SeaSyde 1.0.0

T.Armadillo, R.Bonciani, S.Devoto, N.Rana, A.Vicini

Release date: May 2022

The SeaSyde package (Series Expansion Approach for SYstems of Differential Equations) can be imported in Mathematica using the command `Get[...]`, i.e. through `<< SeaSyde.m`. Note that it has been developed and tested on Mathematica 12.0, no previous version of Mathematica has been tested. Below we present all different functions and their functionalities.

- `CurrentConfiguration[]`

  It returns the current configuration of the package. The configuration parameters can be modified using the function `UpdateConfiguration`.

- `UpdateConfiguration[NewConfig_]`

  `NewConfig` must be a `List` whose elements are replacement rules `NameParameter -> NewValue`. See Table 1 for a complete overview on all the parameters that the user can modify.

- `ReadFrom[FilePath_]`

  Utility function which reads from the specified path and returns the content of the file.

- `SetSystemOfDifferentialEquation[System_, BCs_, MIs_, Variables_, PointBC_, Param_:{}]`

  It sets all the internal variables of the package and prepare the system of differential equations. It receives in input

  - `System_`: the system of differential equations. The system equations must be given in triangular form and it must be ordered so that, order by order in ε, every equation contains only MIs from previous equations. If there are multiple kinematic variables, for example `x` and `y`, the first n equations in `System` must be the ones with respect to `x`, while the last n the ones with respect to `y`, where n is the number of MIs. The equations can be given expanded in ε or in a closed form in ε.
    Example:

```
{  B1^(1,0)[x,y] == 0 ,
    B2^(1,0)[x,y] == (- 1/x - ε/x)B2[x,y] - ε B1[x,y] / x,
```

```
B₁⁽⁰,¹⁾[x,y] == 0 ,
B₂⁽⁰,¹⁾[x,y] == 0 }
```

- BCs_: the boundary conditions for the given equation. They can be given in a closed form in ε or as a series. They can also be given as an asymptotic limit. They can be exact or floating numbers, in this case make sure that the precision of the boundary condition is sufficient for your final precision goal.
  Example:

```
{  B1[1,1] == - 1/32 + ε/32 + ε^2( - 1/32 + π^2/96 ),
   B2[1,1] == - ε Log[2]/16 + ε^2(-π^2/192 + Log[2]^2/8) }
```

- MIs_: the list of master integrals, as they appear in the equations.
  Example:

```
{  B1[x,y], B2[x,y] }
```

- Variables_: the list of variables appear in the equations, together with their Feynman prescriptions.
  Example:

```
{  x - I δ, y + I δ }
```

- PointBC_: the point in the phase-space in which the boundary conditions are imposed.
  Example:

```
{ 1, 1 }
```

- Param_: this is an optional parameter. Some equations might contain some external parameters, for example some masses Mw, Mz. This substitutions are performed before solving the system.
  Example:

```
{ MW -> 80.38, MZ -> 91.19 }
```

- GetSystemOfDifferentialEquation[] and GetSystemOfDifferentialEquationExpanded[]

  They return the system of differential equations before and after it has been expanded in ε. These functions can be used to check if everything has been set correctly.

- SolveSystem[Variable_]

  It solves the system of differential equations with respect to the kinematic variable Variable. The series solution in centred in the point where the boundary conditions are imposed. After solving the system of differential equations, it is possible to obtain the solution through Solution[], SolutionValue[] or SolutionTable[].

- GetPoint[]

  It returns the current point in which the boundary conditions are imposed.

- `TransportVariable[Variable_, Destination_, Line_:{}]`

  It transports the boundary conditions for the variable `Variable` from the current point to `Destination`. After transporting the boundary conditions, the point in which the boundary conditions are imposed is updated to `Destination`. The `Line` parameter is optional. If the user is not satisfied by the path automatically chosen by the package can use their own. The `Line` object must be created with `CreateLine`

- `CreateLine[Points_]`

  It returns a line object that can be used in `TransportVariable`

- `TransportBoundaryConditions[PhaseSpacePoint_]`

  `PhaseSpacePoint` must be a `List` whose length is given by the number of kinematic variables. Its first element must be the final value for the first variable, its second element the value for the second variable, and so on. The order of the kinematic variables is the same passed as an input in`SetSystemOfDifferentialEquation`. After transporting the boundary conditions, the point in which they are imposed is updated to `PhaseSpacePoint`.

- `Solution[]`

  It returns the series solution in the current point. The coefficients of the series are given with `InternalWorkingPrecision` digits. The result is given as a `List` and every MI as a Laurent series in ε.

- `SolutionValue[]`

  It returns the value of the MIs, in the centre of the series, as a Laurent expansion in ε. The coefficients of the ε-series are given with `InternalWorkingPrecision` digits.

- `SolutionValue[]`

  It returns the value of the MIs, in the centre of the series, as a `List` going from the minimum to the maximum order in ε. The coefficients of the ε-series are given with `InternalWorkingPrecision` digits.

- `CheckSingularities[]`

  It checks whether the singularities are logarithmic, i.e. if they develop a branch-cut. The check is done by performing a path round the singularity and checking if the solution has developed, or not, an imaginary part for at least one of the coefficients. If it is not, it means that in doing so we crossed a branch-cut and, hence, the singularity is logarithmic. If this function is not called, `SeaSyde` consider every singularity as logarithmic, and the analytic continuation is still possible. However, if the user is planning to make an intensive use of the function `TransportBoundaryConditions`, e.g. create a numerical grid, knowing the position of non-logarithmic singularities might allow for more direct paths in the complex plane. The output of

`CheckSingularities` can be passed back in the `SafeSingularities` and `LogarithmicSingularities` parameters for future runs.

- `CreateGraph[MI_, EpsOrder_, Left_, Right_, OtherFunctions_:{}]`

  Draws a `ReImPlot` of the solution of order `EpsOrder` for the master `MI`. The graph runs from `Left` to `Right`. The argument `OtherFunctions` may contains other functions to be plotted in the same graph.

In the `Example/` folder of the GitHub repository of `SeaSyde`, the user can find working examples to play with.

NameParameter	Value type	Default	Description
`EpsilonOrder`	`Integer`	2	The maximum order in the dimensional regulator ε at which the system is expanded. Note that the minimum order is determined by the boundary conditions, i.e. if they contain a term $1/\varepsilon^2$ the minimum order will be -2.
`ExpansionOrder`	`Integer`	50	The maximum order in the kinematic variable at which the solution is expanded.
`InternalWorkingPrecision`	`Integer`	250	Specifies the number of digits that are used in internal calculations. If it is too high, the execution will require more time and space, if it is too low, we may face rounding errors.
`LogarithmicExpansion`	`Bool`	`False`	It specifies which method to use for transporting boundary conditions. If it is set to `True`, SeaSyde will expand also on top of singularities.
`RadiusOfConvergence`	`Integer`	2	It controls how fast we move at every expansion. If `RadiusOfConvergence` is n, and the maximum radius of convergence of the solution is r, then the new point will be distant r/n from the center of the series. Note that r is determined internally at every step, based on the position of singularities.
`LogarithmicSingularities`	`List`	{}	The user can explicitly state which singularities are of Logarithmic type, i.e. develop a branch-cut, and which ones are not. This might speed up the evaluation of numerical grids since it allows more direct paths in the phase-space. The format in which the singular points are passed must the one returned by `CheckSingularities[]`.
`SafeSingularities`	`List`	{}	Same as `LogarithmicSingularities`.

Table 1: All the parameters that can be modified by the user.