How to Install the Package

In Julia REPL, switch to the “pkg” environment by typing “]” and execute the following code.

```
(@v1.6) pkg> add "https://github.com/Mengzhen-Zhang/Symplectic.jl.git"
```

Get back to the ‘julia’ environment and the package is ready to use now.

```
julia> using Symplectic
```

Data Type for Manipulating Symplectic Matrices

The package offers a symplectic-matrix data type, “Symp”, which can be constructed as follows:

```
julia> typeof(Symp([1 0; 0 1]))
Symp{Int64}

julia> Symp([1 0; 0 1])
2×2 Matrix{Int64}:
 1  0
 0  1

julia> Symp([1 0; 0 1]).S
2×2 Matrix{Int64}:
 1  0
 0  1

julia> typeof(Symp([1 0; 0 1]).S)
Matrix{Int64} (alias for Array{Int64, 2})
```

The argument of this constructor must be a symplectic matrix (::AbstractMatrix). Although the elements of “Symp” can be any data type supported by “Matrix{T}”, the “BigFloat” type is strongly recommended in order to ensure the precision of the calculation using “Symp”s. The “Symp” data type has a field named “S” containing the stored matrix. It can be accessed as follows:

Overloaded Methods in “Base”

The “Symp” date type supports the following methods contained in “Base”:

1. **Base.:***
   Multiplication

```
julia> typeof(S1), typeof(S2)
(Symp{BigFloat}, Symp{BigFloat})

julia> typeof(S1 * S2 )
Symp{BigFloat}

julia> typeof(S1), typeof(M1)
(Symp{BigFloat}, Matrix{BigFloat})

julia> typeof(S1 * M1)
Matrix{BigFloat} (alias for Array{BigFloat, 2})
```

2. **Base.inv**
   Inverse

```
julia> typeof(S1), typeof(inv(S1))
(Symp{BigFloat}, Symp{BigFloat})

julia> S1 * inv(S1)
4×4 Matrix{BigFloat}:
  1.0           0.0           4.31808e-78   0.0
  0.0           1.0          -1.29543e-77  -6.47713e-78
 -6.47713e-78  -5.39761e-79   1.0           0.0
  1.29543e-77   4.31808e-78   0.0           1.0
```

3. **Base.transpose** and **Base.adjoint**
   Transpose and Adjoint

```
julia> typeof(transpose(S1))
Symp{BigFloat}

julia> typeof(S1')
Symp{BigFloat}
```

4. **Base.:-** and **Base.:+**
   Unary minus and unary plus

```
julia> typeof(-S1)
Symp{BigFloat}

julia> typeof(+S1)
Symp{BigFloat}
```

5. **Base.:+**
   Addition

```
julia> typeof(S1), typeof(S2), typeof(M1)
(Symp{BigFloat}, Symp{BigFloat}, Matrix{BigFloat})

julia> typeof(S1 + S2), typeof(S1 + M1), typeof(M1 + S1)
(Matrix{BigFloat}, Matrix{BigFloat}, Matrix{BigFloat})
```

6. **Base:-**
   Subtraction

```
julia> typeof(S1), typeof(S2), typeof(M1)
(Symp{BigFloat}, Symp{BigFloat}, Matrix{BigFloat})

julia> typeof(S1 - S2), typeof(S1 - M1), typeof(M1 - S1)
(Matrix{BigFloat}, Matrix{BigFloat}, Matrix{BigFloat})
```

7. **⊗ = Base.kron**
   Tensor product/Kronecker product

```
julia> typeof(S1), typeof(S2), typeof(M1)
(Symp{BigFloat}, Symp{BigFloat}, Matrix{BigFloat})

julia> typeof(S1 ⊗ S2), typeof(S1 ⊗ M1), typeof(M1 ⊗ S1)
(Matrix{BigFloat}, Matrix{BigFloat}, Matrix{BigFloat})
```

8. **Base.:^**
   Matrix Power

```
julia> typeof(S1), typeof(S1 ^ -3)
(Symp{BigFloat}, Symp{BigFloat})
```

9. **Base.:/**
   Division by number

```
julia> typeof(S1)
Symp{BigFloat}

julia> typeof(S1 / 1.5)
Matrix{BigFloat} (alias for Array{BigFloat, 2})
```

Overloaded Methods in “LinearAlgebra”

These methods in “LinearAlgebra” are reloaded or renamed:

1. **isSquare = LinearAlgebra.checksquare**
   Check if a matrix (::AbstractMatrix) is square and return the order of square matrix

```
julia> isSquare([1 2; 3 1])
2
```

2. **LinearAlgebra.issymmetric**
   Check if a matrix is symmetric

```
julia> using LinearAlgebra

julia> issymmetric(S1)
false

julia> typeof(S1)
Symp{BigFloat}
```

3. **LinearAlgebra.isposdef**
   Check if a matrix is positive definite

```
julia> using LinearAlgebra

julia> isposdef(S1)
false

julia> typeof(S1)
Symp{BigFloat}
```

4. **LinearAlgebra.isdag**
   Check if a matrix is diagonal

```
julia> using LinearAlgebra

julia> isdiag(S1)
false

julia> typeof(S1)
Symp{BigFloat}
```

5. **LinearAlgebra.norm**
   Calculate the matrix norm

```
julia> using LinearAlgebra

julia> norm(S1)
2.70066327046728111967960974478854671522300630771333421527685261663553087475881

julia> typeof(S1)
Symp{BigFloat}
```

Other Methods

The following methods are defined for “Symp”:

1. **nModes**
   Retrieve the number of modes

```
julia> S1
4×4 Matrix{BigFloat}:
  0.706637   -0.437139   -0.0396703   0.208195
  0.0544726   1.35446    -0.195511    0.545202
  0.0137328   0.0310724   0.321723   -0.983636
 -0.403134    0.476948    0.458531    1.64705

julia> typeof(S1)
Symp{BigFloat}

julia> nModes(S1)
2
```

2. **sympRound**
   Rounding a symplectic matrix using Cayley Transform

```
julia> typeof(S1)
Symp{BigFloat}

julia> sympRound(S1) - S1
4×4 Matrix{BigFloat}:
 -1.72723e-77   1.72723e-77  -4.85784e-78   1.29543e-77
  8.09641e-78   1.72723e-77  -2.15904e-78   8.63617e-78
  6.47713e-78  -4.31808e-78   0.0          -8.63617e-78
 -8.63617e-78   2.15904e-77  -4.31808e-78   1.72723e-77
```

   This is unnecessary most of the times.

3. Ω **= Omega**
   Generate an n-mode symplectic form

```
julia> typeof(Ω(2)), typeof(Omega(2))
(Symp{Int64}, Symp{Int64})

julia> Ω(2)
4×4 Matrix{Int64}:
  0  1   0  0
 -1  0   0  0
  0  0   0  1
  0  0  -1  0

julia> Omega(2)
4×4 Matrix{Int64}:
  0  1   0  0
 -1  0   0  0
  0  0   0  1
  0  0  -1  0
```

4. **Id**
   Generate an n-mode identity symplectic matrix ():

```
julia> typeof(Id(2))
Symp{Int64}

julia> Id(2)
4×4 Matrix{Int64}:
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
```

5. ⊕ **= dsum**
   Direct sum of matrices and symplectic matrices

```
julia> typeof(S1), typeof(S2), typeof(M1)
(Symp{BigFloat}, Symp{BigFloat}, Matrix{BigFloat})

julia> typeof(S1 ⊕ S2), typeof(S1 ⊕ M1), typeof(M1 ⊕ S1), typeof(M1 ⊕ M1)
(Symp{BigFloat}, Matrix{BigFloat}, Matrix{BigFloat}, Matrix{BigFloat})

julia> Ω(2) - ( Ω(1) ⊕ Ω(1) )
4×4 Matrix{Int64}:
 0  0  0  0
 0  0  0  0
 0  0  0  0
 0  0  0  0
```

Other Functions

The following functions are defined for use:

1. **isGeneric**
   Check if a Symplectic Matrix is "generic"

```
julia> isGeneric(S1)
true
```

2. **monoSymp**
   Generate a single mode phase shifting symplectic matrix

```
julia> typeof(monoSymp(π/4))
Symp{Float64}

julia> monoSymp(π/4)
2×2 Matrix{Float64}:
  0.707107  0.707107
 -0.707107  0.707107
```

3. **beamsplitter**
   Generate a two-mode beamsplitter

```
julia> beamSplitter(π/4)
4×4 Matrix{Float64}:
  0.707107   0.0       0.707107  0.0
  0.0        0.707107  0.0       0.707107
 -0.707107  -0.0       0.707107  0.0
 -0.0       -0.707107  0.0       0.707107

julia> typeof(beamSplitter(π/4))
Symp{Float64}
```

4. **randomSymp**
   Generate an n-mode random symplectic matrix

```
julia> randomSymp(2)
4×4 Matrix{BigFloat}:
  0.689225  -0.770731  -0.389114   0.183684
  0.59005    1.33152    0.216431   0.855099
 -0.268891   0.25658    0.514506  -0.321003
  0.196801   1.19748    1.03206    2.02367
```

5. **randomGenericSymp**
   Generate an n-mode random generic symplectic matrix

```
julia> randomGenericSymp(2)
4×4 Matrix{BigFloat}:
  0.711389   -0.501381  -0.403911    0.686776
  0.570813    1.13201   -0.38362     0.878799
 -0.128814    0.483964   0.690532   -0.481639
 -0.0503398   0.89942   -0.0588743   1.62172
```

6. **localSympFromQuad**
   Generate a local symplectic matrix tranforming a quadrature u to another quadrature Ωv, where u, v are arguments of the function and Ω is the symplectic form

```
julia> localSympFromQuad([1,2,3,4], [2,1,3,4])
4×4 Matrix{Float64}:
 0.6  -0.8  0.0   0.0
 0.8   0.6  0.0   0.0
 0.0   0.0  0.0  -1.0
 0.0   0.0  1.0   0.0
```

7. **getDecoupleSequence**
   Generate the sequence $S^{(4)}L^{(3)}S^{(3)}L^{(2)}S^{(2)}L^{(1)}S^{(1)}$ that yields a symplectic matrix with the mth

decoupled.

```
julia> typeof(S1)
Symp{BigFloat}

julia> typeof(getDecoupleSequence(S1, S1, S1, S1, 1))
Vector{Symp} (alias for Array{Symp, 1})

julia> typeof(getDecoupleSequence(S1, S1, S1, S1))
Vector{Symp} (alias for Array{Symp, 1})

julia> *(getDecoupleSequence(S1, S1, S1, S1)...)
4×4 Matrix{BigFloat}:
  1.18747e-77  -1.0           3.3735e-79    7.82653e-78
  1.0           0.506761     -8.36629e-78   1.29543e-77
 -2.15904e-78   4.26411e-77  -0.486532      1.13213
 -1.40338e-77   6.47713e-78   2.06576      -6.86225
```

8. **getInterferenceBasedSequence**
   Given a generic Gaussian unitary coupler and a target symplectic matrix, generate the sequence $L^{(R)}SL^{(15)}SL^{(14)}\cdots L^{(1)}S$ that yieds the desired target operation.

```
julia> sequence = getInterferenceBasedSequence(randomGenericSymp(6), beamSplitter(π/4));

julia> round.(*(sequence...).S, digits = 3)
12×12 Matrix{BigFloat}:
  0.707   0.0     0.707  -0.0        -0.0    …       0.0        0.0       0.0        0.0
 -0.0     0.707   0.0     0.707       0.0           -0.0       -0.0      -0.0        0.0
 -0.707  -0.0     0.707   0.0         0.0           -0.0       -0.0      -0.0        0.0
 -0.0    -0.707  -0.0     0.707      -0.0            0.0        0.0       0.0       -0.0
 -0.0     0.0    -0.0    -0.0       392.339      -1956.21    -858.95   -433.374   -459.145
  0.0    -0.0     0.0    -0.0     -6738.96   …    18336.0     7650.64   5262.09    1962.1
  0.0    -0.0     0.0     0.0       312.517        264.875    163.82    -96.624    352.834
  0.0     0.0     0.0    -0.0    -13960.7         38368.6    16027.4   10872.2     4309.22
 -0.0    -0.0    -0.0     0.0      5148.96       -13337.4    -5533.66  -3934.07   -1237.79
 -0.0    -0.0    -0.0     0.0      3405.66        -9933.25   -4176.03  -2700.8    -1306.82
 -0.0    -0.0    -0.0     0.0      3726.43   …   -10508.8    -4402.43  -2918.12   -1279.24
 -0.0     0.0    -0.0    -0.0      7671.45       -22701.2    -9560.25  -6191.14   -3018.58
```

9. **sympToGraph**
   Generate a graph based on the given symplectic matrix

```
julia> sympToGraph( beamSplitter(π/2) )
2×2 Matrix{Bool}:
 0  1
 1  0
```

10. **circulator**
    Generate a circulator-like symplectic matrix based on the given permutation

```
julia> circulator([2, 3, 4, 1])
8×8 Matrix{Float64}:
 0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
```

11. **getColorSetsFromSymp**
    Color the graph generated from a symplectic matrix

12. **teleportationBasedSymplecticControl**
    Generate a symplectic matrix using the generalized CV teleportation protocol. The arguments are a symplectic matrix, an array of input modes and an array of the output modes.

```
julia> teleportationBasedSymplecticControl( randomSymp(4), [1, 2], [1, 2] )
4×4 Matrix{BigFloat}:
 -0.522644  -0.61642    0.492881   0.801113
  2.72389    1.5479    -0.261     -0.160588
  2.68674    1.52683    1.72416    1.06125
  1.11946    0.684533   1.16731    1.22313

julia> typeof(teleportationBasedSymplecticControl( randomSymp(4), [1, 2], [1, 2] ))
Symp{BigFloat}
```

13. **randomPassiveLocalSymp**
    Generate a random n-mode passive local symplectic matrix
14. **randomLocalSymp**
    Generate a random n-mode local symplectic matrix
15. **randomizeSymp**
    Randomize a symplectic matrix with an interference-based sequence of $l + 1$ local symplectic matrices: