# A Guide to *Mathematica* Packages for Physicists

Yi-Zhuang You, UCSD

2019-10-05

**Abstract**

This document provides a guide to the bundle of *Mathematica* packages for physicists. The bundle is available in the Github repository.

**Keywords:** *Mathematica*

## Introduction

### ■ Content

This repository contains *Mathematica* packages and stylesheets that are useful for me.

- Package
  - **PauliAlgebra**: symbolic handling the algebra and representation of Pauli operators
  - **LoopIntegrate**: performing loop integration in quantum field theory (with dimension regularization)
  - **MatsubaraSum**: performing Matsubara summation analytically
  - **DiagramEditor**: an interactive editor of Feynman diagrams (no diagrammatic evaluation)
  - **Themes**: a self-made plot theme for Mathematica, called “Academic”
  - **Toolkit**: miscellaneous functions, including `BZPlot` for plotting band structure, `tTr` for tensor network contraction, `ComplexMatrixPlot` for complex matrix visualization, `Pf` for matrix Pfaffian
- Stylesheet
  - **CMU Article**: *Mathematica* style sheet based on Computer Modern Unicode fonts (the fonts need to be installed separately to the operating system)
- FrontEnd Configuration

## ■ Installation Instruction

The bundle of packages can be downloaded from

https://github.com/EverettYou/Mathematica-for-physics

To install everything:

1. unzip this repository in a folder,
2. open `install.m` in *Mathematica*,
3. click the `Run Package` button to the top right,
4. quit Mathematica and restart.

---

# PauliAlgebra Package

## ■ Overview

The package `PauliAlgebra` provides the following functions:

```
?PauliAlgebra`*
```

PauliAlgebra`

| | | |
|---|---|---|
| Abstract | LieAlgebra | σExp |
| ActionSpace | nTr | σHermitian |
| Anticommutator | OrthogonalTransform | σInverse |
| AnticommuteQ | Qubit | σLog |
| AntisymmetricQ | Represent | σPolynomialQ |
| C4 | Swap | σPower |
| Cl | SymmetricQ | σSelect |
| Commutator | UnitaryTransform | σSqrt |
| CommuteQ | σ | σTr |
| ConjugateTransform | σ0 | σTranspose |
| Controlled | σConjugate | |
| Hadamard | σDet | |

# Arithmetic

## Symbolic Representation

Four basic **Pauli matrices** are defined as

$$\sigma^0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \sigma^1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma^2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma^3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1}$$

**Pauli operators** are tensor products of series of *Pauli matrices*,

$$\sigma^{abc\ldots} = \sigma^a \otimes \sigma^b \otimes \sigma^c \otimes \ldots, \tag{2}$$

which can be input as `σ[a,b,c,…]` directly. The indices $a$, $b$, $c$, … only take values of 0, 1, 2, 3.

### Tensor Product

*Pauli operators* can explicitly constructed by the **tensor product** (⊗ is entered as ESCc*ESC).

```
σ[a] ⊗ σ[b] ⊗ σ[c]
```

```
σ[a, b, c]
```

The tensor product construction will be translated by *Mathematica* to the short-handed notation automatically.

### Dot Product

The *composition* of Pauli operators is denoted by a **dot product** ·, which can be entered as ESC.ESC. Dot product is carried out according to the Pauli algebra. For example,

```
σ[2] · σ[3]
```

```
ⅈ σ[1]
```

```
σ[1] · σ[1]
```

```
σ[0]
```

Following shows the multiplication table calculated by *Mathematica*.

```
TableForm[Table[σ[i] · σ[j], {i, 0, 3}, {j, 0, 3}],
 TableHeadings → {Table[σ[i], {i, 0, 3}], Table[σ[j], {j, 0, 3}]}]
```

| | σ[0] | σ[1] | σ[2] | σ[3] |
|---|---|---|---|---|
| σ[0] | σ[0] | σ[1] | σ[2] | σ[3] |
| σ[1] | σ[1] | σ[0] | ⅈ σ[3] | -ⅈ σ[2] |
| σ[2] | σ[2] | -ⅈ σ[3] | σ[0] | ⅈ σ[1] |
| σ[3] | σ[3] | ⅈ σ[2] | -ⅈ σ[1] | σ[0] |

## Matrix Representation

The **matrix representation** of a Pauli operator can be constructed by `Represent` in the form of a sparse array.

```
σ[1] // Represent
```

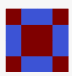
SparseArray[  Specified elements: 2 Dimensions: {2, 2} ]

```
σ[1, 2] // Represent // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix}$$

Symbolic expression can also represented

```
Represent[a σ[1, 1] + b σ[3, 3]]
```

SparseArray[  Specified elements: 8 Dimensions: {4, 4} ]

Use `MatrixForm` to convert the sparse array to its dense version for display.

```
% // MatrixForm
```

$$\begin{pmatrix} b & 0 & 0 & a \\ 0 & -b & a & 0 \\ 0 & a & -b & 0 \\ a & 0 & 0 & b \end{pmatrix}$$

Use `Abstract` to recover the Pauli operator from the above matrix.

```
Abstract[%]
```

a σ[1, 1] + b σ[3, 3]

**Abstraction** is just the *inverse function* of *representation*. Here is one more example.

```
Abstract[DiagonalMatrix[{0, 0, 0, 1}]]
```

$$\frac{1}{4}\sigma[0,0]-\frac{1}{4}\sigma[0,3]-\frac{1}{4}\sigma[3,0]+\frac{1}{4}\sigma[3,3]$$

## ■ Dimension

`Qubit` gives the **qubit number**, i.e. the log 2 of the matrix dimension.

```
Qubit[σ[1, 2, 3]]
```

3

This means that the representation space of $\sigma^{123}$ is spanned by 3 qubits or $2^3 = 8$ states.

The **identity operator** of a given qubit number $n$ can be constructed by `σ0[n]`,

```
σ0[3]
```

```
σ[0, 0, 0]
```

or the qubit number can be automatically inferred from an operator

```
σ0[σ[1, 0, 0] + σ[2, 0, 0]]
```

```
σ[0, 0, 0]
```

## Distributive Properties

The tensor product and dot product automatically distributes over plus, such that the Pauli algebra expression is always expanded.

```
(σ[1] + 2 σ[2]) ⊗ (σ[1] - σ[2])
```

```
σ[1, 1] - σ[1, 2] + 2 σ[2, 1] - 2 σ[2, 2]
```

```
(σ[1] + 2 σ[2]) · (σ[1] - σ[2])
```

```
-σ[0] - 3 ⅈ σ[3]
```

## Conjugation

Three types of conjugations are defined:

- **Complex conjugation** $A \to A^*$

```
σConjugate[σ[0] + σ[2] + ⅈ σ[3]]
```

```
σ[0] - σ[2] - ⅈ σ[3]
```

- **Transpose** $A \to A^{\mathsf{T}}$

```
σTranspose[σ[0] + σ[2] + ⅈ σ[3]]
```

```
σ[0] - σ[2] + ⅈ σ[3]
```

- **Hermitian conjugate** $A \to A^\dagger$

```
σHermitian[σ[0] + σ[2] + ⅈ σ[3]]
```

```
σ[0] + σ[2] - ⅈ σ[3]
```

## Trace

`σTr` gives the **trace** of the Pauli operator.

```
σTr[σ[0, 0] + σ[3, 3]]
```

```
2
```

# Operator Algebra

## Action Space

A generic Hermitian operator can always be expanded as a *superposition* of Pauli operators

$$A = \sum_{[\mu]} A_{[\mu]}\, \sigma^{[\mu]}, \tag{3}$$

where $[\mu] = \mu_1\, \mu_2 \ldots$ labels the **Pauli basis**. The *superposition coefficient* is given by

$$A_{[\mu]} = \frac{1}{D} \operatorname{Tr} A\, \sigma^{[\mu]}, \tag{4}$$

where $D$ is the *Hilbert space dimension*. So each operator can be mapped to a *super-state* in the operator space

$$A \to \left|A\right\rangle = \sum_{[\mu]} A_{[\mu]}\, |[\mu]\rangle. \tag{5}$$

*Composition* of two operators can be calculated using **operator product expansion**,

$$A\, B = \sum_{[\mu],[\nu],[\lambda]} c_{[\lambda]}^{[\mu][\nu]}\, A_{[\mu]}\, B_{[\nu]}\, \sigma^{[\lambda]}. \tag{6}$$

Therefore each operator can also be represented as a *super-operator* in the operator space

$$A \to \mathbb{A} = \sum_{[\mu],[\nu],[\lambda]} |[\lambda]\rangle\, c_{[\lambda]}^{[\mu][\nu]}\, A_{[\mu]}\, \langle[\nu]|, \tag{7}$$

such that . The advantage of representing $A$ in the operator space is to reduce the representation dimension, because an operator acting on itself often only spans a *subspace* whose dimension is smaller than the Hilbert space dimension.

Such subspace is called the **action space** of an operator. The *matrix representation* of an operator in the action space and the corresponding *basis* can be found by `ActionSpace`.

```
MatrixForm /@ ActionSpace[a σ[1, 1, 0] + b σ[1, 0, 0] + c σ[0, 1, 0]]
```

$$\left\{\begin{pmatrix} 0 & c & b & a \\ c & 0 & a & b \\ b & a & 0 & c \\ a & b & c & 0 \end{pmatrix}, \begin{pmatrix} \sigma[0,0,0] \\ \sigma[0,1,0] \\ \sigma[1,0,0] \\ \sigma[1,1,0] \end{pmatrix}\right\}$$

Operator algebra can be carried out in the action space.

## Operator Power

`σPower[A, n]` returns the $n$th power of an operator $A$, i.e. $A^n$.

```
σPower[a σ[1, 1] + b σ[1, 0] + c σ[0, 1], 3]
```

$$6\, a\, b\, c\, \sigma[0,0] + \left(3\, a^2\, c + 3\, b^2\, c + c^3\right) \sigma[0,1] + \left(3\, a^2\, b + b^3 + 3\, b\, c^2\right) \sigma[1,0] + \left(a^3 + 3\, a\, b^2 + 3\, a\, c^2\right) \sigma[1,1]$$

```
σPower[a σ[0] + b σ[3], -4]
```

$$\left(\frac{4\,a^2\,b^2}{\left(-a^2+b^2\right)^4}+\frac{\left(a^2+b^2\right)^2}{\left(-a^2+b^2\right)^4}\right)\sigma[0]-\frac{4\,a\,b\,\left(a^2+b^2\right)\sigma[3]}{\left(-a^2+b^2\right)^4}$$

- The package uses divide-and-conquer algorithm for fast computation of high integer powers

```
Timing[σPower[a σ[1] + b σ[3], 99]]
```

$$\left\{0.003612,\ \left(a^3\left(\left(a^3+a\,b^2\right)^2+\left(a^2\,b+b^3\right)^2\right)^{16}+a\,b^2\left(\left(a^3+a\,b^2\right)^2+\left(a^2\,b+b^3\right)^2\right)^{16}\right)\sigma[1]+\left(a^2\,b\left(\left(a^3+a\,b^2\right)^2+\left(a^2\,b+b^3\right)^2\right)^{16}+b^3\left(\left(a^3+a\,b^2\right)^2+\left(a^2\,b+b^3\right)^2\right)^{16}\right)\sigma[3]\right\}$$

- The **inverse operator** $(A \to A^{-1})$ can also be obtained via `σInverse[A]`.

```
σInverse[a σ[0] + b σ[1] + c σ[2] + d σ[3]]
```

$$\frac{-a\,\sigma[0]+b\,\sigma[1]+c\,\sigma[2]+d\,\sigma[3]}{-a^2+b^2+c^2+d^2}$$

Singular matrix can not be inverted.

```
σInverse[σ[0] + σ[3]]
```

LinearSolve: Linear equation encountered that has no solution.

Inverse: Matrix σ[0] + σ[3] is singular.

```
σInverse[σ[0] + σ[3]]
```

- The **square root operator** $(A \to A^{1/2})$ can also be obtained via `σSqrt[A]`.

```
σSqrt[a σ[0] + b σ[1]]
```

$$\left(\frac{\sqrt{a-b}}{2}+\frac{\sqrt{a+b}}{2}\right)\sigma[0]+\left(-\frac{\sqrt{a-b}}{2}+\frac{\sqrt{a+b}}{2}\right)\sigma[1]$$

## Determinant

**Determinant** of an operator can be computed as

```
σDet[b σ[1, 3] + c σ[2, 2]]
```

$$b^4-2\,b^2\,c^2+c^4$$

For large matrix, the algorithm is faster than the ordinary symbolic determinant algorithm, if the matrix has simple decomposition in terms of Pauli matrices.

```
σDet[b σ[0, 0, 0, 0, 0, 0, 0, 0, 0] + c σ[1, 3, 2, 2, 3, 1, 3, 2, 1]]
```

$$\left(b^2-c^2\right)^{256}$$

## Operator Exp and Log

- **Operator exponential** $(A \to e^A)$

```
σExp[ⅈ ϕ σ[3, 2]]
```

```
Cos[ϕ] σ[0, 0] + ⅈ Sin[ϕ] σ[3, 2]
```

- **Operator logarithm** $(A \to \ln A)$

```
σLog[(σ[0, 0] + ⅈ Sqrt[3] σ[1, 3]) / 2]
```

$\frac{1}{3}\, \mathbb{i}\, \pi\, \sigma[1, 3]$

# Clifford Algebra and Lie Algebra

## Commutators

- **Commutator** $[A, B]$

```
Commutator[σ[1], σ[3]]
```

```
-2 ⅈ σ[2]
```

- **Anticommutator** $\{A, B\}$

```
Anticommutator[σ[3, 1], σ[2, 2]]
```

```
2 σ[1, 3]
```

## Clifford Algebra

- `Cl[n]` provides a choice of the generators of **complex Clifford algebra** $C\ell_n$.

$$\forall\ i, j = 1, \ldots, n : \{\gamma_i, \gamma_j\} = 2\,\delta_{ij}. \tag{8}$$

```
Cl[4]
```

```
{σ[1, 0], σ[2, 0], σ[3, 1], σ[3, 2]}
```

- `Cl[p,q]` provides a choice of the generators of **real Clifford algebra** $C\ell_{p,q}$.

$$\begin{aligned} &\forall\ i, j = 1, \ldots, p+q : \{\gamma_i, \gamma_j\} = 2\,\delta_{ij}, \\ &\forall\ i = 1, \ldots, p : \gamma_i^{\mathrm{T}} = \gamma_i, \\ &\forall\ i = p+1, \ldots, p+q : \gamma_i^{\mathrm{T}} = -\gamma_i. \end{aligned} \tag{9}$$

```
Cl[2, 3]
```

```
{σ[1, 0, 0], σ[3, 1, 0], σ[2, 0, 0], σ[3, 2, 0], σ[3, 3, 2]}
```

### Lie Algebra

`LieAlgebra[{g1,g2,…}]` completes the **Lie algebra** generators

```
LieAlgebra[{σ[1, 2], σ[3, 0]}]
{σ[1, 2], σ[3, 0], σ[2, 2]}
```

## Operator Transformations

### Basis Gates

- `C4[A]` gives $C_4$ rotation generated by a Pauli operator $\sigma^{[\mu]}$

$$e^{\frac{i\pi}{4}\sigma^{[\mu]}} \equiv \frac{1 + i\,\sigma^{[\mu]}}{\sqrt{2}}. \tag{10}$$

```
C4[σ[2, 3]]
```

$$\frac{\sigma[0, 0] + i\,\sigma[2, 3]}{\sqrt{2}}$$

- `Swap[σ[0,…,0,_,0,…,0,_,0,…,0]]` gives the swap operator that exchange the two qubits masked by _.

```
Swap[σ[0, _, 0, _]]
```

$$\frac{1}{2}\,(\sigma[0, 0, 0, 0] + \sigma[0, 1, 0, 1] + \sigma[0, 2, 0, 2] + \sigma[0, 3, 0, 3])$$

- `Hadamard[σ[0,…,0,_,0,…,0]]` gives the Hadamard gate acting on the single qubit masked by _.

```
Hadamard[σ[0, 0, _, 0]]
```

$$\frac{\sigma[0, 0, 1, 0] + \sigma[0, 0, 3, 0]}{\sqrt{2}}$$

- `Controlled[σ[…,μ,_,ν,…]]` gives the control gate, which implements $\sigma^{[\dots\mu 0\nu\dots]}$ controlled by the qubit masked by _.

```
Controlled[σ[_, 1, 2]]
```

$$\frac{1}{2}\,(\sigma[0, 0, 0] + \sigma[0, 1, 2] + \sigma[3, 0, 0] - \sigma[3, 1, 2])$$

### Transformations

Three types of transformations are defined:

- `OrthogonalTransform[O]` represents the **orthogonal** transformation

$$A \to O^{T} A\, O. \qquad (11)$$

- `UnitaryTransform[O]` represents the **unitary** transformation

$$A \to O^{\dagger} A\, O. \qquad (12)$$

- `ConjugateTransform[O]` represents the **conjugate** transformation

$$A \to O^{-1} A\, O. \qquad (13)$$

They can be implemented as

```
OrthogonalTransform[C4[σ[2]]][σ[3]]
```

```
σ[1]
```

or can be applied to a list of operators

```
UnitaryTransform[Controlled[σ[_, 1]]] /@ {σ[1, 0], σ[3, 0], σ[0, 1], σ[0, 3]}
```

```
{σ[1, 1], σ[3, 0], σ[0, 1], σ[3, 3]}
```

It is convenient to use `AssociationMap` to view the transformation

```
AssociationMap[UnitaryTransform[Hadamard[σ[_, 0]]],
 {σ[3, 1], σ[3, 2], σ[3, 3], σ[1, 0], σ[2, 0]}]
```

```
<|σ[3, 1] → σ[1, 1], σ[3, 2] → σ[1, 2],
 σ[3, 3] → σ[1, 3], σ[1, 0] → σ[3, 0], σ[2, 0] → -σ[2, 0]|>
```

# Pauli Operator Selection

## Boolean Functions

The following Boolean function are useful in setting selection criterion.

- **Commutation relation**

```
CommuteQ[σ[1, 2], σ[3, 1]]
```

```
True
```

```
AnticommuteQ[σ[1], σ[3]]
```

```
True
```

- **Symmetry condition**

```
SymmetricQ[σ[2]]
```

```
False
```

```
AntisymmetricQ[σ[2]]
```

```
True
```

## Pauli Select

`σSelect[{criterion,…},n]` selects the Pauli operators that satisfy given criterion. Number of qubits can be specified by $n$.

```
σSelect[AnticommuteQ /@ {σ[0, 2], σ[1, 1]}]
```

```
{σ[0, 3], σ[1, 3], σ[2, 1], σ[3, 1]}
```

Without any criterion, all Pauli basis are returned

```
σSelect[{}, 2]
```

```
{σ[0, 0], σ[0, 1], σ[0, 2], σ[0, 3], σ[1, 0], σ[1, 1], σ[1, 2], σ[1, 3],
 σ[2, 0], σ[2, 1], σ[2, 2], σ[2, 3], σ[3, 0], σ[3, 1], σ[3, 2], σ[3, 3]}
```

# LoopIntegrate Package

## Overview

The package `LoopIntegrate` provides the following functions:

```
? LoopIntegrate`*
```

LoopIntegrate`

| | | |
|---|---|---|
| DimensionRegularize | LeviCivitaEpsilon | MomentumIntegrate |
| FeynmanParameterize | Loop | MomentumShift |
| Index | LoopIntegrate | ParameterReduce |
| IntegrandInformation | LoopReduce | |

## Loop Integral and Dimensional Regularization

## Loop Object

The central object of the package is called `Loop`. It is a symbolic container of the data that defines a loop integral in general dimensions. Its structure is like

```
Loop[expr,{p1,…},D,x,nx]
```

- `expr` - the integrand of the loop integral.
- `{p1,…}` - a list of the integral variables, specifying the momenta to be integrated over.
- `D` - the symbol for the spacetime dimension. Better not specify an integer dimension here, unless the integral does not need to be regularize. (One should use `DimensionRegularize` to properly calculate loop integrals in specific dimensions).
- `x` - the symbol for Feynman parameter.
- `nx` - the number of Feynman parameter.

`D`, `x` and `nx` are optional. To create a `Loop` object, typically one just need to specify the integrand and variables.

The `Loop` object is represented as a loop integral.

```
Loop[1 / (p^2 + m^2), p]
```

$$\int \frac{d^{D} p}{(2\pi)^{D}} \frac{1}{m^2 + p^2}$$

```
Loop[Index[k + 2 p, μ] / ((k + p)^2 q^3), {p, q}]
```

$$\int \frac{d^{D} p}{(2\pi)^{D}} \int \frac{d^{D} q}{(2\pi)^{D}} \frac{k_{\mu} + 2\, p_{\mu}}{(k+p)^2\, q^3}$$

## Momentum Indexing

The momentum can be indexed by `Index`.

```
Index[k, μ]
```

$k_{\mu}$

Index distributes into the linear combination of momenta.

```
Index[(k + 2 p) / 3, μ]
```

$$\frac{k_{\mu} + 2\, p_{\mu}}{3}$$

For a concrete vector with a specific index, it extracts the component.

```
Index[{k1, k2, k3}, 2]
```

k2

## Feynman Parameterization

`FeynmanParameterize` analyzes the denominator of the integrand and combines the denominator factors. The Feynman parameters are introduced automatically.

```
# → FeynmanParameterize[#] &@Loop[Index[k + q, μ] / ((k + q) ^2 q^3), q]
```

$$\int \frac{d^{D}q}{(2\pi)^{D}} \frac{k_{\mu} + q_{\mu}}{q^{3}\,(k+q)^{2}} \to \frac{3}{2}\left(\int_{\triangle} d^{2}x \int \frac{d^{D}q}{(2\pi)^{D}} \frac{(k_{\mu}+q_{\mu})\sqrt{x_{1}}}{\left(q^{2}x_{1} + (k+q)^{2}x_{2}\right)^{5/2}}\right)$$

The small $\triangle$ of $\int_{\triangle} d^{n}x$ reminds that the integral must be performed under the constraint $\sum_{i=1}^{n} x_i = 1$ (which is a codimension-1 simplex $\triangle$). Demonstrate the Feynman parameterization formula for multiple factors of the denominator.

```
FeynmanParameterize[Loop[1 / ((p^2 + Δ1) ^a1 (p^2 + Δ2) ^a2 (p^2 + Δ3) ^a3), p]]
```

$$\left(\text{Gamma}[a1 + a2 + a3]\left(\int_{\triangle} d^{3}x \int \frac{d^{D}p}{(2\pi)^{D}} x_{1}^{-1+a1} x_{2}^{-1+a2} x_{3}^{-1+a3}\left(\left(p^{2} + \triangle 1\right)x_{1} + \left(p^{2} + \triangle 2\right)x_{2} + \left(p^{2} + \triangle 3\right)x_{3}\right)^{-a1-a2-a3}\right)\right)\Big/ (\text{Gamma}[a1]\,\text{Gamma}[a2]\,\text{Gamma}[a3])$$

## ■ Loop Reduction

`LoopReduce` applies the momentum integral formulas to reduce the loop integral. The momenta are integrated out one by one. For each momentum integral, a momentum shift is first performed to transform the denominator to the standard form. Then the integral formulas are applied. Delta symbols $\delta_{\mu\nu}$ ... are generated automatically under Wick contraction.

```
LoopReduce[Loop[1 / (p^2 + Δ) ^a, p]]
```

$$\frac{2^{-D}\pi^{-D/2}\triangle^{-a+\frac{D}{2}}\,\text{Gamma}\left[a - \frac{D}{2}\right]}{\text{Gamma}[a]}$$

```
LoopReduce[Loop[Index[p, μ] × Index[p, ν] / (p^2 + Δ) ^a, p]]
```

$$\frac{2^{-1-D}\pi^{-D/2}\triangle^{1-a+\frac{D}{2}}\,\text{Gamma}\left[-1 + a - \frac{D}{2}\right]\delta_{\mu,\nu}}{\text{Gamma}[a]}$$

```
LoopReduce[Loop[Index[p, μ] × Index[p, ν] × Index[p, λ] × Index[p, κ] / (p^2 + Δ) ^a, p]]
```

$$\frac{2^{-2-D}\pi^{-D/2}\triangle^{2-a+\frac{D}{2}}\,\text{Gamma}\left[-2 + a - \frac{D}{2}\right](\delta_{\kappa,\nu}\delta_{\lambda,\mu} + \delta_{\kappa,\mu}\delta_{\lambda,\nu} + \delta_{\kappa,\lambda}\delta_{\mu,\nu})}{\text{Gamma}[a]}$$

```
LoopReduce[Loop[Index[p, μ] × Index[p, ν] ×
   Index[p, λ] × Index[p, κ] × Index[p, ρ] × Index[p, τ] / (p^2 + Δ)^a, p]]
```

$$\frac{1}{\text{Gamma}[a]} 2^{-3-D} \pi^{-D/2} \Delta^{3-a+\frac{D}{2}} \text{Gamma}\left[-3+a-\frac{D}{2}\right] (\delta_{\kappa,\tau} \delta_{\lambda,\rho} \delta_{\mu,\nu} + \delta_{\kappa,\rho} \delta_{\lambda,\tau} \delta_{\mu,\nu} + \delta_{\kappa,\tau} \delta_{\lambda,\nu} \delta_{\mu,\rho} + \delta_{\kappa,\nu} \delta_{\lambda,\tau} \delta_{\mu,\rho} + \delta_{\kappa,\rho} \delta_{\lambda,\nu} \delta_{\mu,\tau} + \delta_{\kappa,\nu} \delta_{\lambda,\rho} \delta_{\mu,\tau} + \delta_{\kappa,\tau} \delta_{\lambda,\mu} \delta_{\nu,\rho} + \delta_{\kappa,\mu} \delta_{\lambda,\tau} \delta_{\nu,\rho} + \delta_{\kappa,\lambda} \delta_{\mu,\tau} \delta_{\nu,\rho} + \delta_{\kappa,\rho} \delta_{\lambda,\mu} \delta_{\nu,\tau} + \delta_{\kappa,\mu} \delta_{\lambda,\rho} \delta_{\nu,\tau} + \delta_{\kappa,\lambda} \delta_{\mu,\rho} \delta_{\nu,\tau} + \delta_{\kappa,\nu} \delta_{\lambda,\mu} \delta_{\rho,\tau} + \delta_{\kappa,\mu} \delta_{\lambda,\nu} \delta_{\rho,\tau} + \delta_{\kappa,\lambda} \delta_{\mu,\nu} \delta_{\rho,\tau})$$

Repeated indices are assumed to be summed over, so $\delta_{\mu\mu} = D$. For example we can show

$$\int \frac{d^D p}{(2\pi)^D} \frac{p_\mu p_\mu}{\left(p^2+\Delta\right)^a} = \int \frac{d^D p}{(2\pi)^D} \frac{p^2}{\left(p^2+\Delta\right)^a}. \tag{14}$$

```
LoopReduce[Loop[Index[p, μ] × Index[p, μ] / (p^2 + Δ)^a, p]]
```

$$\frac{2^{-1-D} \pi^{-D/2} D \Delta^{1-a+\frac{D}{2}} \text{Gamma}\left[-1+a-\frac{D}{2}\right]}{\text{Gamma}[a]}$$

```
LoopReduce[Loop[p^2 / (p^2 + Δ)^a, p]]
```

$$\frac{2^{-D} \pi^{-D/2} \Delta^{1-a+\frac{D}{2}} \text{Gamma}\left[-1+a-\frac{D}{2}\right]}{\text{Gamma}[-1+a]} - \frac{2^{-D} \pi^{-D/2} \Delta^{1-a+\frac{D}{2}} \text{Gamma}\left[a-\frac{D}{2}\right]}{\text{Gamma}[a]}$$

```
FullSimplify[% - %%]
```

```
0
```

In general, Feynman parameters will be generated

```
LoopReduce[Loop[1 / ((k + p)^2 p), p]]
```

$$2^{-D} \pi^{-\frac{1}{2}-\frac{D}{2}} \text{Gamma}\left[\frac{3}{2}-\frac{D}{2}\right] \left(\int_{\Delta} d^2 x \frac{\left(-k^2 (-1+x_2) x_2\right)^{-\frac{3}{2}+\frac{D}{2}}}{\sqrt{x_1}}\right)$$

## ▪ Dimensional Regularization

Explicitly specifying the dimension in `Loop` may leads to divergent result.

```
LoopReduce[Loop[1 / (p^2 + m^2), p, 2]]
```

```
ComplexInfinity
```

One should first perform the loop integral with a symbolic dimension, and then use the dimensional regularization to find the regular part of the integral.

```
DimensionRegularize[LoopReduce[Loop[1 / (p^2 + m^2), p, D]], D → 2]
```

$$-\frac{\text{Log}\left[\frac{m}{\Lambda}\right]}{2\pi}$$

## ▪ Parameter Reduction

Finally Feynman parameters can be integrated over by parameter reduction. For example,

```
ParameterReduce[Loop[Indexed[x, 1]^2 × Indexed[x, 2], {}, D, x, 2]]
```

$$\frac{1}{12}$$

It is equivalent to the following integral,

```
Integrate[Indexed[x, 1]^2 × Indexed[x, 2] × DiracDelta[Sum[Indexed[x, i], {i, 2}] - 1],
 x ∈ Rectangle[{0, 0}, {1, 1}]]
```

$$\frac{1}{12}$$

Apply to loop integral results.

```
ParameterReduce[
 DimensionRegularize[LoopReduce[Loop[Index[k + p, μ] / ((k + p)^2 p), p, D]], D → 3]]
```

$$-\frac{k_\mu \,\text{Log}\left[\frac{k}{\Lambda}\right]}{6\pi^2}$$

## ▪ Function LoopIntegrate

The package also provide the high-level function `LoopIntegrate` that automates the above procedures of Feynman parameterization, momentum shift and integration, dimensional regularization and parameter reduction.

```
LoopIntegrate[expr,p]
LoopIntegrate[expr,p,D]
LoopIntegrate[expr,{p1,p2,…},D]
```

It can be used with either general or specific dimensions.

```
FullSimplify@LoopIntegrate[Index[k + p, μ] / ((k + p)^2 p), p]
```

$$\frac{4^{1-D}\left(k^2\right)^{\frac{1}{2}(-3+D)}\pi^{-D/2}\,\text{Gamma}\left[\frac{3-D}{2}\right]\text{Gamma}[-1+D]\,k_\mu}{\text{Gamma}\left[-\frac{1}{2}+D\right]}$$

If a specific dimension is given, dimensional regularization will be applied to obtain the regular part of the integral.

```
LoopIntegrate[Index[k + p, μ] / ((k + p) ^2 p), p, 3]
```

$$-\frac{k_\mu \, \mathrm{Log}\left[\frac{k}{\Lambda}\right]}{6\,\pi^2}$$

# MatsubaraSum Package

## Overview

The package `MatsubaraSum` provides the following functions:

```
? MatsubaraSum`*
```

MatsubaraSum`

| | | |
|---|---|---|
| Bosonic | Fermionic | ZeroTemperatureLimit |
| ControlledPlane | MatsubaraSum | |
| DistributionFunction | StatisticalSign | |

## Single Frequency Summation

### Basic Summation

`MatsubaraSum[f(z), z]` evaluates the following summation

$$\frac{1}{\beta}\sum_z f(z).$$

where $f(z)$ is a function of the Matsubara frequency $z = i\,\omega_n$ and $\omega_n$ is taken from either one of the following sets

$$n \in \mathbb{Z} : \omega_n = \begin{cases} \frac{2\pi}{\beta}\, n & \text{Bosonic,} \\ \frac{2\pi}{\beta}\left(n + \frac{1}{2}\right) & \text{Fermionic.} \end{cases}$$

In the current version of the package, it is assumed that $f(z)$ is a fraction, whose denominator is a polynomial of $z$ and whose numerator can be arbitrary.

Without specifying the type of the Matsubara frequency $z$, the general result is returned.

```
MatsubaraSum[1 / (z - ϵ), z]
```

$$-n_{\eta_z}[\epsilon]\,\eta_z$$

$\eta_z = \pm 1$ is the statistical sign of the frequency $z$:

$$\eta_z = \begin{cases} +1 & \text{if } z \in \text{Bosonic,} \\ -1 & \text{if } z \in \text{Fermionic.} \end{cases}$$

$n_\eta(\epsilon)$ represents the distribution function

$$n_\eta(\epsilon) = \frac{1}{e^{\beta\,\epsilon} - \eta} = \begin{cases} n_B(\epsilon) & \eta = +1, \\ n_F(\epsilon) & \eta = -1. \end{cases}$$

```
MatsubaraSum[1 / (z - ϵ) ^2, z]
```

$-\eta_z\, n'_{\eta_z}[\epsilon]$

```
MatsubaraSum[1 / (z - ϵ) ^3, z]
```

$-\frac{1}{2}\,\eta_z\, n''_{\eta_z}[\epsilon]$

```
MatsubaraSum[1 / (z - ϵ) ^4, z]
```

$-\frac{1}{6}\,\eta_z\, n^{(3)}_{\eta_z}[\epsilon]$

$n'_\eta(\epsilon)$, $n''_\eta(\epsilon)$ and $n^{(k)}_\eta(\epsilon)$ represent the 1st, 2nd and $k$th derivatives of the distribution function respectively.

## Specify Frequency Type

One can specify the Matsubara frequency type by

$z \in$ `Bosonic`: asserts $z = i\,\omega_n$ to be bosonic,

$z \in$ `Fermionic`: asserts $z = i\,\omega_n$ to be fermionic.

The symbol ∈ can be entered as ESC el ESC.

```
MatsubaraSum[1 / (z - ϵ), z ∈ Fermionic]
```

$n_F[\epsilon]$

```
MatsubaraSum[1 / (z - ϵ), z ∈ Bosonic]
```

$-n_B[\epsilon]$

Another way to specify the frequency type is to use the `Assumptions` option.

```
MatsubaraSum[1 / (z - ϵ), z, Assumptions → z ∈ Fermionic]
```

$n_F[\epsilon]$

```
MatsubaraSum[1 / (z - ϵ), z, Assumptions → z ∈ Bosonic]
```

$-n_B[\epsilon]$

The `Assumptions` option can also be use to specify the type of external frequencies (i.e. the Matsubara frequencies that will not be summed over).

```
MatsubaraSum[1 / (z1 - ϵ1) / (z1 - z2 - ϵ2), z1 ∈ Fermionic, Assumptions → z2 ∈ Bosonic]
```

$- \frac{n_F[\epsilon 1]}{z2 - \epsilon 1 + \epsilon 2} + \frac{n_F[\epsilon 2]}{z2 - \epsilon 1 + \epsilon 2}$

## Multiple Frequency Summation

Summation over multiple frequencies can be calculated by specifying more frequency variables to be summed over.

```
MatsubaraSum[1 / (z1 - ϵ1) / (z2 - ϵ2), z1, z2]
```

$n_{\eta_{z1}}[\epsilon 1]\; n_{\eta_{z2}}[\epsilon 2]\; \eta_{z1}\, \eta_{z2}$

Each frequency variable can be assigned a frequency type (either `Bosonic` or `Fermionic`).

```
MatsubaraSum[1 / (z1 - ϵ1) / (z2 - ϵ2), z1 ∈ Bosonic, z2 ∈ Fermionic]
```

$-n_F[\epsilon 2]\; n_B[\epsilon 1]$

The external frequency type can be specified by `Assumptions`.

```
MatsubaraSum[1 / (z1 - ϵ1) / (z2 - ϵ2) / (z1 + z2 - z - ϵ3),
  z1 ∈ Fermionic, z2 ∈ Fermionic, Assumptions → z ∈ Fermionic]
```

$- \frac{n_F[\epsilon 2]\; (n_F[\epsilon 1] - n_F[\epsilon 3])}{z - \epsilon 1 - \epsilon 2 + \epsilon 3} - \frac{(n_F[\epsilon 1] - n_F[\epsilon 3])\; n_B[-\epsilon 1 + \epsilon 3]}{z - \epsilon 1 - \epsilon 2 + \epsilon 3}$

## Miscellaneous

### Simplify Distribution Functions

The package knowns about the properties of distribution functions, such as

$n_\eta(-x) = -\eta - n_\eta(x),$

$n_\eta^{(k)}(-x) = -(-1)^k\, n_\eta^{(k)}(x),$

$2\, n_B(x)\, n_F(x) = n_B(x) - n_F(x).$

Using these, expressions of distribution functions can be simplified.

```
FullSimplify[1 - nF[-ϵ]]
```

$n_F[\epsilon]$

```
FullSimplify[(1 + 2 nB[ϵ]) (1 - 2 nF[ϵ])]
```

1

Generic distribution functions and statistical signs can be reduced by making explicit assumptions about the types of frequency variables.

```
Assuming[z1 ∈ Bosonic && z2 ∈ Fermionic, n_{η_{z1}}[ϵ1] n_{η_{z2}}[ϵ2] η_{z1} η_{z2}]
```

$-n_F[\epsilon 2]\, n_B[\epsilon 1]$

```
Assuming[z ∈ Fermionic, n_F[z + ϵ]]
```

$-n_B[\epsilon]$

## Zero Temperature Limit

The distribution function can further be expressed in terms of Heaviside Θ function in the zero temperature limit.

$$\Theta(\epsilon) = \begin{cases} 1 & \epsilon > 0, \\ 0 & \epsilon < 0. \end{cases} \tag{15}$$

```
ZeroTemperatureLimit[n_F[ϵ]]
```

1 - HeavisideTheta[ϵ]

```
ZeroTemperatureLimit[n_B[ϵ]]
```

-1 + HeavisideTheta[ϵ]

## Convergence Control

If the Matsubara summation does not converge, the result will depend on the regularization. We regularize the summation by $e^{-\delta z}$ factor with $\delta \to 0$,

$$\frac{1}{\beta} \sum_z f(z)\, e^{-\delta z}.$$

If $\delta = 0_+$ ($\delta = 0_-$) the convergence is controlled on the right(left)-half plane. This choice can be set by the option `ControlledPlane`, which can take `Right` (default) or `Left`.

```
Table[FullSimplify@MatsubaraSum[1 / (z - ϵ), z ∈ Fermionic, ControlledPlane → cp],
 {cp, {Right, Left, All}}]
```

$\{n_F[\epsilon],\ -1 + n_F[\epsilon],\ -\frac{1}{2} + n_F[\epsilon]\}$

For convergent summation, the result does not depend on regularization.

```
Table[FullSimplify@MatsubaraSum[1 / (z - ϵ) ^2, z ∈ Fermionic, ControlledPlane → cp],
 {cp, {Right, Left, All}}]
```

$\{n'_F[\epsilon],\ n'_F[\epsilon],\ n'_F[\epsilon]\}$