What's inside \?

Let's ask Julia

In [14]:
```
?\
```

```
search: \ .\
```

Out[14]:
```
\(x, y)
```

Left division operator: multiplication of `y` by the inverse of `x` on the left. Gives floating-point results for integer arguments.

```
\(A, B)
```

Matrix division using a polyalgorithm. For input matrices `A` and `B`, the result `X` is such that `A*X == B` when `A` is square. The solver that is used depends upon the structure of `A`. If `A` is upper or lower triangular (or diagonal), no factorization of `A` is required and the system is solved with either forward or backward substitution. For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt` factorization

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

What's inside \?

Let's ask Julia

In [14]:
```
?\
```

```
search: \ .\
```

Out[14]: `\(x, y)`

Left division operator: multiplication of `y` by the inverse of `x` on the left. Gives floating-point results for integer arguments.

`\(A, B)`

Matrix division using a polyalgorithm. For input matrices `A` and `B`, the result `X` is such that `A*X == B` when `A` is square. The solver that is used depends upon the structure of `A`. If `A` is upper or lower triangular (or diagonal), no factorization of `A` is required and the system is solved with either forward or backward substitution. For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt` factorization

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

```
search: \ .\
```

Out[14]: `\(x, y)`

Left division operator: multiplication of `y` by the inverse of `x` on the left. Gives floating-point results for integer arguments.

`\(A, B)`

Matrix division using a polyalgorithm. For input matrices `A` and `B`, the result `X` is such that `A*X == B` when `A` is square. The solver that is used depends upon the structure of `A`. If `A` is upper or lower triangular (or diagonal), no factorization of `A` is required and the system is solved with either forward or backward substitution. For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt` factorization does not use pivoting during the numerical factorization and therefore the procedure can fail even for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Left division operator: multiplication of `y` by the inverse of `x` on the left. Gives floating-point results for integer arguments.

`\(A, B)`

Matrix division using a polyalgorithm. For input matrices `A` and `B`, the result `X` is such that `A*X == B` when `A` is square. The solver that is used depends upon the structure of `A`. If `A` is upper or lower triangular (or diagonal), no factorization of `A` is required and the system is solved with either forward or backward substitution. For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt` factorization does not use pivoting during the numerical factorization and therefore the procedure can fail even for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

for integer arguments.

`\(A, B)`

Matrix division using a polyalgorithm. For input matrices `A` and `B`, the result `X` is such that `A*X == B` when `A` is square. The solver that is used depends upon the structure of `A`. If `A` is upper or lower triangular (or diagonal), no factorization of `A` is required and the system is solved with either forward or backward substitution. For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt` factorization does not use pivoting during the numerical factorization and therefore the procedure can fail even for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The following two calls to \ are therefore dispatching to the same method

for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]: `true`

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]: `Base.LinAlg.LU{Float64,Array{Float64,2}}`

In [4]:
```
supertype(typeof(F))
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

Let's ask differently

In [1]:
```julia
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```julia
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```julia
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```julia
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]:
\(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

for invertible matrices.

Let's ask differently

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [1]:
```julia
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```julia
F = lufact(A)
F\b == A\b
```

Out[2]: `true`

...but what is `lufact(A)` returning?

In [3]:
```julia
typeof(F)
```

Out[3]: `Base.LinAlg.LU{Float64,Array{Float64,2}}`

JuliaLang / julia

Unwatch 596 Unstar 8,293 Fork 1,915

Tree: 3c9d75391c julia / base / linalg / generic.jl

Find file Copy path

timholy Some indices generalizations for linalg/generic (#18032) 1aa1175 Aug 20, 2016

25 contributors

670 lines (573 sloc) 18.8 KB

Raw Blame History

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

## linalg.jl: Some generic Linear Algebra definitions

# For better performance when input and output are the same array
# See https://github.com/JuliaLang/julia/issues/8415#issuecomment-56608729
function generic_scale!(X::AbstractArray, s::Number)
    @simd for I in eachindex(X)
        @inbounds X[I] *= s
    end
    X
end

stractArray)
```

```
For non-triangular square matrices, an LU factorization is used.

For rectangular `A` the result is the minimum-norm least squares solution computed by a
pivoted QR factorization of `A` and a rank estimate of `A` based on the R factor.

When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt`
factorization does not use pivoting during the numerical factorization and therefore the
procedure can fail even for invertible matrices.
"""
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwise version has been deprecated long enough
# /(x::Number,A::StridedMatrix) = x*inv(A)
```

```
When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt`
factorization does not use pivoting during the numerical factorization and therefore the
procedure can fail even for invertible matrices.
"""
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwise version has been deprecated long enough
# /(x::Number,A::StridedMatrix) = x*inv(A)

cond(x::Number) = x == 0 ? Inf : 1.0
cond(x::Number, p) = cond(x)

#Skeel condition numbers
```

```
When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt`
factorization does not use pivoting during the numerical factorization and therefore the
procedure can fail even for invertible matrices.
"""
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwise version has been depreca
# /(x::Number,A::StridedMatrix) = x*inv(A)
```

```julia
When `A` is sparse, a similar polyalgorithm is used. For indefinite matrices, the `LDLt`
factorization does not use pivoting during the numerical factorization and therefore the
procedure can fail even for invertible matrices.
"""
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end
```

```julia
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwi
# /(x::Number,A::StridedMatrix) = x*inv(A)
```

```
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwi
# /(x::Number,A::StridedMatrix) = x*inv(A)

```

```
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwi
# /(x::Number,A::StridedMatrix) = x*inv(A)

```

```julia
factorization does not use pivoting during the numerical factorization and therefore the
procedure can fail even for invertible matrices.
"""
function (\)(A::AbstractMatrix, B::AbstractVecOrMat)
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
```

```
    m, n = size(A)
    if m == n
        if istril(A)
            if istriu(A)
                return Diagonal(A) \ B
            else
                return LowerTriangular(A) \ B
            end
        end
        if istriu(A)
            return UpperTriangular(A) \ B
        end
        return lufact(A) \ B
    end
    return qrfact(A,Val{true}) \ B
end

(\)(a::AbstractVector, b::AbstractArray) = reshape(a, length(a), 1) \ b
(/)(A::AbstractVecOrMat, B::AbstractVecOrMat) = (B' \ A')'
# \(A::StridedMatrix,x::Number) = inv(A)*x Should be added at some point when the old elementwi
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [1]:
```
A = randn(5,5)
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349](https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/base/linalg/generic.jl#L349)

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/base/linalg/generic.jl#L349

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```julia
b = randn(5)
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The following two calls to \ are therefore dispatching to the same method

In [2]:
```julia
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```julia
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```julia
supertype(typeof(F))
```

```
@which A\b
```

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Out[1]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

linalg/generic.jl:349

The following two calls to \ are therefore dispatching to the same method

In [2]:
```
F = lufact(A)
F\b == A\b
```

Out[2]:
```
true
```

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Out[2]: true

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/factorization.jl:37]

So Julia dispatches to a non-allocating method

...but what is `lufact(A)` returning?

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/factorization.jl:37]

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [3]:
```
typeof(F)
```
Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```
Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```
Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/factorization.jl:37

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```
Out[6]:
```
true
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [3]:
```
typeof(F)
```
Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```
Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```
Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/factorization.jl:37

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```
Out[6]:
```
true
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [3]:
```
typeof(F)
```

Out[3]:
```
Base.LinAlg.LU{Float64,Array{Float64,2}}
```

In [4]:
```
supertype(typeof(F))
```

Out[4]:
```
Factorization{Float64}
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/factorization.jl:37]

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/factorization.jl:37

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/factorization.jl:37]

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,

Waiting for github.com...

This repository | Search | Pull requests | Issues | Gist

JuliaLang / julia

Unwatch 596

Code | Issues 1,770 | Pull requests 424 | Projects 6 | Pulse | Graphs

Tree: 3c9d75391c | julia / base / linalg / **factorization.jl**

tkelman Fix backslash in A_ldiv_B! docstring

23 contributors

82 lines (64 sloc) | 2.83 KB | Raw

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

## Matrix factorizations and decompositions

```

Waiting for avatars0.githubusercontent.com...

This repository Search Pull requests Issues Gist

JuliaLang / julia

Unwatch 596

Code Issues 1,770 Pull requests 424 Projects 6 Pulse Graphs

Tree: 3c9d75391c

julia / base / linalg / factorization.jl

tkelman Fix backslash in A_ldiv_B! docstring

23 contributors

82 lines (64 sloc) 2.83 KB Raw

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

## Matrix factorizations and decompositions

```

Waiting for avatars0.githubusercontent.com...

```julia
end

for (f1, f2) in ((:\, :A_ldiv_B!),
                 (:Ac_ldiv_B, :Ac_ldiv_B!),
                 (:At_ldiv_B, :At_ldiv_B!))
    @eval begin
        function $f1(F::Factorization, B::AbstractVecOrMat)
            TFB = typeof(one(eltype(F)) / one(eltype(B)))
            BB = similar(B, TFB, size(B))
            copy!(BB, B)
            $f2(convert(Factorization{TFB}, F), BB)
        end
    end
end

# support the same 3-arg idiom as in our other in-place A_*_B functions:
for f in (:A_ldiv_B!, :Ac_ldiv_B!, :At_ldiv_B!)
    @eval $f(Y::AbstractVecOrMat, A::Factorization, B::AbstractVecOrMat) =
        $f(A, copy!(Y, B))
end

"""
```

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/factorization.jl:37]

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2,
(A::**Base.LinAlg.LU{T,S}**,

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

What happens after the matrix has been factorized? Let's ask Julia

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/factorization.jl:37

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [5]:
```
@which F\b
```

Out[5]: \(F::**Factorization**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/factorization.jl:37

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]: true

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2,
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverse**
at linalg/lu.jl:176

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```
Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```
Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverse**
at linalg/lu.jl:176

What if the matrix is diagonal?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

So Julia dispatches to a non-allocating method

In [6]:
```
A_ldiv_B!(F, copy(b)) == A\b
```

Out[6]:
```
true
```

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Out[6]: true

We can continue the search

In [7]:
```
@which A_ldiv_B!(F, copy(b))
```

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
                     0.0       0.18875  0.0
```

https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75f1187ce77b2d5effc8/base/linalg/lu.jl#L176

This repository Search Pull requests Issues Gist

JuliaLang / julia

Unwatch 596

Code Issues 1,770 Pull requests 423 Projects 6 Pulse Graphs

Tree: 3c9d75391c julia / base / linalg / lu.jl

tkelman Revert "Towards #12251 part 2, replace all full(X) calls with convert...

18 contributors

480 lines (435 sloc) 15.8 KB Raw

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

####################
# LU Factorization #
```

This repository Search Pull requests Issues Gist

JuliaLang / julia

Unwatch 596

Code Issues 1,770 Pull requests 423 Projects 6 Pulse Graphs

Tree: 3c9d75391c julia / base / linalg / lu.jl

tkelman Revert "Towards #12251 part 2, replace all full(X) calls with convert...

18 contributors

480 lines (435 sloc) 15.8 KB Raw

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

####################
# LU Factorization #
```

Waiting for avatars2.githubusercontent.com...

```julia
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```julia
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
tor) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipi
rix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipi

::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
ctor) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ip
trix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ip

idedVecOrMat{T}) = At_ldiv_B!(F, B)
 B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
ctor) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ip
trix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ip

::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
tor) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv,
rix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv,

::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
ctor) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv
trix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv

idedVecOrMat{T}) = At_ldiv_B!(F, B)
 B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
ctor) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv
trix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv

::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
anspose(B))
```

```
dVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b)
ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1

VecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
t_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ip
t_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ip

Mat{T}) = At_ldiv_B!(F, B)
edVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
c_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ip
c_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ip

VecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
)
```

```
invperm(F[:p]))]




StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ip
ix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.f
ix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.f

StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipi
rix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(
rix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(

dedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.i
rix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(
rix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(

StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipi
```

```
invperm(F[:p]))]




StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ip
ix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.f
ix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.f

StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipi
rix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(
rix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(

dedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.i
rix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(
rix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(

StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipi
dVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
 m)[:,invperm(F[:p])]

r(d))



t, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors
dMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangul
dMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangul

at,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors,
edMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTrian
edMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTrian

:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
plex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factor
edMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTrian
edMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTrian

at,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors,
StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors),
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors),

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular L
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.fac
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.fac

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.fac
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.fac

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular L
```

```
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
:P
eye(T, m)[:,invperm(F[:p])]

eyError(d))



asFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A
StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLower
StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLower

lasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.
:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(Up
:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(Up

eal,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
lasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C',
:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(Up
:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(Up

lasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.
```

```
:P
eye(T, m)[:,invperm(F[:p])]

eyError(d))



asFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A
StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLower
StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLower

.asFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.
 StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(Up
 StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(Up

eal,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
.asComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C',
 StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(Up
 StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(Up

.asFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.
```

```
ı)[:,invperm(F[:p])]

 d))



 S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factor
latrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangu
latrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangu

:,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors
lMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTria
lMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTria

;tridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
.ex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.facto
lMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTria
lMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTria

:,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors
```

```
S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
dVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(
dMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(

}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
edVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invpe
edMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invpe

:StridedVecOrMat{T}) = At_ldiv_B!(F, B)
,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
edVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invpe
edMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invpe

}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.inf
  transpose(B))
```

```
S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
dVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(
dMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(

}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
edVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invpe
edMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invpe

:StridedVecOrMat{T}) = At_ldiv_B!(F, B)
,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
edVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invpe
edMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invpe

}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.in
  transpose(B))
```

```
at{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b))]))
(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1)),:]))

t{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, len
B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, siz

 = At_ldiv_B!(F, B)
Mat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, len
B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, siz

t{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
t{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b))]))
UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1)),:]))

{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, len
!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, siz

= At_ldiv_B!(F, B)
at{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, len
!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, siz

{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
perTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b))]))
perTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1)),:]))

}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, lengt
UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, size(

At_ldiv_B!(F, B)
{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv, lengt
UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv, size(

}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
cOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
v_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b))]
v_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1))

OrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv
ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv

{T}) = At_ldiv_B!(F, B)
ecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ipiv
ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ipiv

OrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
VecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
div_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b)
div_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B, 1

ecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ip
_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ip

at{T}) = At_ldiv_B!(F, B)
dVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.ip
_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.ip

ecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
dVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, length(b
ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, size(B,

VecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
t_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.i
t_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.i

Mat{T}) = At_ldiv_B!(F, B)
edVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
c_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2perm(A.i
c_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2perm(A.i

VecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
tridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
 = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, len
 = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, siz

ridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2per
) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2per

VecOrMat{T}) = At_ldiv_B!(F, B)
StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2per
) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2per

ridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
ose(B))
```

```
StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2perm(A.ipiv, le
) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2perm(A.ipiv, si

tridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
r) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2pe
x) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2pe

dVecOrMat{T}) = At_ldiv_B!(F, B)
:StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
r) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[invperm(ipiv2pe
x) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[invperm(ipiv2pe

tridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A.info
```

```
LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2pe
StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2pe

LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
:StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[in
:StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[in

S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
:StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[in
:StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[in

LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A
```

```
LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ipiv2pe
StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ipiv2pe

LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
:StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b))[in
:StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B))[in

S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.info
:StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b))[in
:StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B))[in

LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(B)) A
```

```
))]




k}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('N', A.factors, A.ipiv, B) A.info
}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), b[ip
}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.factors), A_ldiv_B!(UnitLowerTriangular(A.factors), B[ip

k}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, B) A.info
S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), b
S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A.factors), At_ldiv_B!(UpperTriangular(A.factors), B

:LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
rix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('C', A.factors, A.ipiv, B) A.inf
S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), b
S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A.factors), Ac_ldiv_B!(UpperTriangular(A.factors), B

k}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingular LAPACK.getrs!('T', A.factors, A.ipiv, transpose(
 At_ldiv_B(A, transpose(B))
```

```
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```
    elseif d == :P
        return eye(T, m)[:,invperm(F[:p])]
    else
        throw(KeyError(d))
    end
end

A_ldiv_B!{T<:BlasFloat, S<:StridedMatrix}(A::LU{T, S}, B::StridedVecOrMat{T}) = @assertnonsingu
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = A_ldiv_B!(UpperTriangular(A.facto
A_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = A_ldiv_B!(UpperTriangular(A.facto

At_ldiv_B!{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))
```

```julia
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = At_ldiv_B!(UnitLowerTriangular(A
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))

Ac_ldiv_Bc{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_Bc(A::LU, B::StridedVecOrMat) = Ac_ldiv_B(A, ctranspose(B))

function det{T,S}(A::LU{T,S})
    n = checksquare(A)
    A.info > 0 && return zero(typeof(A.factors[1]))
    P = one(T)
    c = 0
    @inbounds for i = 1:n
        P *= A.factors[i,i]
        if A.ipiv[i] != i
```

Chrome File Edit View History Bookmarks People Window Help

CSE17 - What's inside back... | julia/lu.jl at 3c9d75391c72d... | julia/factorization.jl at 3c9d7... | julia/generic.jl at 3c9d75391...

GitHub, Inc. [US] https://github.com/JuliaLang/julia/blob/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/base/linalg/lu.jl#L176

```
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))

Ac_ldiv_Bc{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_Bc(A::LU, B::StridedVecOrMat) = Ac_ldiv_B(A, ctranspose(B))

function det{T,S}(A::LU{T,S})
    n = checksquare(A)
    A.info > 0 && return zero(typeof(A.factors[1]))
    P = one(T)
    c = 0
    @inbounds for i = 1:n
        P *= A.factors[i,i]
```

```julia
At_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = At_ldiv_B!(UnitLowerTriangular(A

Ac_ldiv_B!{T<:Real,S<:StridedMatrix}(F::LU{T,S}, B::StridedVecOrMat{T}) = At_ldiv_B!(F, B)
Ac_ldiv_B!{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, b::StridedVector) = Ac_ldiv_B!(UnitLowerTriangular(A
Ac_ldiv_B!{T,S<:StridedMatrix}(A::LU{T,S}, B::StridedMatrix) = Ac_ldiv_B!(UnitLowerTriangular(A

At_ldiv_Bt{T<:BlasFloat,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsingul
At_ldiv_Bt(A::LU, B::StridedVecOrMat) = At_ldiv_B(A, transpose(B))

Ac_ldiv_Bc{T<:BlasComplex,S<:StridedMatrix}(A::LU{T,S}, B::StridedVecOrMat{T}) = @assertnonsing
Ac_ldiv_Bc(A::LU, B::StridedVecOrMat) = Ac_ldiv_B(A, ctranspose(B))

function det{T,S}(A::LU{T,S})
    n = checksquare(A)
    A.info > 0 && return zero(typeof(A.factors[1]))
    P = one(T)
    c = 0
    @inbounds for i = 1:n
        P *= A.factors[i,i]
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [7]: @which A_ldiv_B!(F, copy(b))

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
 0.0       0.0       0.0       0.18875  0.0
 0.0       0.0       0.0       0.0      0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
 0.0       0.0       0.0       0.18875  0.0
 0.0       0.0       0.0       0.0      0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at
linalg/generic.jl:349

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
 0.0       0.0       0.0       0.18875  0.0
 0.0       0.0       0.0       0.0      0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [9]:
```julia
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```julia
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```julia
fieldnames(D)
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]: 1-element Array{Symbol,1}:

linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅ 
  ⋅        0.564776   ⋅         ⋅         ⋅ 
  ⋅         ⋅        0.998446   ⋅         ⋅ 
  ⋅         ⋅         ⋅        0.183225   ⋅ 
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
 0.564776
 0.998446
```

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅ 
  ⋅        0.564776   ⋅         ⋅         ⋅ 
  ⋅         ⋅        0.998446   ⋅         ⋅ 
  ⋅         ⋅         ⋅        0.183225   ⋅ 
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
 0.564776
 0.998446
 0.183225
 0.890447
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```
 ⋅   0.564776   ⋅          ⋅          ⋅
 ⋅    ⋅        0.998446    ⋅          ⋅
 ⋅    ⋅         ⋅         0.183225    ⋅
 ⋅    ⋅         ⋅          ⋅         0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
 0.564776
 0.998446
 0.183225
 0.890447
```

In [13]:
```
@which D\b
```

linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```
 0.0       0.0       0.441463  0.0       0.0
 0.0       0.0       0.0       0.18875   0.0
 0.0       0.0       0.0       0.0       0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```
 0.0         0.0         0.0         0.0         0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅          ⋅          ⋅          ⋅ 
  ⋅        0.564776    ⋅          ⋅          ⋅ 
  ⋅         ⋅         0.998446    ⋅          ⋅ 
  ⋅         ⋅          ⋅         0.183225    ⋅ 
  ⋅         ⋅          ⋅          ⋅         0.890447
```

What's inside?

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

0.0 0.0 0.0 0.0 0.821599

In [9]:
```julia
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The good diagonal matrix is

In [10]:
```julia
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```julia
fieldnames(D)
```

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
```

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅      
  ⋅        0.564776   ⋅         ⋅         ⋅      
  ⋅         ⋅        0.998446   ⋅         ⋅      
  ⋅         ⋅         ⋅        0.183225   ⋅      
  ⋅         ⋅         ⋅         ⋅        0.890447
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```
 0.0       0.389069  0.0       0.0       0.0
 0.0       0.0       0.441463  0.0       0.0
 0.0       0.0       0.0       0.18875   0.0
 0.0       0.0       0.0       0.0       0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
  ⋅         ⋅         ⋅         ⋅        0.890447
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
 0.0       0.0       0.0       0.18875  0.0
 0.0       0.0       0.0       0.0      0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

Out[7]: A_ldiv_B!
{T<:Union{Complex{Float32},Complex{Float64},Float32,Float64},S<:Union{Base.ReshapedArray{T,2,
(A::**Base.LinAlg.LU{T,S}**,
B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInvers**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [8]:
```
D = diagm(rand(5))
```

Out[8]:
```
5×5 Array{Float64,2}:
 0.454292  0.0       0.0       0.0      0.0
 0.0       0.389069  0.0       0.0      0.0
 0.0       0.0       0.441463  0.0      0.0
 0.0       0.0       0.0       0.18875  0.0
 0.0       0.0       0.0       0.0      0.821399
```

In [9]:
```
@which D\b
```

Out[9]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

B::**Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverse**
at linalg/lu.jl:176

What if the matrix is diagonal?

In [16]:
```
D = diagm(rand(5))
```

Out[16]:
```
5×5 Array{Float64,2}:
 0.43952  0.0       0.0         0.0      0.0
 0.0      0.562064  0.0         0.0      0.0
 0.0      0.0       0.00455724  0.0      0.0
 0.0      0.0       0.0         0.45622  0.0
 0.0      0.0       0.0         0.0      0.118231
```

In [17]:
```
@which D\b
```

Out[17]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

What if the matrix is diagonal?

In [16]:
```
D = diagm(rand(5))
```

Out[16]:
```
5×5 Array{Float64,2}:
 0.43952   0.0       0.0         0.0      0.0
 0.0       0.562064  0.0         0.0      0.0
 0.0       0.0       0.00455724  0.0      0.0
 0.0       0.0       0.0         0.45622  0.0
 0.0       0.0       0.0         0.0      0.118231
```

In [17]:
```
@which D\b
```

Out[17]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131  ⋅         ⋅         ⋅         ⋅
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

In [16]:
```
D = diagm(rand(5))
```

Out[16]:
```
5×5 Array{Float64,2}:
 0.43952  0.0       0.0         0.0      0.0
 0.0      0.562064  0.0         0.0      0.0
 0.0      0.0       0.00455724  0.0      0.0
 0.0      0.0       0.0         0.45622  0.0
 0.0      0.0       0.0         0.0      0.118231
```

In [17]:
```
@which D\b
```

Out[17]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at linalg/generic.jl:349

The good diagonal matrix is

In [10]:
```
D = Diagonal(rand(5))
```

Out[10]:
```
5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

```
Out[16]: 5×5 Array{Float64,2}:
 0.43952  0.0       0.0         0.0      0.0
 0.0      0.562064  0.0         0.0      0.0
 0.0      0.0       0.00455724  0.0      0.0
 0.0      0.0       0.0         0.45622  0.0
 0.0      0.0       0.0         0.0      0.118231
```

```
In [17]: @which D\b
```

Out[17]: \(A::**AbstractArray{T<:Any,2}**, B::**Union{AbstractArray{T<:Any,1},AbstractArray{T<:Any,2}}**) at [linalg/generic.jl:349]

The good diagonal matrix is

```
In [10]: D = Diagonal(rand(5))
```

```
Out[10]: 5×5 Diagonal{Float64}:
 0.229131   ⋅         ⋅         ⋅         ⋅
  ⋅        0.564776   ⋅         ⋅         ⋅
  ⋅         ⋅        0.998446   ⋅         ⋅
  ⋅         ⋅         ⋅        0.183225   ⋅
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

The good diagonal matrix is

In [18]:
```
D = Diagonal(rand(5))
```

Out[18]:
```
5×5 Diagonal{Float64}:
 0.91572   ⋅         ⋅         ⋅         ⋅
  ⋅       0.922452   ⋅         ⋅         ⋅
  ⋅        ⋅        0.940005   ⋅         ⋅
  ⋅        ⋅         ⋅        0.610354   ⋅
  ⋅        ⋅         ⋅         ⋅        0.411102
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
```

File Edit View Insert Cell Kernel Widgets Help Julia 0.5.0

The good diagonal matrix is

In [18]:
```
D = Diagonal(rand(5))
```

Out[18]:
```
5×5 Diagonal{Float64}:
 0.91572   ⋅         ⋅         ⋅         ⋅ 
  ⋅        0.922452  ⋅         ⋅         ⋅ 
  ⋅         ⋅        0.940005  ⋅         ⋅ 
  ⋅         ⋅         ⋅        0.610354  ⋅ 
  ⋅         ⋅         ⋅         ⋅        0.411102
```

What's inside?

In [11]:
```
fieldnames(D)
```

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
 0.564776
```

File Edit View Insert Cell Kernel Widgets Help | Julia 0.5.0

Out[11]:
```
1-element Array{Symbol,1}:
 :diag
```

In [12]:
```
D.diag
```

Out[12]:
```
5-element Array{Float64,1}:
 0.229131
 0.564776
 0.998446
 0.183225
 0.890447
```

In [13]:
```
@which D\b
```

Out[13]: \(D::**Diagonal**, b::**AbstractArray{T<:Any,1}**) at linalg/diagonal.jl:267

Waiting for github.com...

JuliaLang / julia

Unwatch 596

Tree: 3c9d75391c julia / base / linalg / diagonal.jl

tkelman Revert "Towards #12251 part 2, replace all full(X) calls with convert...

23 contributors

323 lines (284 sloc) 10.1 KB Raw

```
# This file is a part of Julia. License is MIT: http://julialang.org/license

## Diagonal matrices

```

Waiting for github.com...

```julia
        end
    end
    return B
end
(\)(D::Diagonal, A::AbstractMatrix) = D.diag .\ A
(\)(D::Diagonal, b::AbstractVector) = D.diag .\ b
(\)(Da::Diagonal, Db::Diagonal) = Diagonal(Db.diag ./ Da.diag)

function inv{T}(D::Diagonal{T})
    Di = similar(D.diag, typeof(inv(zero(T))))
    for i = 1:length(D.diag)
        if D.diag[i] == zero(T)
            throw(SingularException(i))
        end
        Di[i] = inv(D.diag[i])
    end
    Diagonal(Di)
end

function pinv{T}(D::Diagonal{T})
```

```
        end
    end
    return B
end
(\)(D::Diagonal, A::AbstractMatrix) = D.diag .\ A
(\)(D::Diagonal, b::AbstractVector) = D.diag .\ b
(\)(Da::Diagonal, Db::Diagonal) = Diagonal(Db.diag ./ Da.diag)

function inv{T}(D::Diagonal{T})
    Di = similar(D.diag, typeof(inv(zero(T))))
    for i = 1:length(D.diag)
        if D.diag[i] == zero(T)
            throw(SingularException(i))
        end
        Di[i] = inv(D.diag[i])
    end
    Diagonal(Di)
end

function pinv{T}(D::Diagonal{T})
```

```julia
            end
            B[i,j] /= di
        end
    end
    return B
end
(\)(D::Diagonal, A::AbstractMatrix) = D.diag .\ A
(\)(D::Diagonal, b::AbstractVector) = D.diag .\ b
(\)(Da::Diagonal, Db::Diagonal) = Diagonal(Db.diag ./ Da.diag)

function inv{T}(D::Diagonal{T})
    Di = similar(D.diag, typeof(inv(zero(T))))
    for i = 1:length(D.diag)
        if D.diag[i] == zero(T)
            throw(SingularException(i))
        end
        Di[i] = inv(D.diag[i])
    end
    Diagonal(Di)
end

function pinv{T}(D::Diagonal{T})
```

```julia
        if d == zero(T)
            throw(SingularException(i))
        end
        d⁻¹ = inv(d)
        for j=1:size(V,2)
            @inbounds V[i,j] *= d⁻¹
        end
    end
    V
end

conj(D::Diagonal) = Diagonal(conj(D.diag))
transpose(D::Diagonal) = D
ctranspose(D::Diagonal) = conj(D)

diag(D::Diagonal) = D.diag
trace(D::Diagonal) = sum(D.diag)
det(D::Diagonal) = prod(D.diag)
logdet{T<:Real}(D::Diagonal{T}) = sum(log(D.diag))
function logdet{T<:Complex}(D::Diagonal{T}) #Make sure branch cut is correct
    x = sum(log(D.diag))
    -pi<imag(x)<pi ? x : real(x)+(mod2pi(imag(x)+pi)-pi)*im
```

```
        throw(DimensionMismatch("diagonal matrix is $(length(D.diag)) by $(length(D.diag)) but
    end
    for i=1:length(D.diag)
        d = D.diag[i]
        if d == zero(T)
            throw(SingularException(i))
        end
        d⁻¹ = inv(d)
        for j=1:size(V,2)
            @inbounds V[i,j] *= d⁻¹
        end
    end
    V
end

conj(D::Diagonal) = Diagonal(conj(D.diag))
transpose(D::Diagonal) = D
ctranspose(D::Diagonal) = conj(D)

diag(D::Diagonal) = D.diag
trace(D::Diagonal) = sum(D.diag)
det(D::Diagonal) = prod(D.diag)
```

```
                throw(SingularException(i))
            end
            v[i] *= inv(d)
        end
        v
end
function A_ldiv_B!{T}(D::Diagonal{T}, V::AbstractMatrix{T})
    if size(V,1) != length(D.diag)
        throw(DimensionMismatch("diagonal matrix is $(length(D.diag)) by $(length(D.diag)) but
    end
    for i=1:length(D.diag)
        d = D.diag[i]
        if d == zero(T)
            throw(SingularException(i))
        end
        d⁻¹ = inv(d)
        for j=1:size(V,2)
            @inbounds V[i,j] *= d⁻¹
        end
    end
    V
```

```
Ac_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(Ac_mul_B!, Tuple{Diagonal,Diagonal}))
A_mul_B!(A::Base.LinAlg.QRPackedQ, D::Diagonal) = throw(MethodError(A_mul_B!, Tuple{Diagonal,Di
A_mul_B!(A::Diagonal,B::AbstractMatrix)  = scale!(A.diag,B)
At_mul_B!(A::Diagonal,B::AbstractMatrix) = scale!(A.diag,B)
Ac_mul_B!(A::Diagonal,B::AbstractMatrix) = scale!(conj(A.diag),B)
A_mul_B!(A::AbstractMatrix,B::Diagonal)  = scale!(A,B.diag)
A_mul_Bt!(A::AbstractMatrix,B::Diagonal) = scale!(A,B.diag)
A_mul_Bc!(A::AbstractMatrix,B::Diagonal) = scale!(A,conj(B.diag))

/(Da::Diagonal, Db::Diagonal) = Diagonal(Da.diag ./ Db.diag )
function A_ldiv_B!{T}(D::Diagonal{T}, v::AbstractVector{T})
    if length(v) != length(D.diag)
        throw(DimensionMismatch("diagonal matrix is $(length(D.diag)) by $(length(D.diag)) but
    end
    for i=1:length(D.diag)
        d = D.diag[i]
        if d == zero(T)
            throw(SingularException(i))
        end
        v[i] *= inv(d)
    end
    v
```

```
Ac_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(Ac_mul_B!, Tuple{Diagonal,Diagonal}))
A_mul_B!(A::Base.LinAlg.QRPackedQ, D::Diagonal) = throw(MethodError(A_mul_B!, Tuple{Diagonal,Di
A_mul_B!(A::Diagonal,B::AbstractMatrix)  = scale!(A.diag,B)
At_mul_B!(A::Diagonal,B::AbstractMatrix) = scale!(A.diag,B)
Ac_mul_B!(A::Diagonal,B::AbstractMatrix) = scale!(conj(A.diag),B)
A_mul_B!(A::AbstractMatrix,B::Diagonal)  = scale!(A,B.diag)
A_mul_Bt!(A::AbstractMatrix,B::Diagonal) = scale!(A,B.diag)
A_mul_Bc!(A::AbstractMatrix,B::Diagonal) = scale!(A,conj(B.diag))

/(Da::Diagonal, Db::Diagonal) = Diagonal(Da.diag ./ Db.diag )
function A_ldiv_B!{T}(D::Diagonal{T}, v::AbstractVector{T})
    if length(v) != length(D.diag)
        throw(DimensionMismatch("diagonal matrix is $(length(D.diag)) by $(length(D.diag)) but 
    end
    for i=1:length(D.diag)
        d = D.diag[i]
        if d == zero(T)
            throw(SingularException(i))
        end
        v[i] *= inv(d)
    end
    v
```

```
end

A_mul_Bc(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, ctranspose(B))
A_mul_Bc(D::Diagonal, Q::Union{Base.LinAlg.QRCompactWYQ,Base.LinAlg.QRPackedQ}) = A_mul_Bc!(Arr
function A_mul_Bc(D::Diagonal, A::AbstractMatrix)
    Ac = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    ctranspose!(Ac, A)
    A_mul_B!(D, Ac)
end

A_mul_Bt(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, transpose(B))
function A_mul_Bt(D::Diagonal, A::AbstractMatrix)
    At = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    transpose!(At, A)
    A_mul_B!(D, At)
end

A_mul_B!(A::Diagonal,B::Diagonal)  = throw(MethodError(A_mul_B!, Tuple{Diagonal,Diagonal}))
At_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(At_mul_B!, Tuple{Diagonal,Diagonal}))
Ac_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(Ac_mul_B!, Tuple{Diagonal,Diagonal}))
A_mul_B!(A::Base.LinAlg.QRPackedQ, D::Diagonal) = throw(MethodError(A_mul_B!, Tuple{Diagonal,Di
```

```
    A_mul_B!(At, D)
end

A_mul_Bc(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, ctranspose(B))
A_mul_Bc(D::Diagonal, Q::Union{Base.LinAlg.QRCompactWYQ,Base.LinAlg.QRPackedQ}) = A_mul_Bc!(Arr
function A_mul_Bc(D::Diagonal, A::AbstractMatrix)
    Ac = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    ctranspose!(Ac, A)
    A_mul_B!(D, Ac)
end

A_mul_Bt(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, transpose(B))
function A_mul_Bt(D::Diagonal, A::AbstractMatrix)
    At = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    transpose!(At, A)
    A_mul_B!(D, At)
end

A_mul_B!(A::Diagonal,B::Diagonal)  = throw(MethodError(A_mul_B!, Tuple{Diagonal,Diagonal}))
At_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(At_mul_B!, Tuple{Diagonal,Diagonal}))
Ac_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(Ac_mul_B!, Tuple{Diagonal,Diagonal}))
A_mul_B!(A::Base.LinAlg.QRPackedQ, D::Diagonal) = throw(MethodError(A_mul_B!, Tuple{Diagonal,Di
```

```
    A_mul_B!(At, D)
end

A_mul_Bc(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, ctranspose(B))
A_mul_Bc(D::Diagonal, Q::Union{Base.LinAlg.QRCompactWYQ,Base.LinAlg.QRPackedQ}) = A_mul_Bc!(Arr
function A_mul_Bc(D::Diagonal, A::AbstractMatrix)
    Ac = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    ctranspose!(Ac, A)
    A_mul_B!(D, Ac)
end

A_mul_Bt(D::Diagonal, B::AbstractTriangular) = A_mul_B!(D, transpose(B))
function A_mul_Bt(D::Diagonal, A::AbstractMatrix)
    At = similar(A, promote_op(*, eltype(A), eltype(D.diag)), (size(A, 2), size(A, 1)))
    transpose!(At, A)
    A_mul_B!(D, At)
end

A_mul_B!(A::Diagonal,B::Diagonal)  = throw(MethodError(A_mul_B!, Tuple{Diagonal,Diagonal}))
At_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(At_mul_B!, Tuple{Diagonal,Diagonal}))
Ac_mul_B!(A::Diagonal,B::Diagonal) = throw(MethodError(Ac_mul_B!, Tuple{Diagonal,Diagonal}))
A_mul_B!(A::Base.LinAlg.QRPackedQ, D::Diagonal) = throw(MethodError(A_mul_B!, Tuple{Diagonal,Di
```

```
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    LowerTriangular(A.data)
end
function A_mul_B!(A::UnitUpperTriangular, D::Diagonal)
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    UpperTriangular(A.data)
end
function A_mul_B!(D::Diagonal, B::UnitLowerTriangular)
    A_mul_B!(D, B.data)
    for i = 1:size(B, 1)
        B.data[i,i] = D.diag[i]
    end
    LowerTriangular(B.data)
end
function A_mul_B!(D::Diagonal, B::UnitUpperTriangular)
    A_mul_B!(D, B.data)
```

```julia
    typeof(A)(A_mul_B!(A.data, D))
function A_mul_B!(A::UnitLowerTriangular, D::Diagonal)
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    LowerTriangular(A.data)
end
function A_mul_B!(A::UnitUpperTriangular, D::Diagonal)
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    UpperTriangular(A.data)
end
function A_mul_B!(D::Diagonal, B::UnitLowerTriangular)
    A_mul_B!(D, B.data)
    for i = 1:size(B, 1)
        B.data[i,i] = D.diag[i]
    end
```

```
Ac_mul_B!{T<:Union{Float32,Float64}}(::Union{Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}) at linalg/matmul.jl:176
  Ac_mul_B!(::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::SparseMatrixCSC{Tv,Ti<:Integer},
 ::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseAr
```

```
julia> @time mapreduce(exp, +, x);
  1.142859 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(exp, +, xd);
  0.316538 seconds (603 allocations: 44.703 KB)

julia> @time mapreduce(identity, +, x);
  3.886222 seconds (11.89 k allocations: 543.521 KB)

julia> @time mapreduce(identity, +, x);
  0.063362 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, x);
  0.061466 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, xd);
  1.902190 seconds (50.13 k allocations: 2.054 MB)

julia> @time mapreduce(identity, +, xd);
  0.044912 seconds (590 allocations: 44.125 KB)

julia> @time mapreduce(identity, +, xd);
  0.042661 seconds (615 allocations: 45.078 KB)

julia> @time mapreduce(identity, +, x);
  0.060044 seconds (5 allocations: 176 bytes)

julia>
```

```
  | | |_| | | | (_| |  |  Version 0.5.0 (2016-09-19 18:14 UTC)
 _/ |\__'_|_|_|\__'_|  |
|__/                   |  x86_64-apple-darwin16.5.0

julia> using Compat

julia> IndexStyle
Base.LinearIndexing

julia> IndexStyle(randn(3,3))
Base.LinearFast()

julia> Base.LinearFast()
Base.LinearFast()

julia> IndexLinear()
Base.LinearFast()

julia>
➜ DataArrays git:(2f235b4) ✗ cd
➜ ~ cd julia-dev
➜ julia-dev git:(master) git diff
➜ julia-dev git:(master) git log
➜ julia-dev git:(master) cc --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.5.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
➜ julia-dev git:(master)
```

```
^C
--- orion.cosmo.fas.nyu.edu ping statistics ---
7 packets transmitted, 0 packets received, 100.0% packet loss
➜ Celeste git:(anj/06) cd
➜ ~ cd .julia/v0.6/Celeste
➜ Celeste git:(anj/06) ✗ cd
➜ ~ cd Dropbox\ \(MIT\)
➜ Dropbox (MIT) cd Talks
➜ Talks cp CSE17
➜ Talks cp CopperMarch2016/CopperMountainMarch2016.ipynb CopperMarch2016/ErrorRecurrences.png CopperMarch2016/ErrorRecurr
opperMarch2016/MIT_logo.svg CopperMarch2016/Screen.png CopperMarch2016/Stats.jpg CopperMarch2016/TSVDConv.jl CopperMarch201
alues.png CopperMarch2016/TSVDConvTenValuesOld.png CopperMarch2016/TSVDConvTenthValueWithError.png CopperMarch2016/TSVDCon
hErrorOld.png CopperMarch2016/TSVDout.jls CopperMarch2016/csail_logo.png CopperMarch2016/eigenvaluesthread30.txt CopperMar
aConv.jl CopperMarch2016/flashpcaOut.txt CopperMarch2016/logo.svg CopperMarch2016/randvalues.txt CopperMarch2016
➜ Talks cp CSE17
➜ Talks ping macmini.julia.csail.mit.edu
PING macmini.julia.csail.mit.edu (128.30.87.54): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout for icmp_seq 6
Request timeout for icmp_seq 7
Request timeout for icmp_seq 8
Request timeout for icmp_seq 9
^C
--- macmini.julia.csail.mit.edu ping statistics ---
```

```
 Ac_mul_B!{T<:Union{Float32,Float64}}(::Union{Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}) at linalg/matmul.jl:176
 Ac_mul_B!(::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::SparseMatrixCSC{Tv,Ti<:Integer},
 ::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseAr
```

```
julia> @time mapreduce(exp, +, x);
  1.142859 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(exp, +, xd);
  0.316538 seconds (603 allocations: 44.703 KB)

julia> @time mapreduce(identity, +, x);
  3.886222 seconds (11.89 k allocations: 543.521 KB)

julia> @time mapreduce(identity, +, x);
  0.063362 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, x);
  0.061466 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, xd);
  1.902190 seconds (50.13 k allocations: 2.054 MB)

julia> @time mapreduce(identity, +, xd);
  0.044912 seconds (590 allocations: 44.125 KB)

julia> @time mapreduce(identity, +, xd);
  0.042661 seconds (615 allocations: 45.078 KB)

julia> @time mapreduce(identity, +, x);
  0.060044 seconds (5 allocations: 176 bytes)

julia> 
```

```
  | | |_| | | | (_| |  |  Version 0.5.0 (2016-09-19 18:14 UTC)
 _/ |\__'_|_|_|\__'_|  |
|__/                   |  x86_64-apple-darwin16.5.0

julia> using Compat

julia> IndexStyle
Base.LinearIndexing

julia> IndexStyle(randn(3,3))
Base.LinearFast()

julia> Base.LinearFast()
Base.LinearFast()

julia> IndexLinear()
Base.LinearFast()

julia>
➜  DataArrays git:(2f235b4) ✗ cd
➜  ~ cd julia-dev
➜  julia-dev git:(master) git diff
➜  julia-dev git:(master) git log
➜  julia-dev git:(master) cc --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.5.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
➜  julia-dev git:(master) 
```

```
^C
--- orion.cosmo.fas.nyu.edu ping statistics ---
7 packets transmitted, 0 packets received, 100.0% packet loss
➜  Celeste git:(anj/06) cd
➜  ~ cd .julia/v0.6/Celeste
➜  Celeste git:(anj/06) ✗ cd
➜  ~ cd Dropbox\ \(MIT\)
➜  Dropbox (MIT) cd Talks
➜  Talks cp [illegible] CSE17
➜  Talks cp CopperMarch2016/CopperMountainMarch2016.ipynb CopperMarch2016/ErrorRecurrences.png CopperMarch2016/ErrorRecurrencesOl
opperMarch2016/MIT_logo.svg CopperMarch2016/Screen.png CopperMarch2016/Stata.jpg CopperMarch2016/TSVDConv.jl CopperMarch2016/TSVDC
alues.png CopperMarch2016/TSVDConvTenValuesOld.png CopperMarch2016/TSVDConvTenthValueWithError.png CopperMarch2016/TSVDConvTenthV
hErrorOld.png CopperMarch2016/TSVDout.jls CopperMarch2016/csail_logo.png CopperMarch2016/eigenvaluesthread30.txt CopperMarch2016/
aConv.jl CopperMarch2016/flashpcaOut.txt CopperMarch2016/logo.svg CopperMarch2016/randvalues.txt CopperMarch2016
➜  Talks cp [illegible] CSE17
➜  Talks ping macmini.julia.csail.mit.edu
PING macmini.julia.csail.mit.edu (128.30.87.54): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout for icmp_seq 6
Request timeout for icmp_seq 7
Request timeout for icmp_seq 8
Request timeout for icmp_seq 9
^C
--- macmini.julia.csail.mit.edu ping statistics ---
```

```
Ac_mul_B!{T<:Union{Float32,Float64}}(::Union{Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::Union{Base.ReshapedArray{T<:Union{Float32,Float64},1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T<:Union{Float32,Float64},2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T<:Union{Float32,Float64},1},DenseArray{T<:Union{Float32,Float64},2},SubArray{T<:Union{Float32,Float64},1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T<:Union{Float32,Float64},2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}) at linalg/matmul.jl:176
  Ac_mul_B!(::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L}}, ::SparseMatrixCSC{Tv,Ti<:Integer}, ::Union{Base.ReshapedArray{T,1,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},Base.ReshapedArray{T,2,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray{T,1},DenseArray{T,2},SubArray{T,1,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArray},I<:Tuple{Vararg{Union{Base.AbstractCartesianIndex,Colon,Int64,Range{Int64}},N}},L},SubArray{T,2,A<:Union{Base.ReshapedArray{T,N,A<:DenseArray,MI<:Tuple{Vararg{Base.MultiplicativeInverses.SignedMultiplicativeInverse{Int64},N}}},DenseArr
```

```
julia> @time mapreduce(exp, +, x);
  1.142859 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(exp, +, xd);
  0.316538 seconds (603 allocations: 44.703 KB)

julia> @time mapreduce(identity, +, x);
  3.886222 seconds (11.89 k allocations: 543.521 KB)

julia> @time mapreduce(identity, +, x);
  0.063362 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, x);
  0.061466 seconds (5 allocations: 176 bytes)

julia> @time mapreduce(identity, +, xd);
  1.902190 seconds (50.13 k allocations: 2.054 MB)

julia> @time mapreduce(identity, +, xd);
  0.044912 seconds (590 allocations: 44.125 KB)

julia> @time mapreduce(identity, +, xd);
  0.042661 seconds (615 allocations: 45.078 KB)

julia> @time mapreduce(identity, +, x);
  0.060044 seconds (5 allocations: 176 bytes)

julia>
```

```
  | | |_| | | | (_| |  |  Version 0.5.0 (2016-09-19 18:14 UTC)
 _/ |\__'_|_|_|\__'_|  |
|__/                   |  x86_64-apple-darwin16.5.0

julia> using Compat

julia> IndexStyle
Base.LinearIndexing

julia> IndexStyle(randn(3,3))
Base.LinearFast()

julia> Base.LinearFast()
Base.LinearFast()

julia> IndexLinear()
Base.LinearFast()

julia>
➜  DataArrays git:(2f235b4) ✗ cd
➜  ~ cd julia-dev
➜  julia-dev git:(master) git diff
➜  julia-dev git:(master) git log
➜  julia-dev git:(master) cc --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.5.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
➜  julia-dev git:(master)
```

```
^C
--- orion.cosmo.fas.nyu.edu ping statistics ---
7 packets transmitted, 0 packets received, 100.0% packet loss
➜  Celeste git:(anj/06) cd
➜  ~ cd .julia/v0.6/Celeste
➜  Celeste git:(anj/06) ✗ cd
➜  ~ cd Dropbox\ \(MIT\)
➜  Dropbox (MIT) cd Talks
➜  Talks cp CSE17
➜  Talks cp CopperMarch2016/CopperMountainMarch2016.ipynb CopperMarch2016/ErrorRecurrences.png CopperMarch2016/ErrorRecurrencesOld.png CopperMarch2016/MIT_logo.svg CopperMarch2016/Screen.png CopperMarch2016/Stata.jpg CopperMarch2016/TSVDConv.jl CopperMarch2016/TSVDConvTenValues.png CopperMarch2016/TSVDConvTenValuesOld.png CopperMarch2016/TSVDConvTenthValueWithError.png CopperMarch2016/TSVDConvTenthValueWithErrorOld.png CopperMarch2016/TSVDout.jls CopperMarch2016/csail_logo.png CopperMarch2016/eigenvaluesthread30.txt CopperMarch2016/flashpcaConv.jl CopperMarch2016/flashpcaOut.txt CopperMarch2016/logo.svg CopperMarch2016/randvalues.txt CopperMarch2016
➜  Talks cp CSE17
➜  Talks ping macmini.julia.csail.mit.edu
PING macmini.julia.csail.mit.edu (128.30.87.54): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout for icmp_seq 6
Request timeout for icmp_seq 7
Request timeout for icmp_seq 8
Request timeout for icmp_seq 9
^C
--- macmini.julia.csail.mit.edu ping statistics ---
```

```
n{LowerTriangular,UpperTriangular}, D::Diagonal) =
mul_B!(A.data, D))
!(A::UnitLowerTriangular, D::Diagonal)
ata, D)
ze(A, 1)
,i] = D.diag[i]

lar(A.data)

!(A::UnitUpperTriangular, D::Diagonal)
ata, D)
ze(A, 1)
,i] = D.diag[i]

lar(A.data)

!(D::Diagonal, B::UnitLowerTriangular)
B.data)
ze(B, 1)
,i] = D.diag[i]
```

```
Version 0.5.0 (2016-09-19 18:14 U
x86_64-apple-darwin16.5.0

julia> using Compat

julia> IndexStyle
Base.LinearIndexing

julia> IndexStyle(randn(3,3))
Base.LinearFast()

julia> Base.LinearFast()
Base.LinearFast()

julia> IndexLinear()
Base.LinearFast()

julia>
→ DataArrays git:(2f235b4) ✗ cd
→ ~ cd julia-dev
→ julia-dev git:(master) git diff
→ julia-dev git:(master) git log
→ julia-dev git:(master) cc --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.5.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/To
→ julia-dev git:(master)
```

```
A_mul_B!(A::Union{LowerTriangular,UpperTriangular}, D::Diagonal) =
    typeof(A)(A_mul_B!(A.data, D))
function A_mul_B!(A::UnitLowerTriangular, D::Diagonal)
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    LowerTriangular(A.data)
end
function A_mul_B!(A::UnitUpperTriangular, D::Diagonal)
    A_mul_B!(A.data, D)
    for i = 1:size(A, 1)
        A.data[i,i] = D.diag[i]
    end
    UpperTriangular(A.data)
end
function A_mul_B!(D::Diagonal, B::UnitLowerTriangular)
    A_mul_B!(D, B.data)
    for i = 1:size(B, 1)
        B.data[i,i] = D.diag[i]
    end
```

Example 1: Rational linear system of equations

?

Example 1: Rational linear system of equations

0 - 1

Example 1: Rational linear system of equations

Example 1: Rational linear system of equations

In [26]:
```
Ar = Matrix{Rational{BigInt}}(rand(1:10,3,3))/10
```

Out[26]:
```
3×3 Array{Rational{BigInt},2}:
 4//5  7//10  1//2
 2//5  3//10  1//5
 1//5  1//1   4//5
```

In [27]:
```
x = ones(Int,3)
b = Ar*x
```

Out[27]:
```
3-element Array{Rational{BigInt},1}:
 2//1
 9//10
 2//1
```

In [28]:
```
Ar\b
```

Out[28]:
```
3-element Array{Rational{BigInt},1}:
 1//1
 1//1
 1//1
```

Example 2: The Hilbert matrix I

$$h_{ij} = \frac{1}{i+j-1}$$

In [23]:
```
n = 8
H = Rational{BigInt}[1//(i+j-1) for i = 1:n,j = 1:n]
```

Out[23]:
```
8×8 Array{Rational{BigInt},2}:
 1//1  1//2  1//3   1//4   1//5   1//6   1//7   1//8
 1//2  1//3  1//4   1//5   1//6   1//7   1//8   1//9
 1//3  1//4  1//5   1//6   1//7   1//8   1//9   1//10
 1//4  1//5  1//6   1//7   1//8   1//9   1//10  1//11
 1//5  1//6  1//7   1//8   1//9   1//10  1//11  1//12
 1//6  1//7  1//8   1//9   1//10  1//11  1//12  1//13
 1//7  1//8  1//9   1//10  1//11  1//12  1//13  1//14
 1//8  1//9  1//10  1//11  1//12  1//13  1//14  1//15
```

?

Example 2: The Hilbert matrix II

In [5]:
```
norm(Float64.(inv(H)) - inv(Float64.(H)), Inf)
```

Out[5]:
```
474.97878527641296
```

not completely fair since

In [29]:
```
H == Float64.(H)
```

Out[29]:
```
false
```

instead

In [7]:
```
Rational{BigInt}.(Float64.(H)) == Float64.(H)
```

Out[7]:
```
true
```

In [8]:
```
norm(Float64.(inv(Rational{BigInt}.(Float64.(H)))) -
    inv(Float64.(H)), Inf)
```

Out[8]:
```
509.50778410799103
```

Example 3: Finite fields GF(p)

In [9]:
```
immutable GF{P,T<:Integer} <: Number
    data::T
    function GF(x::Integer)
        return new(mod(x, P))
    end
end
```

In [10]:
```
using Primes
function (::Type{GF{P}}){P}(x::Integer)
    if !isprime(P)
        throw(ArgumentError("P must be a prime"))
    end
    return GF{P,typeof(x)}(mod(x, P))
end
Base.convert{P,T}(::Type{GF{P,T}}, x::Integer) = GF{P}(x)
Base.convert{P}(::Type{GF{P}}, x::Integer) = GF{P}(x)
Base.convert{P,T}(::Type{GF{P,T}}, x::GF{P}) = GF{P,T}(x.data)
Base.promote_rule{P,T1,T2<:Integer}(::Type{GF{P,T1}}, ::Type{T2}) = GF{P,promote_
)}
Base.show{P}(io::IO, x::GF{P}) = print(io, "$(x.data) mod $P")
```

0-4

Division requires a little more care

In [13]:

```
function Base.inv{P,T}(x::GF{P,T})
    if x == 0
        throw(DivideError())
    end
    r, u, v = gcdx(x.data, P)
    GF{P,T}(u)
end
Base.:/{P}(x::GF{P}, y::GF{P}) = x*inv(y)
```

In [14]:

```
@show x/y;
```

```
x / y = 3 mod 5
```

```
Out[16]: 4-element Array{GF{5,Int64},1}:
          1 mod 5
          1 mod 5
          1 mod 5
          1 mod 5
```

Example 4: Symbolic linear algebra with SymPy+Julia I

Example 4: Symbolic linear algebra with SymPy+Julia I

In [17]:

```
using SymPy
z = Sym[Sym("zeta_$i") for i = 1:3]
D = diagm(Sym["delta_$i" for i = 1:3])
A = [Sym("alpha") z'; z D]
```

```
WARNING: using SymPy.H in module Main conflicts with an existing identif
ier.
```

Out[17]:

$$\begin{bmatrix} \alpha & \overline{\zeta_1} & \overline{\zeta_2} & \overline{\zeta_3} \\ \zeta_1 & \delta_1 & 0 & 0 \\ \zeta_2 & 0 & \delta_2 & 0 \\ \zeta_3 & 0 & 0 & \delta_3 \end{bmatrix}$$