solution of the equation

$$Ax = b \Rightarrow x = A \setminus b$$

```
julia> A = [1 2 ; 2 2];

julia> B = [3 2 ; 1 1];

julia> A\B
2×2 Array{Float64,2}:
 -2.0  -1.0
  2.5   1.5

julia> A \ [3, 1]
2-element Array{Float64,1}:
 -2.0
  2.5
```

Inverse multiplication (left division)

```
julia> A = rand(1:.001:10, 4,3)
4×3 Array{Float64,2}:
 5.172  5.827  8.954
 6.403  8.315  2.693
 6.358  4.447  2.582
 9.884  8.551  3.69

julia> b = rand(1:.001:5,4)
4-element Array{Float64,1}:
 3.763
 4.469
 3.944
 4.717
```

```
julia> x = A \ b
3-element Array{Float64,1}:
 0.2971516308543035
 0.2425866764309917
 0.09459172375020722

julia> A * x
4-element Array{Float64,1}:
 3.797395092801207
 4.174505618943109
 3.212308849783316
 5.360448850163610
```

Division symbols

There are two operators allowing to divide in Julia:

- The right division represented by the symbol / (slash)

Division symbols

There are two operators allowing to divide in Julia:

- The right division represented by the symbol / (slash)
- The left division represented by the symbol \ (backslash)

Right division

Left division

```
julia> A
3×3 Array{Int64,2}:
  3   7  2
 -2   2  5
 -9  -5  8

julia> b
3-element Array{Int64,1}:
 36
 21
 -6
```

```
julia> x_ = A \ b
3-element Array{Float64,1}:
 3.9999999999999876
 2.000000000000008
 4.999999999999991

julia> x__ = inv(A) * b
3-element Array{Float64,1}:
 3.999999999999986
 1.9999999999999956
 4.999999999999998
```

```
julia> b \approx<tab> A * x_
true

julia> b \approx<tab> A * x__
true
```