Amplitude Term Hunt

1 INTRODUCTION

The purpose of this project is to design code that can help the user find specific sequence of terms in long expressions derived from string amplitudes.

String theory is considered the best possible candidate so far to constitute a theory of everything. Within the same formalism, all the fundamental forces, including gravity, [1] are quantised. When some specific limit behaviour is approached (i.e. the desired energetic scale to describe is low compared to some parameters[2] in the general definition) one can recover semi-classical description of well known fields (i.e Gravity, Electromagnetism, etc) .

Equipped with such powerful theory, we are interested in describing the dynamics of the Universe within that framework. [3] The discipline in charge of this study from a stringy perspective is String Cosmology.

The main aim of String Cosmology is to take String Theory and perform a gentle transition to derive a semi-classical description of usual cosmology. In principle, this can be done at tree level (i.e. Just looking at the most relevant contributions of the interaction between strings) but... not always, as it is the concerning case. Here, one needs to account for sub-leading corrections (i.e. not so relevant, yet not neglegible contributions) to achieve one of the fundamental values of classical cosmology to be non 0. Λ, the cosmological constant. This constant is a fixed[4] positive value in Eintein's General Relativity equations to account for **the expansion of the Universe.**

These aforamentioned corrections can be obtained from all possible interaction of different strings at sub-leading order. In the case we are interested in, it results to be around 7000 terms + all possible permutations for some values inside them. At this point, we need to ask a computer to identify which terms will contribute to our task, imposing a set of rules to localise them in such a long list. This is what the package of this repository aim to achieve.

[1] Gravity is the most ellusive force to quantise. Plenty of problems arise when quantising from the classical description.

[2] The scale of the string $l_s \to 0$, which is analogous to not have enough energy to resolve how to atoms interact and we just see two little balls clashing.

[3] Classical cosmology is the disciple which studies this. It makes use of Einstein's General Relativity to describe the evolution of the Universe. Up to some extend, it gives the most accurate description we have of it.

[4] Or maybe not. This is hot topic research right now.

2 Description of Problem

The starting point is the following amplitude:

$$\begin{aligned}\mathscr{A}(1,2,3) \sim & \sin[\pi s_4]\ A(1,3,6,4,5,2) \\ & + \sin\left[\pi\left(\tfrac{s_1}{2} - \tfrac{s_3}{2} - \tfrac{s_5}{2} - s_4\right)\right] A(1,4,6,3,5,2) \\ & + \sin\left[\pi\left(-\tfrac{s_1}{2} + \tfrac{s_3}{2} - \tfrac{s_5}{2} - s_6\right)\right][A(1,3,4,6,2,5) + A(1,4,3,6,2,5)] \\ & + \left(\sin\left[\pi\left(\tfrac{s_1}{2} - \tfrac{s_3}{2} - \tfrac{s_5}{2}\right)\right] - \sin\left[\pi\left(\tfrac{s_1}{2} - \tfrac{s_3}{2} + \tfrac{s_5}{2}\right)\right]\right)[A(1,3,4,6,5,2) + A(1,4,3,6,5,2)],\end{aligned} \tag{1}$$

which describes the interaction of three closed strings in terms of a combination of the interactions of six open strings. Each $A(i,j,k,l,m,n)$ accounts specific permutations of these six strings in diagram of their interaction. The most possible general interaction is given in the file called *Long.csv*, inside *Code/ Sequences*. This file contains all possible contributions from string theory for six open strings over a given topology. From this long list, we are only interested on some terms with a specific given form as:

$$\epsilon_1.\epsilon_2\,\epsilon_3.\epsilon_4\,\epsilon_5.\epsilon_6, \tag{2}$$

where ϵ_i stands for the polarisation of the strings. Not only this form, but any other possible permutation of previous term is relevant for the aforamentioned computation. Furthermore, permutation rules affecting ϵ_i will affect other variables in the expression, which our code has to take account for.

3 Method

Hence, the method to hunt down only those relevant terms for eq (1) would be as follows:

1. Load the Amplitude expression into python from the *.csv* file.
2. Chop the string into different terms, separated by $\pm$ signs.
3. Run a function that identifies the desired expression (2). If this substring is part of the term, store that term in a new file.
4. Perform permutations, accounting for the transformation rules of other variables.
5. Repeat from step 1, but now saving only the desired terms that match the requirement $A(i,j,k,l,m,n)$
6. Perform for all combinations in eq (1)

4 Package description

This package is a combination of the following tools.

1. *Chopper*: This is a class that eats a file and has two methods.
   (a) *Split Monster* which will chop the string read from the file down to the desired step.
   (b) *Check Right Split* which proves that the previous splitting was right.
2. *Selector*: A class that inherits from *Chopper* all its method and adds two more:
   (a) *Looking for e general*: This methods loads the file, chop it down to terms in the polynomial and selects only those terms that match a specific requirement. In this case, only those terms with the desired polarisation 2.
   (b) *Looking for e specific*: This method is similar to previous one, but in this case the ordering of polarisation terms matter. It will spit out a file with only the chosen terms that match the requirement.

3. *Permutator*: Another class, in charge of permuting the terms. It has two more methods:
4. (a) *Replacement Dict Creator*: This method creates a dictionary of future replacements, based on an input sequence (i.e. 136452). Identifies each position in the sequence and assign them the right transformation terms for the polarisations 2 and their associated momenta.
   (b) *Permuting*: This method calls the previous one plus the *Replacetor* function. It will take the file, chop the string, replace terms according to the transformation rules described in the previous method and then spits out a file with the new permutations.

You can find further information about each class inside the package code.