实验题目5 高斯(Gauss)列主元消去法

代码实现

In [39]:
```
using Printf
using LinearAlgebra
```

In [40]:
```
# from: https://stackoverflow.com/questions/58667332/is-there-a-way-to-swap-columns-
function swapcols!(X::AbstractMatrix, i::Integer, j::Integer)
    @inbounds for k = 1:size(X, 1)
        X[k, i], X[k, j] = X[k, j], X[k, i]
    end
end
# from: https://discourse.julialang.org/t/swap-cols-rows-of-a-matrix/47904/9
function _swapcol!(x, i, j)
    for k in axes(x, 1)  # <- give dimension as input to axes function
        x[k, i], x[k, j] = x[k, j], x[k, i]
    end
end
```

```
_swapcol! (generic function with 1 method)
```

In [41]:
```
function swaprows!(X::AbstractMatrix, i::Integer, j::Integer)
    @inbounds for k = 1:size(X, 2)
        X[i, k], X[j, k] = X[j, k], X[i, k]
    end
end
```

```
swaprows! (generic function with 1 method)
```

In [42]:
```
# https://stackoverflow.com/questions/45396685/what-does-an-exclamation-mark-mean-af
# https://people.richland.edu/james/lecture/m116/matrices/pivot.html
function pivoting!(A::Matrix{Float64}, k::Integer, n::Integer)
    val, idx = findmax(A[k:n, k])
    idx += k - 1  # index must add previous length that omitted by slice operator
    return val, idx
end
function pivoting!(A::Matrix{Float64}, b::Vector{Float64}, k::Integer, n::Int
    s = [maximum(A[i, k:n]) for i in k:n]
    if 0 in s
        println("Cannot solve a singular matrix!")
        return
    end
    if implicit
        val, idx = findmax(A[k:n, k] ./ s[1:n-k+1])
    else
        A[k:n, k:n] = A[k:n, k:n] ./ s
        b[k:n] = b[k:n] ./ s
        val, idx = findmax(A[k:n, k])
    end
    idx += k - 1  # index must add previous length that omitted by slice operator
    return val, idx
end
```

```
pivoting! (generic function with 2 methods)
```

In [43]:
```
# Gauss列主元消去法
# Todo: modify it using . operator
function gauss(n, A::Matrix{Float64}, b::Vector{Float64})
    for k = 1:n-1
```

```
        # select pivot in columns
        val, idx = pivoting!(A, k, n)
        if val == 0
            println("Cannot solve a singular matrix!")
            return
        end
        # swap rows
        if idx != k
            swaprows!(A, idx, k)
            b[idx], b[k] = b[k], b[idx]
        end
        # elimination
        for i = k+1:n
            m = A[i, k] / A[k, k]
            A[i, :] -= A[k, :] * m
            b[i] -= b[k] * m
        end
    end
    if A[n, n] == 0
        println("Cannot solve a singular matrix!")
        return
    end
    # https://stackoverflow.com/questions/62142717/julia-quick-way-to-initialise-an-
    x = similar(b, Float64)
    x[n] = b[n] / A[n, n]
    for k = n-1:-1:1  # the usage of reverse sequence
        x[k] = (b[k] - dot(A[k, k+1:n], x[k+1:n])) / A[k, k]  # something really an
    end
    x
end
```

```
gauss (generic function with 2 methods)
```

In [44]:

```
# Gauss列主元消去法
# Todo: modify it using . operator
function gauss(n, A::Matrix{Float64}, b::Vector{Float64}, implicit::Bool)
    for k = 1:n-1
        # select pivot in columns
        val, idx = pivoting!(A, b, k, n, implicit)
        if val == 0
            println("Cannot solve a singular matrix!")
            return
        end
        # swap rows
        if idx != k
            swaprows!(A, idx, k)
            b[idx], b[k] = b[k], b[idx]
        end
        # elimination
        for i = k+1:n
            m = A[i, k] / A[k, k]
            A[i, :] -= A[k, :] * m
            b[i] -= b[k] * m
        end
    end
    if A[n, n] == 0
        println("Cannot solve a singular matrix!")
        return
    end
    # https://stackoverflow.com/questions/62142717/julia-quick-way-to-initialise-an-
    x = similar(b, Float64)
    x[n] = b[n] / A[n, n]
    for k = n-1:-1:1  # the usage of reverse sequence
        x[k] = (b[k] - dot(A[k, k+1:n], x[k+1:n])) / A[k, k]  # something really an
```

```
    end
    x
end
```

```
gauss (generic function with 2 methods)
```

测试代码

Test 1 - Correctness

In [45]:
```
# test random result of standard library
# test pass
for i in 1:5
    M = rand(300, 300)
    v = rand(300)
    A, b = copy(M), copy(v)  # ? Todo: move this line into try block will extend com
    try
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b), 2))\t")

        A, b = copy(M), copy(v)
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b, false), 2))\t")  # impli

        A, b = copy(M), copy(v)
        @time print("$(norm(A \ b - gauss(size(A, 1), A, b, true), 2))\t")  # implic
    catch SigularException
        println("Cannot solve a singular matrix!")
    end
    println()
end
```

```
9.318988586837023e-13    0.275640 seconds (278.72 k allocations: 444.834 MiB, 14.4
4% gc time, 32.70% compilation time)
1.4865593573213545e-12   0.493795 seconds (475.89 k allocations: 665.127 MiB, 10.6
8% gc time, 47.91% compilation time)
1.893917465969191e-12    0.213457 seconds (227.26 k allocations: 515.581 MiB, 16.8
5% gc time)

3.5673692065171883e-11   0.160950 seconds (180.32 k allocations: 439.810 MiB, 19.1
3% gc time)
1.1903308858224424e-10   0.241354 seconds (228.37 k allocations: 653.650 MiB, 13.2
0% gc time)
7.475839084501319e-11    0.198274 seconds (227.26 k allocations: 515.581 MiB, 15.8
1% gc time)

2.484235319254942e-13    0.155229 seconds (180.31 k allocations: 439.810 MiB, 18.2
5% gc time)
2.3602954910384214e-13   0.244467 seconds (228.37 k allocations: 653.650 MiB, 15.8
0% gc time)
2.6888000866323927e-13   0.200813 seconds (227.26 k allocations: 515.581 MiB, 15.3
3% gc time)

1.981893077413214e-12    0.157618 seconds (180.31 k allocations: 439.810 MiB, 17.8
1% gc time)
2.6310099338685277e-12   0.235614 seconds (228.37 k allocations: 653.650 MiB, 15.0
1% gc time)
3.4283323739836462e-12   0.207191 seconds (227.26 k allocations: 515.581 MiB, 15.0
8% gc time)

7.446480464646344e-12    0.171749 seconds (180.32 k allocations: 439.810 MiB, 18.8
0% gc time)
9.250087276929403e-12    0.412266 seconds (228.37 k allocations: 653.650 MiB, 45.6
6% gc time)
9.312773369641041e-12    0.202125 seconds (227.26 k allocations: 515.581 MiB, 17.1
1% gc time)
```

Test 2 - Performance

In [46]:

```
# x = [@elapsed(rand(i,i)\rand(i)) for i = 10:15]
# for i = 10:15
#     A, b = rand(i,i), rand(i)
#     @time A\b
# end
# # compute moving average
# # https://stackoverflow.com/questions/28820904/how-to-efficiently-compute-average-
# # n=1;
# # curAvg = 0;
# # loop{
# #   curAvg = curAvg + (newNum - curAvg)/n;
# #   n++;
# # }
# n = 2000
# curAvg = zeros(n)
# # curAvg[i] = 0
# for i = 2:n
#     curAvg[i] = curAvg[i-1] + (@elapsed(rand(i,i)\rand(i)) - curAvg[i-1])/(i-1)
# end
# plot(curAvg)

# function ma(n)
#     curAvg = zeros(n)
#     # curAvg[i] = 0
#     for i = 2:n
```

```
#         curAvg[i] = curAvg[i-1] + (@elapsed(rand(i,i)\rand(i)) - curAvg[i-1])/(i-1
#     end
#     plot(curAvg)
# end
```

In [47]:
```
# test random result of standard library
# test pass
for i in 1:5
    M = rand(300, 300)
    v = rand(300)
    A, b = copy(M), copy(v)
    try
        @time A \ b

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b)

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b, false)  # implicit=false

        A, b = copy(M), copy(v)
        @time gauss(size(A, 1), A, b, true)  # implicit=true
    catch SigularException
        println("Cannot solve a singular matrix!")
    end
    println()
end
```

```
  0.005614 seconds (4 allocations: 708.172 KiB)
  0.157441 seconds (180.30 k allocations: 439.115 MiB, 18.55% gc time)
  0.230287 seconds (228.35 k allocations: 652.955 MiB, 14.68% gc time)
  0.192803 seconds (227.24 k allocations: 514.886 MiB, 18.65% gc time)

  0.006017 seconds (4 allocations: 708.172 KiB)
  0.147328 seconds (180.30 k allocations: 439.115 MiB, 18.96% gc time)
  0.228614 seconds (228.35 k allocations: 652.955 MiB, 15.55% gc time)
  0.197291 seconds (227.24 k allocations: 514.886 MiB, 17.19% gc time)

  0.005568 seconds (4 allocations: 708.172 KiB)
  0.159448 seconds (180.30 k allocations: 439.115 MiB, 19.94% gc time)
  0.226598 seconds (228.35 k allocations: 652.955 MiB, 14.99% gc time)
  0.194048 seconds (227.24 k allocations: 514.886 MiB, 14.52% gc time)

  0.005533 seconds (4 allocations: 708.172 KiB)
  0.153294 seconds (180.30 k allocations: 439.115 MiB, 19.95% gc time)
  0.223405 seconds (228.35 k allocations: 652.955 MiB, 14.09% gc time)
  0.194211 seconds (227.24 k allocations: 514.886 MiB, 15.78% gc time)

  0.006003 seconds (4 allocations: 708.172 KiB)
  0.146435 seconds (180.30 k allocations: 439.115 MiB, 19.11% gc time)
  0.225317 seconds (228.35 k allocations: 652.955 MiB, 15.15% gc time)
  0.194285 seconds (227.24 k allocations: 514.886 MiB, 17.41% gc time)
```

Test 3 - Special Matrix

Bug: 处理上三角矩阵计算有异常

Todo: 应当增加对于特殊矩阵的测试，待测试列表同Julia Special Matrix:

- [ ] Symmetric
- [ ] Hermitian

- [ ] UpperTriangular
- [ ] UnitUpperTriangular
- [ ] LowerTriangular
- [ ] UnitLowerTriangular
- [ ] UpperHessenberg
- [ ] Tridiagonal
- [ ] SymTridiagonal
- [ ] Bidiagonal
- [ ] Diagonal
- [ ] UniformScaling

基本语法为 `M = SpecialMatrix(rand(300,300))`

实验题目

问题 1

In [48]:
```
function show_result(A, b)
    n = size(A, 1)
    A = Float64.(A)
    b = Float64.(b)
    display("input matrix:")
    # https://www.geeksforgeeks.org/creating-array-with-repeated-elements-in-julia-r
    # http://www.jlhub.com/julia/manual/en/function/repeat
    display([A repeat([], inner=(n, 1)) b])
    display("result by standard library:")
    display(@time A \ b)
    display("result by my gauss method:")
    display(@time gauss(n, A, b))
    # display(@time gauss(n, A, b, false))  # implicit=false
    # display(@time gauss(n, A, b, true))  # implicit=true
end
```

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

In [49]:
```
A = [0.4096 0.1234 0.3678 0.2943
    0.2246 0.3872 0.4015 0.1129
    0.3645 0.1920 0.3781 0.0643
    0.1784 0.4002 0.2786 0.3927]
b = [1.1951; 1.1262; 0.9989; 1.2499]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 0.4096  0.1234  0.3678  0.2943  │  1.1951
 0.2246  0.3872  0.4015  0.1129  │  1.1262
 0.3645  0.192   0.3781  0.0643  │  0.9989
 0.1784  0.4002  0.2786  0.3927  │  1.2499
"result by standard library:"
4-element Vector{Float64}:
 0.9999999999999994
 1.0
 1.0000000000000004
 1.0000000000000002
"result by my gauss method:"
```

```
4-element Vector{Float64}:
 1.0000000000000027
 1.0000000000000018
 0.9999999999999971
 0.9999999999999992
  0.000012 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [50]:
```
A = [136.01 90.860 0 0
    90.860 98.810 -67.590 0
    0 -67.590 132.01 46.260
    0 0 46.260 177.17]
b = [226.87; 122.08; 110.68; 223.43]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 136.01   90.86    0.0     0.0    │  226.87
  90.86   98.81  -67.59    0.0    │  122.08
   0.0   -67.59  132.01   46.26   │  110.68
   0.0     0.0   46.26   177.17   │  223.43
"result by standard library:"
4-element Vector{Float64}:
 1.0000000000000704
 0.9999999999998946
 0.9999999999999408
 1.0000000000000155
"result by my gauss method:"
4-element Vector{Float64}:
 1.000000000000133
 0.9999999999998009
 0.999999999999888
 1.0000000000000293
  0.000010 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [51]:
```
A = [1 1/2 1/3 1/4
     1/2 1/3 1/4 1/5
     1/3 1/4 1/5 1/6
     1/4 1/5 1/6 1/7]
b = [25 / 12; 77 / 60; 57 / 60; 319 / 420]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 1.0       0.5       0.333333  0.25      │  2.08333
 0.5       0.333333  0.25      0.2       │  1.28333
 0.333333  0.25      0.2       0.166667  │  0.95
 0.25      0.2       0.166667  0.142857  │  0.759524
"result by standard library:"
4-element Vector{Float64}:
 0.9999999999999779
 1.000000000000241
 0.9999999999994366
 1.0000000000003588
"result by my gauss method:"
4-element Vector{Float64}:
 0.9999999999999927
 1.0000000000000688
 0.9999999999998556
 1.0000000000000846
  0.000009 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [52]:
```
A = [10 7 8 7
```

```
    7 5 6 5
    8 6 10 9
    7 5 9 10]
b = [32; 23; 33; 31]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 10.0  7.0   8.0   7.0  |  32.0
  7.0  5.0   6.0   5.0  |  23.0
  8.0  6.0  10.0   9.0  |  33.0
  7.0  5.0   9.0  10.0  |  31.0
"result by standard library:"
4-element Vector{Float64}:
 1.000000000000083
 0.9999999999998619
 1.000000000000035
 0.999999999999979
"result by my gauss method:"
4-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
  0.000014 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

问题 2

In [53]:
```
A = [197 305 -206 -804
     46.8 71.3 -47.4 52.0
     88.6 76.4 -10.8 802
     1.45 5.90 6.13 36.5]
b = [136; 11.7; 25.1; 6.60]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 197.0   305.0  -206.0   -804.0  |  136.0
  46.8    71.3   -47.4     52.0  |   11.7
  88.6    76.4   -10.8    802.0  |   25.1
   1.45    5.9     6.13    36.5  |    6.6
"result by standard library:"
4-element Vector{Float64}:
  0.9536791069017718
  0.32095684552110354
  1.0787080757932384
 -0.09010850953957893
"result by my gauss method:"
4-element Vector{Float64}:
  0.9536791069017717
  0.3209568455211036
  1.0787080757932384
 -0.09010850953957895
  0.000009 seconds (3 allocations: 384 bytes)
  0.000005 seconds (34 allocations: 3.031 KiB)
```

In [54]:
```
A = [0.5398 0.7161 -0.5554 -0.2982
     0.5257 0.6924 0.3565 -0.6255
     0.6465 -0.8187 -0.1872 0.1291
     0.5814 0.9400 -0.7779 -0.4042]
b = [0.2058; -0.0503; 0.1070; 0.1859]
show_result(A, b)
```

```
"input matrix:"
4×6 Matrix{Any}:
 0.5398   0.7161  -0.5554  -0.2982  |   0.2058
 0.5257   0.6924   0.3565  -0.6255  |  -0.0503
 0.6465  -0.8187  -0.1872   0.1291  |   0.107
 0.5814   0.94    -0.7779  -0.4042  |   0.1859
"result by standard library:"
4-element Vector{Float64}:
 0.5161772979585422
 0.4152194728301359
 0.10996610286788958
 1.0365392233362019
"result by my gauss method:"
4-element Vector{Float64}:
 0.5161772979585416
 0.41521947283013527
 0.10996610286788916
 1.0365392233362005
  0.000011 seconds (3 allocations: 384 bytes)
  0.000007 seconds (34 allocations: 3.031 KiB)
```

In [55]:
```
A = [10 1 2
    1 10 2
    1 1 5]
b = [13; 13; 7]
show_result(A, b)
```

```
"input matrix:"
3×5 Matrix{Any}:
 10.0   1.0  2.0  |  13.0
  1.0  10.0  2.0  |  13.0
  1.0   1.0  5.0  |   7.0
"result by standard library:"
3-element Vector{Float64}:
 1.0
 0.9999999999999998
 1.0
"result by my gauss method:"
3-element Vector{Float64}:
 1.0
 0.9999999999999998
 1.0000000000000002
  0.000014 seconds (3 allocations: 288 bytes)
  0.000006 seconds (19 allocations: 1.453 KiB)
```

In [56]:
```
A = [4 -2 -4
    -2 17 10
    -4 10 9]
b = [-2; 25; 15]
show_result(A, b)
```

```
"input matrix:"
3×5 Matrix{Any}:
  4.0  -2.0  -4.0  |  -2.0
 -2.0  17.0  10.0  |  25.0
 -4.0  10.0   9.0  |  15.0
"result by standard library:"
3-element Vector{Float64}:
 1.0
 1.0
 1.0
"result by my gauss method:"
```

```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
  0.000011 seconds (3 allocations: 288 bytes)
  0.000007 seconds (19 allocations: 1.453 KiB)
```

总结

Todo: 直接法效率评价，写一个迭代法来算，用于比较效率，库函数使用的方法

Todo: 参考资料链接放在文末

Todo: 特殊矩阵的测试代码，剩余bug修复