THE UNIVERSITY
of EDINBURGH

Automatation of Decomposed Multi-Loop Feynman-Graphs

Tom Owen Rippon

Thesis submitted for the degree of Master of Philosophy

University of Edinburgh
2010

Abstract

This thesis is an investigation into the link between momentum space Feynman graph integrals under a henge decomposition and parametric representation of Feynman graph integrals under a sector decomposition. A program is then constructed in the language Maple based on the method of henge decomposition and Feynman parameterization as a potential method to automatize the calculation of multi-loop Feynman graph calculations. The methods of henge decomposition and sector decomposition are explored with all the necessary background detail required for the understanding of the methods explained.

Declaration

This thesis has been composed by myself, all work within is my own or work done in collaboration with Thomas Binoth and Anthony D. Kennedy. The link between the methods of henge decomposition and sector decomposition forms part of the paper

- Automating Renormalization of Quantum Field Theories, A.D. Kennedy, T. Binoth and T. Rippon, arXiv:0712.1016v1

and has been published in "Proceedings of the 2007 international workshop on Symbolic-numeric computation, London, Ontario, Canada, 18-27 (2007)".

The code presented in chapter 3 is based on the code of Tony Kennedy and has been largely modified and extended by myself.

Tom Owen Rippon October 22, 2010

Dedication

This thesis is dedicated to Dr Thomas Binoth whose tireless help and encouragement kept me on the straight and narrow despite all of my best intentions.

Acknowledgements

I would like to thank Thomas Binoth and Tony Kennedy for accepting me as a student, for their patient explanations and invaluable help throughout this thesis.

I would like to thank my office–mates Ben O'Leary, Dave Antonio, Simone Marzani, Eike Müller, and Maria Ubiali for an intelligent and friendly working enviroment.

I would like to thank my colleagues Thomas Reiter, George Burton, Paul Cooney and Chris Kelly for the often useful discussions.

A special thank you to my long suffering house–mates Liz Longden and Ken Sowamber-Rubiola.

And last but by no means least, I would like to say thank you to all my family and friends without whom I would be lost.

Contents

Chapter 1

Introduction

Sub-atomic physics is currently best described using Lorentz invariant quantum field theories (hereafter abbreviated as QFTs) with gauge mediated interactions. The most successful of these is the Standard Model (SM) which is a non-abelian gauge theory with the symmetry group $SU(3)_C \times SU(2)_L \times U(1)$.

The interactions associated with the gauge group $SU(3)_C$ correspond with the Strong interaction, that is the force which binds partons into hadrons. The theory of the strong interaction is called Quantum Chromodynamics (QCD) due to the charge carried by the partons being known as colour [1, 2, 3]. Two of the most famous features of QCD are confinement and asymptotic freedom, those being an explanation for why Quarks are never observed outside of their parent Hadrons and the reason why the partons react weakly at high energy scales. The asymptotic freedom of QCD was discovered in 1973 by D. Gross and F. Wilczek [4] and D. Politzer [5], this discovery was very important as it allowed the use of perturbative techniques at high energies. The colour confinement of QCD has been confirmed in lattice QCD but as yet does not a have a formal proof.

The remaining $SU(2)_L \times U(1)$ gauge group gives rise to the Electroweak Standard Model (EW) [6, 7, 8]. The interactions mediated by the group $SU(2)_L$ means that left and right handed fermions have different couplings and thus a mass term for the fermions is forbidden. The presence of massive gauge bosons ($W^{\pm}$ and Z) within the SM means that the $SU(2)_L$ symmetry has to be spontaneously broken to give account of the masses. This spontaneous symmetry breaking is the famous Higgs mechanism [9, 10, 11]. The first successful realistic QFT Quantum Electrodynamics (QED)[12, 13, 14, 15, 16, 17] is a subset of EW.

So far the phenomena calculated from the SM has been consistent with the experimental findings [18, 19], with only the Higgs sector, predicted by the SM, so far being

experimentally unverified. Fig 1.1 shows the deviation between theory and experiment of 18 observables from the SM taking into account the experimental uncertainty. As can be seen, all data points are within a 3σ interval and therefore are considered consistant.

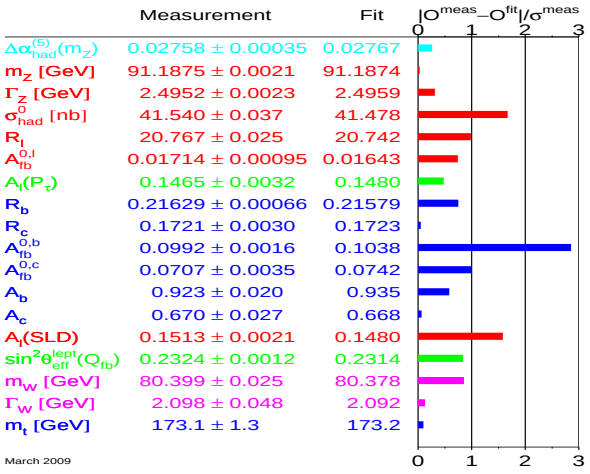


Figure 1.1: Results from LEP and Tevetron of electroweak data from winter 2009 [20]

Despite all this success, the SM can only be at best an effective-theory as the SM does not include gravity or have any explanation for the dark matter and dark energy problems. Dark matter is an especially compelling reason for extensions to the standard model as there is a experimental evidence for its existence [21, 22]. The lack of any explanation for the values of mixing angles, the three generation of particles leads many to find the SM unsatisfactory.

The current collider located at CERN, the Large Hadron Collider (LHC), although primarily a discovery machine, will also push the experimental precision of existing observables.Therefore, in order to test the limits of the SM there is a need for increasingly accurate phenomenological results which necessitates the calculation of multi-loop and multi-leg calculations.

As a result of this in 2001, the Run 2 Monte Carlo workshop in Fermilab [23] presented a list, Table 1.2, of the most desirable processes to be calculated from the experimenters' point of view. In 2005, at the Les Houches Physics at Tev Colliders workshop [24] this '*Experimenters' Wish List*' was addressed with the release of a

list of processes which the phenomenological community felt was more realistic given the then current calculational methods and the timescales available [25]. It was felt that many of the multi-particle processes withmany particles in the final state such as $WWW + b\bar{b} + 3$ jets were unlikely to be calculated at NLO level at any point soon. This list was further revised and updated at Les Houches Physics at TeV Colliders workshop in 2007 [26, 27].

Run II Monte Carlo Workshop, April 2001

Single boson	Diboson	Triboson	Heavy flavour
$W + \leq 5j$	$WW + \leq 5j$	$WWW + \leq 3j$	$t\bar{t} + \leq 3j$
$W + b\bar{b} + \leq 3j$	$WW + b\bar{b} + \leq 3j$	$WWW + b\bar{b} + \leq 3j$	$t\bar{t} + \gamma + \leq 2j$
$W + c\bar{c} + \leq 3j$	$WW + c\bar{c} + \leq 3j$	$WWW + \gamma\gamma + \leq 3j$	$t\bar{t} + W + \leq 2j$
$Z + \leq 5j$	$ZZ + \leq 5j$	$Z\gamma\gamma + \leq 3j$	$t\bar{t} + Z + \leq 2j$
$Z + b\bar{b} + \leq 3j$	$ZZ + b\bar{b} + \leq 3j$	$WZZ + \leq 3j$	$t\bar{t} + H + \leq 2j$
$Z + c\bar{c} + \leq 3j$	$ZZ + c\bar{c} + \leq 3j$	$ZZZ + \leq 3j$	$t\bar{b} + \leq 2j$
$\gamma + \leq 5j$	$\gamma\gamma + \leq 5j$		$b\bar{b} + \leq 3j$
$\gamma + b\bar{b} + \leq 3j$	$\gamma\gamma + b\bar{b} + \leq 3j$		
$\gamma + c\bar{c} + \leq 3j$	$\gamma\gamma + c\bar{c} + \leq 3j$		
	$WZ + \leq 5j$		
	$WZ + b\bar{b} + \leq 3j$		
	$WZ + c\bar{c} + \leq 3j$		
	$W\gamma + \leq 3j$		
	$Z\gamma + \leq 3j$		

Figure 1.2: The notorious experimenters wish list from the Run 2 Monte Carlo workshop in Fermilab [23], table from [28]

The calculation of processes in perturbation theory increases in complexity with each additional external particle (leg) and/or each independant internal momenta integrated over (loop); this partially due to the presence of Ultra-Violet (UV) and Infra-Red (IR) divergences. The UV divergences occur when propagators within the integrals grow very large, these handled by first using a regularizer on the integral then taking the continuum limit of the regularizing parameter and storing the resulting divergence in some function. This then allows the original or bare parameters of the integrals to be dressed so as to absorb the function in to a redefinition of the parameters. The IR divergences present only exist in the intermediate states of the process calculations and in general do not exist in the Euclidean region, which is the region used in this thesis. The IR divergences in the Minkowski space are usually cancelled amongst different contributions from the real emission and virtual corrections, the rest are absorbed inside the parton distribution functions. The main focus of LHC physics is concerned primar-

ily with multileg calculations, to calculate standard model backgrounds and possible Higgs channels. However multiloop calculations are still desirable but are currently much harder to accomplish. Figure 1.3, gives an overview of the state of the current multileg and multiloop calculations.

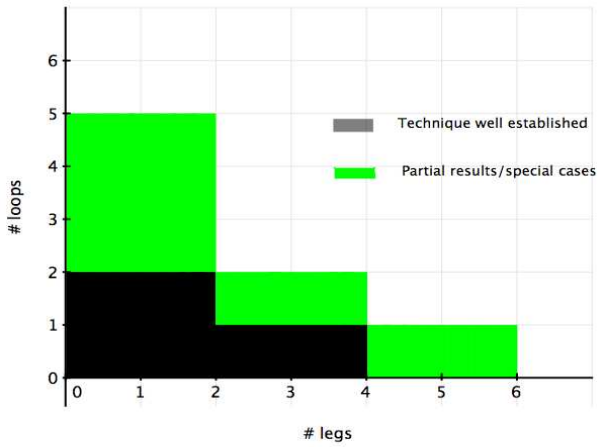


Figure 1.3: The current status of of multileg and multiloop calculations, progress in tree-level calculations are not shown. The black shaded area show processes for which the techniques are well established and/or have public codes. The Green area show areas which have partial results or special cases solved but where the general results are currently unknown. This graph is based on the graph by S. Heinemeyer [29] and has been updated appropriately.

For the multileg computations Next-to-Leading Order (NLO) $2 \to 2$ processes are now fairly well established. In addition a lot of progress has also been made on NLO $2 \to 3$ processes with the majority of those necessary to LHC physics completed, full evaluation of NLO $2 \to 3$ processes is under way (see [27, 30] for a recent reviews and references therein). The current frontier on the NLO multileg calculations are currently on the $2 \to 4$ processes with very few results available within the literature. The first of these to be calculated was the electroweak corrections to four fermion production in electron positron [31] which is more relevant for future possible colliders. Due to the recent completion of the building of the LHC there has been a focus on processes which are relevant to it, as such recent studies in the literature relevant for LHC phenomology at the NLO precision have been [32, 33, 34, 35, 36]. Many different techniques have been developed for the evaluating of multileg calculations but all share some basic similarities; the processes are split into their LO (tree level), NLO virtual corrections (which contain the loop integral) and the NLO real emission (a tree with soft and collinear particles radiating away from the final state). The UV divergence is dealt with by a regularization and renormolization, the IR divergences are treated with the use of subtraction terms such as the dipole subtraction [37, 38] or the antenna

subtraction [39, 40, 41]. From here the methods fall into two broad categories; the more traditional Feynman graph approach and a Unitarity based approach.

The standard Feynman graph approach involves evaluating the 1-loop amplitude by a sum of Feynman graphs sorted by their colour, the kinematics are represented by a 1-loop tensor integral which can be reduced to a set of scalar integrals in momenta-space [42] or in parameter space [43].

The Unitarity method is based upon the Cutkosky rules [44] and the unitarity of the S-Matrix, it involves sewing together tree-level graphs into loop graphs, a process also known as bootstrapping. This method was further developed by Bern, Dixon, Lance, Dunbar and Kosower [45, 46] and was linked to Twistor theory by Witten [47], a recent review of this method can be found in Annals of Physics [48].

When going beyond the NLO level, many new new problems arise. The UV divergent structures present in the NLO cases become a lot more complicated at Next-to-Next-to-Leading-Order (NNLO) due to the presence of an additional kind of divergence. The UV divergences found within a QFT fall into two main categories, the first kind is an overall divergence, which will occur if the naïve power counting degree is ≥ 0, that is to say that the graph diverges when all the loop parameters go large. The second type are subdivergences which occur when a sub-set of the overall divergence goes large. At the 1-loop level, there can only ever be an overall divergence as the 1-loop graphs do not contain a proper subgraph. Subdivergences are first introduced in 2-loop graphs, and these must be removed before the overall divergence of the graph can be cancelled.The subdivergences can present problems as the different subdivergences of a graph can overlap. These overlapping divergences occur when two or more loop momenta gets large. There can also be a mixing of UV and IR divergences in certain renormalization prescriptions (on-shell, $\overline{\text{MS}}$).

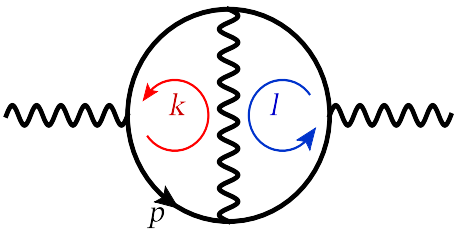


Figure 1.4: A 2-loop contribution to the photon self-energy with a common virtual photon propagator resulting in an overlapping divergence.

The second problem at the multiloop level is the number of terms which need to be evaluated, for example the QED photon self-energy at NLO (1-loop) has only one Feynman graph, at NNLO (2-loop) there are 2 graphs at the NNNLO (3-loop) there

are 20 graphs. In the non-abelian gauge theories this problem is much worse, the 3-loop gluon self-energy of QCD has 1319 graphs.

For these reasons there are relatively few calculations beyond NLO in the literature and the shear number of terms needing evaluation strongly suggests that a computer is the best way forward for multi-loop calculations. However the divergences present would need to be removed first, and it should be noted that anything with more than one mass-scale has essentially only a numerical solution. Some of the more important of these results have been the calculation of the 4-loop QCD β function [49], a result which has later been independently confirmed [50]. The anomalous magnetic moment of the muon has also receives much attention as the QED aspect of it is an important test for the theory, there is an analytic result at 3-loop level[51] but this is viewed as the limit of closed form calculations. Numerically the anomalous magnetic moment for QED has been calculated for 4-loops [52, 53] as well as some important 5-loop contributions [54, 55] by Kinoshita and his group. Recently, there has also been some development in some possible checks for these 4 and 5 loop contributions [56]. Other remarkable analytic results have been the 3-loop splitting functions of QCD [57, 58] and the renormalization constants constants of QCD to the same 3-loop level [59].

There are many methods for handling multiloop calculations, the two main methods are reduction to Master integrals and Sector decomposition. Master integrals are derived by first reducing the Feynman graphs to master integrals, an approach that first uses the parametric tensor reduction [43] which relates a tensor integral to some scalar integral of the form:

$$F(a_1,\ldots,a_n) = \int \cdots \int \frac{d^{2\omega}k_1 \ldots d^{2\omega}k_n}{E_1^{a_1}\ldots E_n^{a_n}} \tag{1.1}$$

where $k_i, i =, 1 \ldots, h$ are the loop momenta and the denominators are

$$E_r = \sum_{i\geq j\geq 1} A_r^{ij} p_i \cdot p_j - m_r^2.$$

with $r = 1,\ldots,n$ and the matrix A_r^{ij} is composed of the Feyman parameters. The momenta p_i can either be internal momenta or independent external momenta.

The resulting scalar integrals are then used with the Integration-By-Parts (IBP) identity [60]:

$$\int d^{2\omega}k_1 \ldots d^{2\omega}k_h \frac{\partial}{\partial k_i^\mu} J(k,\ldots) = 0 \tag{1.2}$$

This technique was used for the break-through calculation of the two-loop box graph with light-like external legs for planar topology [61] and the non-planar topology [62], which used the Master Integrals and applied the Mellin-Barnes expression:

$$\frac{1}{(a+b)^{\alpha}} = b^{-\alpha}\frac{1}{2\pi i}\int_{\mathcal{C}} dw \left(\frac{a}{b}\right)^{w} \frac{\Gamma(-w)\Gamma(\alpha+w)}{\Gamma(\alpha)} \tag{1.3}$$

This technique was has also been used in algorithm for numerical solutions [63].

The master integrals of (1.2) can be further constrained by the using the Lorentz Invariance Identity [64]:

$$\left(p_1^{\nu}\frac{\partial}{\partial p_{1\mu}} - p_1^{\mu}\frac{\partial}{\partial p_{1\nu}} + \ldots + p_n^{\nu}\frac{\partial}{\partial p_{n\mu}} - p_n^{\mu}\frac{\partial}{\partial p_{n\nu}}\right) I(p_1,\ldots,p_n) = 0. \tag{1.4}$$

and then solved by creating an over-determined system using differential equations and lorentz identities [64], [65]. There has also been some developments in using grȍbner basis methods to solve the master integrals [66].

The method of Sector Decomposition, a method particularly relevant for this thesis, was based on the proof by Hepp [67] of what would later be known as the BPHZ theorem of Bogliubov and Parasiuk [68]. This method divided the parameter space of a Feynman integral into different sectors to avoid overlapping UV singularities. This was later used by Denner and Roth [69] and extensively by Binoth and Heinrich [70, 71, 72, 73].

This method will be explained in greater detail in the next chapter.

Figure 1.5 shows the dividing of the parameter space and the remapping to new coordinates.

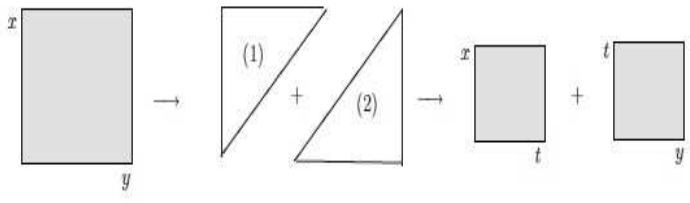


Figure 1.5: A schematic representation of the Sector decomposition process

All the methods introduced above can be found in the books by V. Smirnov [74, 75].

This Thesis presents a formal connection between the method of Sector Decomposition and the theoretically well understood BPHZ theorem, by comparing the sector decomposition of Feynman integrals in the parametric representation with a momentum decomposition of the same integrals in the momentum representation. Based on this connection, a code for handling the UV divergences and the calculation of the finite part of any given Feynman Graph in a general QFT is given based upon the recursive manipulations of the Feynman Graphs.

This is especially useful in the calculation of multi-loop graphs, as both these methods are algebraic; making them extremely well suited for use in automated codes. The basis for one such code is presented in this Thesis.

Chapter 2

Regularization and Renormalization

This thesis is concerned with automation of Feynman graph calculations through the use of recursive graph manipulations. Therefore it is imperative that both regularization and renormalization are explored. As such this chapter is concerned with looking at how in Quantum Field Theory at the 1-loop and higher orders the formalism breaks down due to divergent integrals. The ways in which these divergences can be categorized will be defined and in the case of Ultra-Violet divergences the techniques for saving the formalism of QFT will be explored with the introduction of the regularization of the integrals and renormalization of the action.

This chapter will focus specifically on the method of dimensional regularization and two recursive methods which are used to overcome the overlapping divergence problem, a general feature of multi-loop Feynman graphs. The first of these methods is the BPHZ theorem, which will be explored in two different representations and will include a proof of the the BPHZ theorem as well as a proof of the equivalence of the two different formulations of this theorem. The second of the recursive methods is that of sector decomposition, a method which is tied intrinsically to the Parametric representation of Feynman integrals. Finally, with an understanding of both the BPHZ theorem and of Sector Decomposition, the connection between the two recursive methods will be presented.

All work henceforth, unless explicitly stated, will be in Euclidean space as opposed to Minkowski space. This is done for a number of reasons most important of which is that in Euclidean space the Green's functions are a well defined mathematical object whereas in Minkowski space they can only be defined as a distribution integrated over a suitably smooth test function. The transition from Euclidean space to Minkowski space is done by analytic continuation. The background theory is based upon [76, 77, 78].

2.1 Divergences in Quantum Field Theory

The easiest way in which to see the disease inherent to Quantum Field Theory is to look at a 1-loop example from the toy model ϕ^3 in 2ω dimensions, as shown in figure 2.1. Given by the expression:

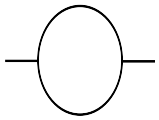


Figure 2.1: The 2-point 1-loop graph of ϕ^3-theory.

$$I_{\mathcal{G}} = \int d^{2\omega}k \left(\left[(k+p)^2 + m^2 \right] \left[k^2 + m^2 \right] \right)^{-1} \tag{2.1}$$

It can be sees then that this integral is divergent if the dimension $2\omega \geq 4$, this defines the Ultra-Violet (UV) divergence. UV divergences always occur when the integrand falls off as or more slowly than the inverse of the power of the denominator as the loop-momenta goes to ∞. There is also a second kind of divergence known as the Infra-Red (IR) divergence, which occurs when the integrand grows as or more rapidly than the the inverse of the power of the propagator as the loop momenta $k \to 0$, this would occur if we set the mass in the above example as $m = 0$. In non-Euclidean space the IR divergence will also reveal itself when the external legs of the diagram are on-shell. In Euclidean space this can only happen when the particles are massless. The IR divergence will not be investigated in this thesis, as for the main part only massive particles will be used for which there are no IR divergences.

2.1.1 Overlapping Divergences

Figure 2.2: The 2-point 2-loop graphs of ϕ^3-theory.

At the 2-loop level it can seen that an additional complication appears, that of the overlapping divergence.

Overlapping divergences occur when one or more loop-momenta share a common edge which cannot be resolved by a simple change of variables, as is the case in left graph of figure 2.2. If all the loop moment gets large simultaneously then the graph diverges as $4\omega - 10$, the problems arise in the case when the loop momenta gets large at

different rates as it is not obvious how to disentangle the subdivergences. This problem of the overlapping divergence is the motivation for the work in this thesis.

2.1.2 Power Counting

The concept of the overall degree of divergence is now introduced based on simple power counting method which was first studied by Dyson [12]. Dyson's work showed that the nature of the UV divergences are related to the power counting degree of divergence.

Using this as the starting point, the overall degree of divergence is defined for ϕ^3 theory as follows:

$$\deg(\mathcal{G}) \equiv 2\omega L - 2I \tag{2.2}$$

where this is to be read as that for any graph $\mathcal{G}$ which has L loops there will be a factor of $\int d^{2\omega}k$ associated with each loop and the I internal lines are of the form $\left[k^2 + m^2\right]^{-1}$.

Using the topology of a Feynman graph, a relation between edges and vertices can also defined. Taking a connected graph $\mathcal{G}$ with E_i external edges of type i, and I_i internal edges of type i connected to V_j vertices of type j; it is seen that the external edges can only be attached to a single vertex whilst internal edges I have to be attached at both ends to vertices. Using this observation, the following properties of $\mathcal{G}$ are established:

$$\sum_b n_{ij} V_j = E_i + 2I_i \geq 2I_i \quad \forall a \tag{2.3}$$

where n_{ij} is the number if edges of type i attached to an appropriate vertex j (for ϕ^3 $n_{ij} = 3$). If a subset of the edges I and vertices V forming a tree (a graph which contains no loops) is looked at, then in any such a graph there will always be $V - 1$ internal edges, and the remaining $I - V + 1$ edges will form loops in $\mathcal{G}$ as they are re-introduced. Using the above argument allows the general formula for the number of independent loops L in $\mathcal{G}$ to be defined:

$$L = \sum_i I_i - \sum_j V_j + 1 \leq \left(\frac{2\omega}{2} - 1\right) \sum_j V_j + 1 \tag{2.4}$$

and thus the number vertices has to obey the following inequality:

$$V \geq \frac{L-1}{\frac{2\omega}{2} - 1} \tag{2.5}$$

where eq.2.3) is used to establish the inequalities.

It is observed that each edge has to end in a vertex and that every vertex has to be attached to three edges, thus the following relation $E + 2I = 3V$ is obtained where V is

the number of vertices and E is the number of external edges to the graph. Using the two topological arguments together the definition for the overall degree of divergence can be re-expressed as a function of Vertices V and external edges E

$$\deg(\mathcal{G}) = (\omega - 3)V - (\omega - 1)E + 2\omega. \tag{2.6}$$

Analysing this expression it can be seen that if the coefficient of V is negative, then all graphs with a sufficient number of vertices will be overall, i.e. naïve power-counting, convergent and there can only ever be a finite number of overall divergent graphs. This is the definition of a Superrenormalizable theory. It should be noted that graphs with overall convergence can still diverge, but these divergence will be strictly from the sub-graphs.

If the case where the coefficient of V is positive is now considered, there will be overall divergent graphs for any Green's function regardless of the number of external legs and as such it would require an infinite number of counterterms to renormalize. These are defined as non-renormalizable theories.

In the final case, that of the coefficient of V is zero, then the overall degree of the graph can only depend on the number of external legs. Graphs with sufficiently many external legs are overall convergent and all others are overall divergent regardless of the number of vertices they have. For the ϕ^3-theory this can be expressed as:

$$2\omega - 6 \begin{cases} < 0 & superrenormalizable, \\ = 0 & renormalizable, \\ > 0 & non - renormalizable. \end{cases} \tag{2.7}$$

This argument can be generalized to arbitrary theories. The interactions of a theory with the set of fields $\{\phi_i\}$ is given by

$$S_I = \int d^{2\omega}x \sum_j g_j \partial^{d_j} \prod_i \phi_i^{n_{ij}} \tag{2.8}$$

the jth vertex has associated with it a coupling constant g_j, d_j derivatives and n_{ij} fields of type i. If each of the fields has a propagator Δ_i that satisfies the condition $\Delta_i(k) \leq \text{constant} \times \max(k, m)^{D_i}$ then for a $1PI$ graph $\mathcal{G}$ containing L loops, I_i internal lines and E_i external lines

$$L = \sum_i I_i - \sum_j V_j + 1, \tag{2.9}$$

$$\deg(\mathcal{G}) = 2\omega L + \sum_i D_i I_i + \sum_j d_j V_j, \tag{2.10}$$

$$2I_i + E_i = \sum_j n_{ij} V_j \qquad \forall i \tag{2.11}$$

which have the solution

$$\deg(\mathcal{G}) = \sum_j \dim(V_j) V_j - \sum_i \dim(\phi_i) E_i + 2\omega \tag{2.12}$$

with the dimension of the fields and verticies given by:

$$\dim(\phi_i) \equiv \frac{2\omega + D_i}{2}, \qquad \dim(V_i) \equiv \sum_i \dim(\phi_i) n_{ij} + d_j - 2\omega. \tag{2.13}$$

D_i distinguishes between fermionic and bosonic propagators $D_i = -1$ and -2 respectively, making the dimensions of the fermionic field $\dim(\psi) = \frac{3}{2}$ and the bosonic field $\dim(\phi) = 1$ in four dimensions which could've have been worked out by looking at the kinetic terms for the bosonic fields $\frac{1}{2}\phi(x)\partial_x^2\phi(x)$ and for the fermionic fields $\psi(x)\partial \cdot \gamma\psi(x)$.

The classification of the renormalizability of an arbitrary QFT is given by taking the dimension of the monomials in the interaction lagrangian.

$$\max \dim(V_j) = \begin{cases} < 0 & superrenormalizable, \\ = 0 & renormalizable, \\ > 0 & non-renormalizable. \end{cases} \tag{2.14}$$

2.2 Gamma Functions and Feynman Parameters

Euler Γ functions have a very important place in the analysis of QFTs. They can be used to provide an elegant introduction to Feynman parameters, a mathematical trick introduced by Richard Feynman to combine the denominators of a product of propagators into a single quadratic polynomial, and to the UV divergences in dimensional regularization. With this in mind and due to the crucial importance of understanding Feynman parameters and parametric forms for the link between henge decomposition methods and sector decomposition, some of the properties of the Γ function are explored as well as its relation to the Euler B-function $B(x, y)$.

The Γ function is defined by Euler's integral,

$$\Gamma(\alpha) = \int_0^\infty dt\, t^{\alpha-1} e^{-t}, \tag{2.15}$$

for $\mathrm{Re}\,\alpha > 0$. Using integration by parts yields the following relation,

$$\begin{aligned}\Gamma(\alpha+1) &= \int_0^\infty dt\, t^\alpha e^{-t} \\ &= \left[-t^\alpha e^{-t}\right]_0^\infty + \alpha\int_0^\infty dt t^{\alpha-1}e^{-t} \\ &= \alpha\Gamma(\alpha). \end{aligned} \tag{2.16}$$

Using equation (2.16) with

$$\Gamma(1) = \int_0^\infty dt\, t^{1-1}e^{-t} = -e^{-\infty} - (-e^0) = 0 - (-1) = 1$$

gives the result $\Gamma(n) = (n-1)!$ for $n \in \mathbb{N}$.

The Γ function thus satisfies the relation $\Gamma(\alpha) = \Gamma(\alpha+1)/\alpha$ which allows the function to be analytically continued to everywhere in the complex plane except for $-z-1 \in \mathbb{N}$, at which points $z = -n$ the Γ function has a simple pole with residue $(-1)^n/n!$.

The substitution $t = s^2$ leads to

$$\Gamma(\alpha) = 2\int_0^\infty ds\, s^{2\alpha-1}e^{-s^2}, \tag{2.17}$$

from which is obtained

$$\begin{aligned}\Gamma(\alpha)\Gamma(\beta) &= 4\int_0^\infty ds \int_0^\infty dt\, s^{2\alpha-1}t^{2\beta-1}e^{-(s^2+t^2)} \\ &= 4\int_0^\infty dr\, r^{2(\alpha+\beta)-1}e^{-r^2}\int_0^{\pi/2} d\theta\,(\cos\theta)^{2\alpha-1}(\sin\theta)^{2\beta-1} \end{aligned} \tag{2.18}$$

by introducing polar coordinates $s = r\cos\theta$ and $t = r\sin\theta$.

This leads to the relation $\Gamma(\alpha)\Gamma(\beta) = \Gamma(\alpha+\beta)B(\alpha,\beta)$, where

$$B(\alpha,\beta) = 2\int_0^{\pi/2} d\theta\,(\cos\theta)^{2\alpha-1}(\sin\theta)^{2\beta-1} = \int_0^1 dx\, x^{\alpha-1}(1-x)^{\beta-1} = \int_0^\infty dt \frac{t^{\beta-1}}{(1+t)^{\alpha+\beta}}, \tag{2.19}$$

upon substituting $x = (\cos\theta)^2$ or $t = (\tan\theta)^2$ respectively.

An important result of equation (2.17) is when $\alpha = \frac{1}{2}$:

$$\Gamma\left(\frac{1}{2}\right)^2 = 4\int_0^\infty dx \int_0^\infty dy e^{-(x^2+y^2)} = 4\int_0^{\pi/2} d\theta \int_0^\infty dr r e^{-r^2} = \pi, \tag{2.20}$$

which gives $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$.

2.2.1 Laplace transform of $t^{\alpha-1}$

The Γ functions is closely related to the Laplace transform $\mathcal{L}(t^{\alpha-1}) \equiv \int_0^\infty dt\, t^{\alpha-1}e^{-At}$ with $\mathrm{Re}\,\alpha > 0$; this may be evaluated by the substitution $s = At$, which leads to $\mathcal{L}(t^{\alpha-1}) = A^{-\alpha}\int_0^\infty ds\, s^{\alpha-1}e^{-s} = A^{-\alpha}\Gamma(\alpha)$.

Substituting $t = s^2$ as before leads to the identity

$$\frac{\Gamma(\alpha)}{A^\alpha} = 2\int_0^\infty ds\, s^{2\alpha-1}e^{-As^2}, \tag{2.21}$$

which when viewed in conjunction with the properties of the Γ-function, suggest that:

$$\begin{aligned}\frac{\Gamma(\alpha)\Gamma(\beta)}{A^\alpha B^\beta} &= 4\int_0^\infty ds\int_0^\infty dt\, s^{2\alpha-1}t^{2\beta-1}e^{-(As^2+Bt^2)} \\ &= 4\int_0^\infty dr\, r^{2(\alpha+\beta)-1}\int_0^{\pi/2} d\theta\,(\cos\theta)^{2\alpha-1}(\sin\theta)^{2\beta-1}e^{-r^2\left[A(\cos\theta)^2+B(\sin\theta)^2\right]} \\ &= 4\int_0^\infty d\rho\,\rho^{2(\alpha+\beta)-1}e^{-\rho^2}\int_0^{\pi/2} d\theta\,\frac{(\cos\theta)^{2\alpha-1}(\sin\theta)^{2\beta-1}}{\left[A(\cos\theta)^2+B(\sin\theta)^2\right]^{\alpha+\beta}},\end{aligned} \tag{2.22}$$

with $\rho^2 = r^2\left[A(\cos\theta)^2 + B(\sin\theta)^2\right]$. Using the substitutions $x = (\cos\theta)^2$ or $t = (\tan\theta)^2$ as before, this becomes

$$\frac{\Gamma(\alpha)\Gamma(\beta)}{A^\alpha B^\beta} = \Gamma(\alpha+\beta)\int_0^1 dx\,\frac{x^{\alpha-1}(1-x)^{\beta-1}}{[Ax+B(1-x)]^{\alpha+\beta}} = \Gamma(\alpha+\beta)\int_0^\infty dt\,\frac{t^{\beta-1}}{(A+Bt)^{\alpha+\beta}}. \tag{2.23}$$

The above equation shows a way of combining two denominators with quadratic forms into a common denominator via the use of the variable x. This relation is the simplest of the Feynman parameter relation, with x being the Feynman parameter. This formula can then be extended to an arbitrary number of denominators with quadratic forms via induction, as will be show in the next section.

2.2.2 Multiple Feynman parameter identity

The fundamental Feynman parameter identity may be written in the inhomogenous form

$$\prod_{\ell=1}^N \frac{\Gamma(\alpha_\ell)}{Q_\ell^{\alpha_\ell}} = \Gamma\left(\sum_{\ell=1}^N \alpha_\ell\right)\left(\prod_{\ell=2}^N \int_0^\infty dt_\ell\, t_\ell^{\alpha_\ell-1}\right)\left[Q_1 + \sum_{j=2}^N t_j Q_j\right]^{-\sum_{k=1}^N \alpha_k}. \tag{2.24}$$

This is established by induction on N. For $N = 1$ the identity is trivial: assuming it holds for N, it is multiplied by $\Gamma(\alpha_{N+1})/Q_{N+1}^{\alpha_{N+1}}$, and within the integrand we apply

the last form of equation (2.23)

$$\begin{aligned}
&\Gamma\left(\sum_{\ell=1}^{N}\alpha_\ell\right)\left[Q_1+\sum_{j=2}^{N}t_jQ_j\right]^{\sum_{k=1}^{N}\alpha_k}\times\frac{\Gamma(\alpha_{N+1})}{Q_{N+1}^{\alpha_{N+1}}}\\
=\;&\Gamma\left(\left(\sum_{\ell=1}^{N}\alpha_\ell\right)+\alpha_{N+1}\right)\times\int_0^\infty dt_{N+1}\,\frac{t_{N+1}^{\alpha_{N+1}-1}}{\left[\left(Q_1+\sum_{j=2}^{N}t_jQ_j\right)+t_{N+1}Q_{N+1}\right]^{\left(\sum_{k=1}^{N}\alpha_k\right)+\alpha_{N+1}}}
\end{aligned}\tag{2.25}$$

to obtain

$$\prod_{\ell=1}^{N+1}\frac{\Gamma(\alpha_\ell)}{Q_\ell^{\alpha_\ell}}=\Gamma\left(\sum_{\ell=1}^{N+1}\alpha_\ell\right)\left(\prod_{\ell=2}^{N+1}\int_0^\infty dt_\ell\,t_\ell^{\alpha_\ell-1}\right)\left[Q_1+\sum_{j=2}^{N+1}t_jQ_j\right]^{-\sum_{k=1}^{N+1}\alpha_k}\tag{2.26}$$

as required.

Equation (2.24) may be written in a slightly more symmetric homogeneous form by introducting the variable t_1 by inserting the identity $\int_0^\infty dt_1\,t_1^{\alpha_1-1}\delta(1-t_1)=1$ into equation (2.24):

$$\prod_{\ell=1}^{N}\frac{\Gamma(\alpha_\ell)}{Q_\ell^{\alpha_\ell}}=\Gamma\left(\sum_{\ell=1}^{N}\alpha_\ell\right)\left(\prod_{\ell=1}^{N}\int_0^\infty dt_\ell\,t_\ell^{\alpha_\ell-1}\right)\frac{\delta(1-t_1)}{\left[\sum_{j=1}^{N}t_jQ_j\right]^{\sum_{k=1}^{N}\alpha_k}}.\tag{2.27}$$

The identity may be cast into an even more symmetric form by multiplying by the identity $\int_0^\infty dT\,\delta\left(T-\sum_{k=1}^{N}t_k\right)$ and changing variable to $x_\ell\equiv t_\ell/T$ for $\ell=1,\ldots,N$:

$$\begin{aligned}
\prod_{\ell=1}^{N}\frac{\Gamma(\alpha_\ell)}{Q_\ell^{\alpha_\ell}}&=\Gamma\left(\sum_{\ell=1}^{N}\alpha_\ell\right)\int_0^\infty dT\left(\prod_{\ell=1}^{N}\int_0^\infty dx_\ell\,x_\ell^{\alpha_\ell-1}\right)\frac{\delta\left(T-T\sum_{k=1}^{N}x_k\right)\delta(1-Tx_1)}{\left[\sum_{j=1}^{N}x_jQ_j\right]^{\sum_{k=1}^{N}\alpha_k}}\\
&=\Gamma\left(\sum_{\ell=1}^{N}\alpha_\ell\right)\left(\prod_{\ell=1}^{N}\int_0^1 dx_\ell\,x_\ell^{\alpha_\ell-1}\right)\delta\left(1-\sum_{k=1}^{N}x_k\right)\left[\sum_{j=1}^{N}x_jQ_j\right]^{-\sum_{k=1}^{N}\alpha_k}
\end{aligned}\tag{2.28}$$

For the special case where $\alpha_j=1\;\forall j$ leads to:

$$\prod_{\ell=1}^{N}\frac{1}{Q_\ell}=\Gamma(N)\int_0^1 dx_1\cdots\int_0^1 dx_N\,\delta\left(1-\sum_{k=1}^{N}x_k\right)\left[\sum_{j=1}^{N}x_jQ_j\right]^{-N}.\tag{2.29}$$

2.3 Regularization

As explained in section 2.1, divergences are a serious problem in QFT and in order to extract any useful information, a way must be found to analyze them. A way in which this is tackled in QFT is with the concept of regularization. This can take many forms and essentially any method which can systematically identify and isolate the divergences is valid. Some of the most widely used methods for regularizing the integrals are by adding additional propagator with a large mass with a sign negative to that of the normal propagator, as used in the Pauli-Villars method [79], or by the use of some cut-off on the integration region such that the remaining integral is finite, as used in methods such as lattice QFT [80]. This cut-off method essentially expresses a divergent Euclidean integral as the limit of a convergent Euclidean integral. Another very important method is that of dimensional regularization, which will be explained in greater detail later.

No matter what method is chosen, the idea of regularization is the same. That is to apply some condition (the regulator) on the integral such that it becomes finite and then relax the same condition so that the original integral is recovered at the expense of having the solution dependent on the regularizing condition and with the breaking some symmetries of the original theory. The divergence within the original result will now be a singularity in the regularizing parameter.

In order to demonstrate this, the following example is examined:

$$I = \int_{-\infty}^{\infty} dx \frac{e^x}{1+e^x} \tag{2.30}$$

This integral is then regularized by imposing a cut-off on the integration region, such that a solution is obtained:

$$\mathrm{Reg}_\Lambda(I) = \int_{-\Lambda}^{\Lambda} dx \frac{e^x}{1+e^x} = ln\left(\frac{1+e^\Lambda}{1+e^{-\Lambda}}\right) \tag{2.31}$$

As can be seen, the solution is now dependent on the cut-off parameter Λ. It is however the solution to a different problem than the one which needed to be solved. In order to recover the solution to the original integral is by taking the limit $\lim_{\Lambda\to\infty}$:

$$\mathrm{Reg}(I) = \lim_{\Lambda\to\infty} \mathrm{Reg}_\Lambda(I) \tag{2.32}$$

It is important to note that each regularization procedure has its advantages and disadvantages and are suitable for different types of calculation. For instance, in lattice regularization gauge symmetry is preserved and it is currently the only method suitable for computational non-perturbative calculations, it does however break Poincare invariancé.

2.4 Dimensional Regularization

As mentioned at the beginning of this chapter, the integral for the 1-loop 2-point function in ϕ^3 theory is divergent if the dimension $2\omega \geq 4$. This observation, that a divergent integral maybe convergent in a dimension other than the one in question leads to the idea of Dimensional Regularization. Dimensional Regularization was first introduced in the context of QFT by Gerardus 't Hooft and Martinus Veltman [81, 82], and in this prescription it defines the dimension 2ω to be a complex number which can be a non-integer number, such that when the integral has been performed, an analytic continuation is then done to restore the result to the correct dimension. This results in the divergence present in the integral to be a pole, though not necessarily a simple one.

With all this in mind, consider the gaussian integral

$$I \equiv \int_{-\infty}^{\infty} dx_1 \cdots \int_{-\infty}^{\infty} dx_{2\omega} e^{-\sum_{i=1}^{2\omega} x_i^2} \tag{2.33}$$

which can be factorised in 2ω gaussian integrals

$$I = \left[\int_{-\infty}^{\infty} dx e^{-x^2}\right]^{2\omega} = \Gamma(\frac{1}{2})^{2\omega} \tag{2.34}$$

This integral can also be evaluated with in spherical polar coordinates and with the help definition of the Γ function eq.(2.17)

$$I = S_{2\omega} \int_0^{\infty} dr r^{2\omega-1} e^{-r^2} = \frac{1}{2} S_{2\omega} \Gamma(\omega) \tag{2.35}$$

which gives the surface area of the unit sphere in 2ω dimensions

$$S_{2\omega} = \frac{2\Gamma\left(\frac{1}{2}\right)^{2\omega}}{\Gamma(\omega)} \tag{2.36}$$

With this and with the use of the definition of B function eq.(2.19) allows the evaluation of the following integral:

$$\begin{aligned} \int d^{2\omega}k \frac{(k^2)^{\rho}}{(k^2+m^2)^{\lambda}} &= S_{2\omega}(m^2)^{\omega+\rho-\lambda} \int_0^{\infty} dk \frac{l^{2(\omega+\rho)-1}}{(k^2+1)^{\lambda}} \\ &= \frac{1}{2} S_{2\omega}(m^2)^{\omega+\rho-\lambda} B(\lambda-\rho-\omega, \rho+\omega) \end{aligned} \tag{2.37}$$

This integral has a simple pole at dimensions where the power-counting degree $d \equiv 2(\omega+\rho-\lambda)$ is a non-negative multiple of 2, $\frac{1}{2}d \in \mathrm{N}$.

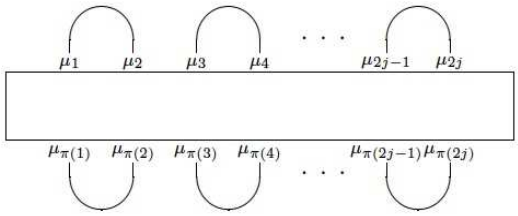
The above integral can be generalized to arbitrary number of pairs of numerator

momenta $\int d^{2\omega}k \frac{k_{\mu_1}k_{\mu_2}\cdots k_{\mu_{2j-1}}k_{\mu_{2j}}}{(k^2+m^2)^\lambda}$.

which is given by:

$$\int d^{2\omega}k \frac{k_{\mu_1}k_{\mu_2}\cdots k_{\mu_{2j-1}}k_{\mu_{2j}}}{(k^2+m^2)^\lambda} = \int \frac{d^{2\omega}k}{(k^2+m^2)^{\lambda-j}} f_j(\omega,\lambda)\frac{1}{(2j)!}\sum_{\pi\in S_{2j}} \delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \tag{2.38}$$

which is some unknown function $f_j(\omega,\lambda)$ times the all possible permutations of the δ functions of the symmetric group $\mathcal{S}_{2j}$ which will contract the numerator of the left-hand side and scalar integral whose power is derived from dimensional analysis. The factor $f_j(\omega,\lambda)$ can be evaluated by multiplying both sides of the equation by $\delta_{\mu_1\mu_2}\cdots\delta_{\mu_{2j-1}\mu_{2j}}$ and summing over the indices $\mu_1\ldots\mu_2 j$. Fig. 2.4 shows the contracted tensors from the right-hand side of the equation.



$$= 2\omega\prod_{i=1}^{j-1}\frac{2\omega+2i}{1+2i} = \frac{\Gamma(\omega+j)\Gamma(\frac{1}{2})}{\Gamma(\omega)\Gamma(j+\frac{1}{2})};$$

Figure 2.3: The symmetrizer of the contracted δ functions of the group $\mathcal{S}_{2j}$ and its result

The result of fig. 2.4 is obtained by considering if the symmetrizer (white box) was an anti-symmetrizer. From [83], it is shown that if the dening dimension is smaller than the number of indices then the anti-symmetrizer returns zero. As such, if fig. 2.4 was an anti-symmetrizer then it would return zero for values of $2\omega = 0, 2, \ldots, 2(j-1)$, also from [83] it is shown that there is relation between the symmetric algebra $so(\omega)$ and the anti-symmetric algebra $sp(\omega)$ and is given by $so(\omega) \leftrightarrow sp(-\omega)$, as such the symmetrizerizer must vanish for values of $2\omega = 0, -2, \ldots, -2(j-1) = \prod_{i=1}^{j-1} 2\omega + 2i$ To normalize this function, it is insisted upon that when $2\omega = 1$ each permutation will give a contribution of $\frac{1}{j!}$, so the factor will be 1. which transforms the symmetrizer product to be $\prod_{i=1}^{j-1}\frac{2\omega+2i}{1+2i}$. Finally setting $i = 0$ in the product yields that factor of 2ω and as such the symmetrizer of fig. 2.4 will be $2\omega\prod_{i=1}^{j-1}\frac{2\omega+2i}{1+2i}$ which can be written in terms of Γ functions.

Using the result from fig. 2.4 to obtain

$$\int d^{2\omega}k \frac{(k^2)^j}{(k^2+m^2)^\lambda} = \int \frac{d^{2\omega}k}{(k^2+m^2)^{\lambda-j}} f_j(\omega,\lambda)\frac{\Gamma(\omega)\Gamma(j+\frac{1}{2})}{\Gamma(\omega+j)\Gamma(\frac{1}{2})}; \tag{2.39}$$

which upon re-arranging gives

$$f_j(\omega,\lambda) = \frac{B(\lambda-j-\omega,j+\omega)}{B(\lambda-j-\omega,\omega)}\frac{\Gamma(\omega)\Gamma(j+\frac{1}{2})}{\Gamma(\omega+j)\Gamma(\frac{1}{2})} = \frac{\Gamma(j+\frac{1}{2})\Gamma(\lambda-j)}{\Gamma(\frac{1}{2})\Gamma(\lambda)} \tag{2.40}$$

which when used, gives the result for a general 1-loop integral in 2ω dimensions:

$$\begin{aligned}\int d^{2\omega}k\frac{k_{\mu_1}\cdots k_{\mu_{2j}}}{(k^2+m^2)^\lambda} &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}(m^2)^{\omega-\lambda+j} \\ &\quad\times\frac{1}{(2j)!}\sum_{\pi\in S_{2j}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}}\end{aligned} \tag{2.41}$$

2.4.1 Examples of Dimensional Regulated Scalar Integrals

$$\int d^{2\omega}k\frac{1}{(k^2+m^2)^\lambda} = \pi^\omega\frac{\Gamma(\lambda-\omega)}{\Gamma(\lambda)}(m^2)^{\omega-\lambda} \tag{2.42}$$

$$\int d^{2\omega}k\frac{k_\mu k_\nu}{(k^2+m^2)^\lambda} = \frac{1}{2}\pi^\omega\frac{\Gamma(\lambda-\omega-1)}{\Gamma(\lambda)}(m^2)^{\omega-\lambda+1}\delta_{\mu\nu} \tag{2.43}$$

$$\int d^{2\omega}k\frac{k_\mu k_\nu k_\rho k_\sigma}{(k^2+m^2)^\lambda} = \frac{3}{4}\pi^\omega\frac{\Gamma(\lambda-\omega-2)}{\Gamma(\lambda)}(m^2)^{\omega-\lambda+2}\left(\frac{\delta_{\mu\nu}\delta_{\rho\sigma}+\delta_{\mu\rho}\delta_{\nu\sigma}+\delta_{\mu\sigma}\delta_{\nu\rho}}{3}\right) \tag{2.44}$$

$$\begin{aligned}\int d^{2\omega}k\frac{k_\mu k_\nu k_\rho k_\sigma k_\alpha k_\beta}{(k^2+m^2)^\lambda} &= \frac{15}{8}\pi^\omega\frac{\Gamma(\lambda-\omega-3)}{\Gamma(\lambda)}(m^2)^{\omega-\lambda+3} \\ &\quad\times\frac{1}{15}\left(\begin{array}{rl} & \delta_{\sigma\mu}\delta_{\alpha\beta}\delta_{\nu\rho}+\delta_{\sigma\alpha}\delta_{\beta\mu}\delta_{\nu\rho}+\delta_{\sigma\alpha}\delta_{\beta\nu}\delta_{\mu\rho} \\ + & \delta_{\sigma\alpha}\delta_{\beta\rho}\delta_{\mu\nu}+\delta_{\sigma\beta}\delta_{\alpha\mu}\delta_{\nu\rho}+\delta_{\sigma\beta}\delta_{\alpha\nu}\delta_{\mu\rho} \\ + & \delta_{\sigma\beta}\delta_{\alpha\rho}\delta_{\mu\nu}+\delta_{\sigma\mu}\delta_{\alpha\nu}\delta_{\beta\rho}+\delta_{\sigma\mu}\delta_{\alpha\rho}\delta_{\beta\nu} \\ + & \delta_{\sigma\nu}\delta_{\alpha\beta}\delta_{\mu\rho}+\delta_{\sigma\nu}\delta_{\alpha\mu}\delta_{\beta\rho}+\delta_{\sigma\nu}\delta_{\alpha\rho}\delta_{\beta\mu} \\ + & \delta_{\sigma\rho}\delta_{\alpha\beta}\delta_{\mu\nu}+\delta_{\sigma\rho}\delta_{\alpha\mu}\delta_{\beta\nu}+\delta_{\sigma\rho}\delta_{\alpha\nu}\delta_{\beta\mu}\end{array}\right)\end{aligned}$$

2.5 Multiloop Integrals in Dimensional Regularization

An L loop Feynman graph whose propagators are inverse quadratic forms in the momenta can be evaluated by introducing Feynman parameters. Introducing a $2\omega L$ dimensional momentum space which is the tensor sum of the individual loop momentum spaces; the components of the vector in this space may be written as k_{ℓ_μ} where $\ell = 1,\ldots,L$ specifies the loop and $\mu = 1,\ldots,2\omega$ the component within that loop. A $2\omega E$ dimensional vector P of all E external momenta is also introduced. The most general homogeneous momentum integral that arises is then of the form

$$\int d^{2\omega L}k \frac{k_{\mu_1}\cdots k_{\mu_{2j}}}{(k^2+m^2)^\lambda} = \int d^{2\omega L}k \frac{k_{\mu_1}\cdots k_{\mu_{2j}}}{Q(k)^\lambda} \tag{2.45}$$

where the quadratic form $Q(k) = k^T\bar{M}k - 2k^T\bar{B}P + P^T\bar{C}P + D$ with the matrices $\bar{M}$, $\bar{B}$, and $\bar{C}$ depending only on the Feynman parameters and D depending on the Feynman parameters and the masses. $SO(n)$ symmetry means that $\bar{M} = M \otimes \mathbb{I}$ where $\mathbb{I}$ is the unit matrix acting on the 2ω spacetime indices and M is an $L \times L$ matrix acting on the loop indices: in terms of components $\bar{M}_{\ell\mu,\ell'\mu'} = M_{\ell\ell'}\delta_{\mu\mu'}$. We thus have

$$k^t\bar{M}k = \sum \ell,\ell' = 1^L \sum_{\mu,\mu'=1}^{2\omega} k_{\ell\mu}M_{\ell,\ell'}k_{\ell'\mu} = \sum \ell,\ell' = 1^L k_\ell^T M_{\ell,\ell'}k'_\ell, \tag{2.46}$$

and

$$Q(k) = k^T(M \otimes \mathbb{I})k - 2k^T(B \otimes \mathbb{I})P + P^T(C \otimes \mathbb{I})P + D. \tag{2.47}$$

The quadratic form may be simplified by completing the square, which gives $Q(k_0 + k') = k'^T(M \otimes \mathbb{I})k' + \mu^2$ where $\mu^2 \equiv P^T\left((C - B^TM^{-1}B) \otimes \mathbb{I}\right)P + D$, which is a polynomial in the Feynman parameters. Under this transformation the numerator factors are of the form $k_{\ell\mu} = (k_0 + k')_{\ell\mu} = [(M^{-1}BP)_\ell]_\mu + k'_{\ell\mu}$, so the integral may be split into a sum of terms each of which is homogeneous in the shifted loop momentum k' with numerators of degree $1, \ldots, 2j$. We may change the integration variable from k to k' with a trivial Jacobian. As the matrix M is symmetric and positive it may be diagonalised by an orthogonal transformation $M = O^TD^2O$, so a new momentum variable $k'' = (DO \otimes \mathbb{I})k'$ can be introduced in terms of which the quadratic form becomes $Q = k''^2 + \mu^2$ and the numerator momenta are

$$k'_{\ell\mu} = \left((O^TD^{-1} \otimes \mathbb{I})k''\right))_{\ell\mu} = (O^TD^{-1})_{\ell,\ell'}k''_{\ell'\mu}. \tag{2.48}$$

The Jacobian associated with this change of variables is

$$\det \frac{\partial k'}{\partial k''} = \det\,(O^TD^{-1} \otimes \mathbb{I}) = \det\,(O^TD^{-1})^{2\omega} = \det\,(D^{-1})^{2\omega} = \det\,M^{-\omega} \tag{2.49}$$

The integral to be performed is then of the form

$$I = \frac{(O^TD^{-1})_{\ell_1,\ell'_1}\cdots(O^TD^{-1})_{\ell_{2j},\ell'_{2j}}}{\det\,M^\omega} \int d^{2\omega L}k'' \frac{k''_{\ell'_1\mu_1}\cdots k''_{\ell'_{2j}\mu_{2j}}}{(k''^2+\mu^2)^\lambda} \tag{2.50}$$

The integration over k'' is easily carried out using equation (2.41), and yields

$$\begin{aligned} I &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}\frac{(\mu^2)^{\omega-\lambda+j}}{\det M^{\omega}}(O^T D^{-1})_{\ell_1,\ell'_1}\cdots(O^T D^{-1})_{\ell_{2j},\ell'_{2j}}\times \\ &\quad\times\frac{1}{(2j)!}\sum_{pi\in S_{2j}}\delta_{\ell'_{\pi(1)}\ell'_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots\delta_{\ell'_{\pi(2j-1)}\ell'_{\pi(2j)}}\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \end{aligned} \tag{2.51}$$

where $d = 2(\omega L + j - \lambda)$ is the overall degree of divergence of the integral,

$$\begin{aligned} &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}\frac{(\mu^2)^{\omega-\lambda+j}}{\det M^{\omega}} \\ &\quad\times\frac{1}{(2j)!}\sum_{\pi\in S_{2j}}(O^T D^{-2})_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(O^T D^{-2})_{\ell_{\pi(2j-1)}\ell_{\pi(2j)}}\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \\ &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}\frac{(\mu^2)^{\omega-\lambda+j}}{\det M^{\omega}} \\ &\quad\times\frac{1}{(2j)!}\sum_{\pi\in S_{2j}}(M^{-1})_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(M^{-1})_{\ell_{\pi(2j-1)}\ell_{\pi(2j)}}\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \\ &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}\frac{(\mu^2)^{\omega-\lambda+j}}{\det M^{\omega+j}} \\ &\quad\times\frac{1}{(2j)!}\sum_{\pi\in S_{2j}}(\text{adj } M)_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(\text{adj } M)_{\ell_{\pi(2j-1)}\ell_{\pi(2j)}}\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \end{aligned} \tag{2.52}$$

where $(\text{adj } M)_{ij}$ is the adjoint of matrix M_{ij}.

2.5.1 An Example of the Parametric Representation of Feynman Integrals

In preparation for Sector Decomposition the parametric representation of the Feynman Graph is introduced with an example from ϕ_6^3 theory.

The algebraic form of the overlapping graph in figure 2.2 is given below:

$$\begin{aligned} I_{\mathcal{G}} &= \int d^{2\omega}k_1\, d^{2\omega}k_2\, \left[\left(k_1^2+m^2\right)\left(k_2^2+m^2\right)\right. \\ &\quad \left.\left((p+k_2)^2+m^2\right)\left((p+k_1)^2+m^2\right)\left((k_1-k_2)^2+m^2\right)\right]^{(-1)} \end{aligned} \tag{2.53}$$

Applying the Feynman parameter formula on the integral eq.(2.29) to get

$$I_{\mathcal{G}} = \Gamma(5)\int_0^1 dx_1\ldots dx_5\int d^{2\omega}k_1 d^{2\omega}k_2\delta\left(1-\sum_{k=1}^{5}x_k\right)\left[x_1\left(k_1^2+m^2\right)+x_2\left(k_2^2+m^2\right)\right.$$
$$\left.+x_3\left((p+k_2)^2+m^2\right)+x_4\left((p+k_1)^2+m^2\right)+x_5\left((k_1-k_2)^2+m^2\right)\right]^{-5} \quad (2.54)$$

It is easily seen that the denominator of the integrand, $Q(k)$, is made up of terms which involve just the internal momenta k_1 and k_2, those which involve a combination of internal k and external momenta p, those which involve only the external momenta p^2 and finally those which do not involve any momenta at all. Therefore the denominator is rewritten in the form shown below

$$Q(k) = \sum_{i,j=1}^{L} K_i \cdot K_j M_{ij} - 2\sum_{i=1}^{L} K_i \cdot b_i p + pcp + d \quad (2.55)$$

where for the example,

$$K = \begin{pmatrix} k1 \\ k2 \end{pmatrix};$$

$$M = \begin{pmatrix} x_1+x_4+x_5 & -x_5 \\ -x_5 & x_2+x_3+x_5 \end{pmatrix};$$

$$b = \begin{pmatrix} -x_4 \\ -x_3 \end{pmatrix};$$

$$c = (x_3+x_4)$$

$$d = m^2\left(x_1+x_2+x_3+x_4+x_5\right)$$

One of the reasons to do this is that due to the symmetric nature of the Feynman Integral, it can be reduced from the 2-loop graph to the form of a 1-loop integral in a higher dimensional space, i.e.

$$I_{\mathcal{G}} = \int d^{2\omega L}K\left[k^2+\mu^2\right]^{\alpha} \quad (2.56)$$

where L is the number of loops in the graph.

Using the value for the Feynman parameter matrices in eq.(2.52), to get

$$I_{\mathcal{G}} = \frac{\Gamma\left(\frac{1}{2}\right)^{2\omega}\Gamma(5-\omega)}{\Gamma(5)}\int_0^1 dx_1\ldots dx_5\delta\left(1-\sum_{i=1}^{5}x_i\right)\frac{\left(\mu^2\right)^{\omega-5}}{\det M^{\omega}} \quad (2.57)$$

with $\det M = x_1x_2 + x_1x_3 + x_1x_5 + x_4x_2 + x_4x_3 + x_4x_5 + x_2x_5 + x_3x_5$ and $\mu^2 = (x_3+x_4)p^2 - \frac{x_3^2(x_1+x_2)p^2}{(x_{12}+x_{13}+x_{32}+x_{41}+x_{42})} + m^2x_{12345}$ where the notation $x_{ijkl} = x_i+x_j+x_k+x_l$.

Now that eq.(2.53) is in the parametric representation, further manipulation can be done in order to get it into a more useful form for sector decomposition.

This is done by introducing two new functions $\mathcal{U}$ and $\mathcal{F}$.

In order to do this we first introduce a new matrix $\tilde{M}$ defined by

$$\tilde{M} = (\det M)^{-1} M$$

this changes our integrand (excluding δ-fn) in the following way

$$\begin{aligned} \Rightarrow \mu^2 &= (\det M)^{-1} \left[b\tilde{M}^{-1}b + \det M\,(c+d) \right] \\ &= \frac{\mathcal{F}}{\mathcal{U}} \end{aligned}$$

where $\mathcal{U}$ and $\mathcal{F}$ are defined to be

$$\mathcal{U} = det M \qquad \text{and} \qquad \mathcal{F} = \left[B^T\tilde{M}^{-1}B + \mathcal{U}\,(C+D) \right]$$

which changes the parametric integral into the form

$$I_{\mathcal{G}} = \frac{\Gamma\left(\frac{1}{2}\right)^{2\omega}\Gamma(5-\omega)}{\Gamma(5)} \int_0^1 dx_1 \ldots dx_5 \delta\left(1 - \sum_{i=1}^{5} x_i\right) \frac{\mathcal{U}^{5-3\omega}}{\mathcal{F}^{5-2\omega}} \tag{2.58}$$

The general form of this equation is

$$I_{\mathcal{G}} = \frac{\Gamma\left(\frac{1}{2}\right)^{2\omega}\Gamma(N-L\omega)}{\prod_{i=1}^{N}\Gamma(\alpha_i)} \int_0^1 \prod_{i=1}^{N} dx_i x_i^{\alpha_i-1} \delta\left(1 - \sum_{l=1}^{N} x_l\right) \frac{\mathcal{U}^{N-(L+1)\omega}}{\mathcal{F}^{N-L\omega}} \tag{2.59}$$

where N is the number of propagators, L is the number of loops, α is the power of the propagators and all other symbols take their usual meaning

It also should be noted that the functions $\mathcal{U}$ and $\mathcal{F}$ may be obtained in another way, the function $\mathcal{U}$ is the obtained from the topology of the graph and $\mathcal{F}$ comes from the cutting rules of the graph and Mandelstam variables [84, 85, 70, 71, 72, 73]. The function $\mathcal{U}$ is positive semi-definite and will always contain the UV subdivergence (the overall UV divergence is contained within the Γ function), i.e. when $\mathcal{U}$ vanishes there will be a UV subdivergence. In Euclidean space the function $\mathcal{F}$ is also positive semi-definite and will contain the IR-divergence only if its parameters (mass, external momenta) vanish, In Minkowski space the situation is more complex. If none of the external momenta or masses vanish then the function F will be safe in Euclidean space.

2.6 Renormalization

Renormalization identifies and eliminates the divergence from the limit of the convergent integral.

It has been shown that the divergences which occur in the loop-graphs of quantum field theories can be solved with the use of a regularizer which breaks some of the symmetries of the original theory.

When the regularizing paramemeter is taken in the limit to which it is removed the original theory is restored, at the expense of making the parameters of the Lagrangian dependent on the regularizing parameter and divergent, that is the parameters of a power series of the regularizing parameter diverge as the limit of the regularizing parameter is taken. For regularizing procedures such as dimensional regularization and analytic regularization, this takes the form that in the regulated theory the Green's functions are analytic functions of the regulator parameter with poles at the physical values of the regulator parameter and that all these poles can be removed by adding counterterms to the action, so that the coefficients of the monomials in the action become functions (power series) in the cut-off that diverge in the physical value of the regulator. The role of renormalization is to remove these infinities in such a way so as to make parameters of the lagrangian give experimentally observable results. This is done by the use of the renormalization conditions. The use of renormalization conditions is justified by the argument that the parameters in the "bare" lagrangian or unobservable and as such are unphysical so it doesn't if these parameters are infinite as long as when the renormalization conditions are imposed, experimentally measurable quantities have well defined finite values.

Using the toy model of ϕ^3 theory to illustrate the above points. As the divergences can occur in any graph of 1-loop or higher, it is useful to look at the action of ϕ^3 rather than individual diagrams. The tree level action for ϕ^3 is given by

$$S_{\text{Tree}} = \int dx \left[\frac{1}{2}\phi(x) \left(\partial_x^2 + m^2 \right) \phi(x) + \frac{g}{3!}\phi(x)^3 \right] \tag{2.60}$$

As the tree level of the perturbative expansion corresponds to the classical case, it is required that the parameters satisfy the renormalization conditions at the tree level. The renormalization conditions are chosen to be:

$$\Gamma^{(2)}(p^2 = -m^2) = 0, \qquad \frac{d}{dp^2}\Gamma^{(2)}(p^2 = -m^2) = 1, \qquad \Gamma^{(3)}(r, s, t) = -g; \tag{2.61}$$

where the momenta satisfies

$$r + s + t = 0, \qquad p^2 = r^2 = s^2 = t^2 = -m^2, \qquad r \cdot s = s \cdot t = t \cdot r = \frac{m^2}{2} \tag{2.62}$$

It should be noted that although it is a requirement for the 2-point function to vanish at $p^2 = -m^2$, this is only true when in Minkowski space as in Euclidean space $p^2 > 0$. As such the renormalization point has to be satisfied upon the Wick rotation to Minkowski and not in Euclidean space itself.

The renormalization conditions above are an example of the *MOM-schemes* or momentum subtractions.

Any number of different renormalization conditions can be chosen but this is the most common for ϕ^3 as it ensures that the renormalized two-point function remains on-shell in Minkowski space and the three-point function is renormalized at the symmetry point $p_r \cdot p_s = \frac{1}{3}(3\delta_{rs} - 1)m^2$. This arbitrariness in renormalization is explored further in the section on renormaliztion group.

The solutions to the quantum theory must satisfy (2.61) to all orders in $\hbar$, which the action S_{Tree} doe. As such, the renormalized action will be defined to be S_{Tree}.

As the renormalized action cannot be the same as the bare action for the theory, the bare action S_B will be written as

$$S_B = \int dx \left[\frac{1}{2}\phi(x) Z_B \left(\partial_x^2 + m_B^2\right)\phi(x) + \frac{g_B}{3!}\phi(x)^3\right] \tag{2.63}$$

where

$$Z_B = 1 + \delta Z, \qquad m_B^2 = m^2 + \delta m^2, \qquad g_B = g + \delta g. \tag{2.64}$$

The wavefunction renormalization parameter Z_B is introduced in front of the kinetic term as although their is freedom to rescale the field $\phi(x)$ to absorb any such factors, it is needed to ensure the same fields are expressed in the bare action as in the renormalized action.

2.6.1 Taylor Series subtraction

The Taylor series subtraction scheme is based observation that $I(\mathcal{G})$ can be viewed as a function of the independent external momenta p and as such $\mathcal{G}$ can be differentiated with respect to the external momenta $I(\partial\mathcal{G}) = \frac{\partial I(\mathcal{G})}{\partial p_\mu}, I(\partial^2\mathcal{G}) = \frac{\partial^2 I(\mathcal{G})}{\partial p_\mu \partial p_\nu}$, etc.. With this observation $I(\mathcal{G})$ can be expanded as Taylor's theorem with a fixed subtraction point P_0

$$I(P) = I\left(P_0\right) + \sum_{\mu=1}^{N} \int_{P_0}^{P} dP'_\mu \frac{\partial}{\partial P_\mu} I(P')$$

which can then iterated further

$$\begin{aligned} I(P) &= \sum_{r=0}^{n} \frac{1}{r!} \sum_{\mu_1=1}^{N} \cdots \sum_{\mu_r=1}^{N} \frac{\partial I_(P_0)}{\partial P_{\mu_1} \cdots \partial P_{\mu_r}} (P-P_0)_{\mu_1} \cdots (P-P_0)_{\mu_r} \\ &\quad + \sum_{\mu_1=1}^{N} \cdots \sum_{\mu_n=1}^{N} \int_{P_0}^{P} dP_{\mu_1}^{(1)} \cdots \int_{P_0}^{P} dP_{\mu_1}^{(1)} \frac{\partial I(P_n)}{\partial P_{\mu_1} \cdots \partial P_{\mu_n}} \\ &= T^n I(P) + \int_{P_0}^{P} dP_1 \cdots dP_n \partial^n I(P_n) \end{aligned} \tag{2.65}$$

where T^n is the Taylor series up to to order n acting on the integral $I(P)$. This becomes useful upon the realisation that that differentiating a Feynman graph with respect to some independent external momenta lowers the overall degree of divergence of the graph

$$\deg(\partial^n \mathcal{G}) \leq \deg(\mathcal{G}) - n \tag{2.66}$$

This can be verified by observing that as the differentiation is with respect to some independent external momenta, ∂ commutes with all the internal loop momenta of $\mathcal{G}$ which itself is a product of vertices, which in general are polynomials in all momenta, and edges, which are of the form $[(k+p)^2 + m^2]^{-1}$. This gives rise to the relation $\partial I(\mathcal{G}) = \partial \int d^{2\omega}k i(\mathcal{G}) = \int d^{2\omega}k \partial i(\mathcal{G})$, where $i(\mathcal{G})$ is the integrand of the integral $I(\mathcal{G})$. Which then allows the power counting degree of a differentiated graph to be calcualted:

$$\deg(\partial\Delta) = \deg\left(\partial[(k+p)^2 + m^2]^{-1}\right) = \deg\left(\frac{-2(k+p)}{[(k+p)^2 + m^2]^2}\right) = -3 = \deg(\Delta) - 1$$

As the sum of degrees of propagators is defined to be the largest degree of each term, (2.66) has therefore been established.

As each term in the Taylor expansion lowers the degree of divergence, each term is power-counting convergent and the Taylor series itself is absolutely convergent. Therefore it can be used to renormalize divergent Feynman graph integrals. The Taylor series terminates after a finite number of differentiations if the graph is a polynomial in the external momenta.

An important thing to note is that differentiating a graph with respect to an external momenta does not change the graphical structure of $\mathcal{G}$, as such the differentiated vertices and edges can be viewed to be associated with some new Feynman rule, i.e. the topology of the graph remains the same but each differentiated edge or vertex is denoted by a cross.

2.6.2 MS and $\overline{MS}$

The Minimal Subtraction (MS) scheme introduced by 't Hooft [82] is a very simple renormalization prescription that is tied intimately with dimensional regularization. It is well known that dimensionally regulated integrals have simple poles when the power counting degree of divergence is a non-negative multiple of 2. The MS scheme simply subtracts away the pole from the Feynman graph integrand at the critical dimension of the theory (i.e. $2\omega = 6$ for ϕ_6^3 or $2\omega = 4$ for QED). The coupling constant gains an additional factor $g \to \mu^{\omega_0 - \omega} g$ where ω_0 is the physical dimension of the theory. This factor of μ is there to ensure that no additional mass–scales are introduced when the theory is evaluated in dimensions other than the critical one. $\overline{MS}$ was introduced to remove the constant factors $\ln(4\pi)$ and γ, the Euler-Mascheroni constant, which appear and is related MS by $\mu = \bar{\mu} \left(\frac{e^\gamma}{4\pi}\right)^{\frac{1}{2}}$.

2.7 BPHZ Theorem

As discussed previously, the infinities in the integrals for processes of perturbative QFT can be made finite with the use of counterterms. The counterterms in the language of Feynman graphs are constructed from a combination of different graphs which only produce a finite result when all Feynman graphs to a given Green's function to a given order in the loop expansion are summed.

What would be useful for both conceptual and calculational ease would be to have a method which can used on a graph by graph basis instead of having to consider entire Green's functions, this method is the BPH theorem. The BPH theorem was first devised by Bogoliubov and Parasiuk in 1957 but part of the proof was based on a lemma which later turned out to be untrue. In 1966 Hepp corrected and expanded upon the work done by Bogoliubov and Parasiuk and in 1969 Zimmerman expanded the recursion of BPH explicitely into the forest formula, this was done by showing that Taylor subtractions in the integrand could be made so as to make all the integrals convergent in Euclidean space. Hepp had already shown that for massive particles the Minkowski space Green's functions exist as distributions if the Euclidean space ones are uniformly convergent.

The BPHZ theorem states that for any local quantum field theory the divergences present in any given Feynman graph $\mathcal{G}$ are shown to be local (polynomial in the external momenta) and thus can be absorbed into counterterms that are monomials in fields and derivatives. The Z part of BPHZ does not need an explicit regulator, but it implicitly needs one to show that it has anything to do with the original field theory, and thus that it leads to a unitary, causal, and local theory.

Before the structure of the BPHZ theorem is discussed, which is defined by the

R-Operation, some new notation is introduced to help distinguish parts of the graph, this is usually done by boxes, but quickly leads to problems with non-planar topologies and with disconnected subgraphs. A spinney $\mathcal{S}$ is defined to be a covering of a graph by a set of disjoint 1PI subgraphs including single vertices. Spinneys can be shrunk into effective vertices, this graph is designated $\mathcal{G}/\mathcal{S}$. By construction all Henges are Spinneys but not all Spinneys are Henges.

Figure 2.4: An example of a Spinney and the associated $\mathcal{G}/\mathcal{S}$, the lines in $\mathcal{S}$ are in black and the lines in $\mathcal{G}/\mathcal{S}$ are in shown in red.

The set of all Spinneys is defined to be a Wood $\mathcal{W}$. A Wood with the Spinney $\mathcal{S} = \mathcal{G}$ removed is defined to be the Proper Wood $\bar{\mathcal{W}}$.

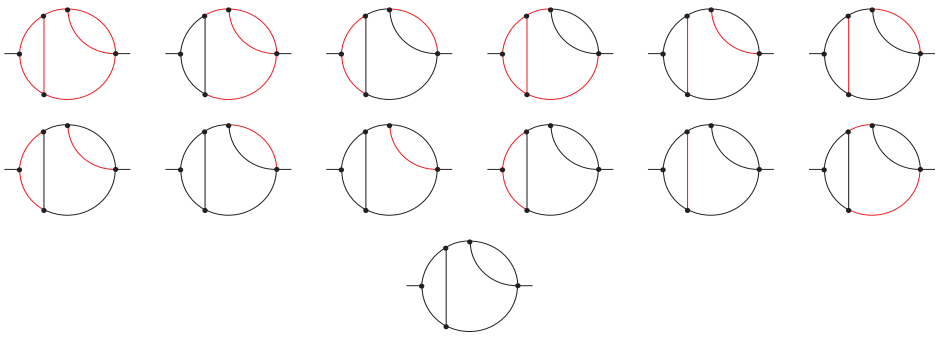


Figure 2.5: The Wood of a a 3-loop contribution to the 2-point function of a generalised scalar theory. The black lines represent the lines included in the Spinney. (The Proper Wood is the above set of graphs with the last one removed.)

2.8 The R-Operation

For any Feynman Graph, $I(\mathcal{G})$ an operation to remove the divergences from this graph can be defined. From the study of Feynman Graphs of 2-loop order or greater it is known that it is not enough to only consider the overall divergence, the subdivergences need to be considered as well which can exist in the subgraphs of $I(\mathcal{G})$. With this in mind the R-Operation is defined to be the operator which subtracts the overall divergence and all the subdivergences recursively from the Feynman Graph $I(\mathcal{G})$. This is done by first isolating the subdivergences of $\mathcal{G}$, then removing these subdivergences along with the overall divergence of $\mathcal{G}$ in the form of subtractions.

Two more operators will also need to be defined, the first will be the operator which subtracts the subdivergence from a Feynman Graph $I(\mathcal{G})$, this we denote as the $\bar{R}$-Operator. The second operator will be a subtraction operator which removes the divergent part of the Graph $I(\mathcal{G})$, this operator is the subtraction operator K. With the use of these defintitions the R-Operator can be constructed as follows:

$$RI(\mathcal{G}) = (1 - K)\bar{R}I(\mathcal{G})$$

Where it should be noted that the operator $1-K$ subtracts the overall divergence from the Feynman Graph $I(\mathcal{G})$. How $KI(\mathcal{G})$ removes the divergences is dependent on the choice of regularization scheme, for example in the Dimensional Regularization, $KI(\mathcal{G})$ subtracts the pole terms in the Laurent expansion of the (dimensionally regularized) Feynman Integral $I(\mathcal{G})$ or, $KI(\mathcal{G})$ can be the Taylor series subtraction T of the Feynman integral $I(\mathcal{G})$ (this method is independent of choice of regulator but is only valid in Euclidean space.).

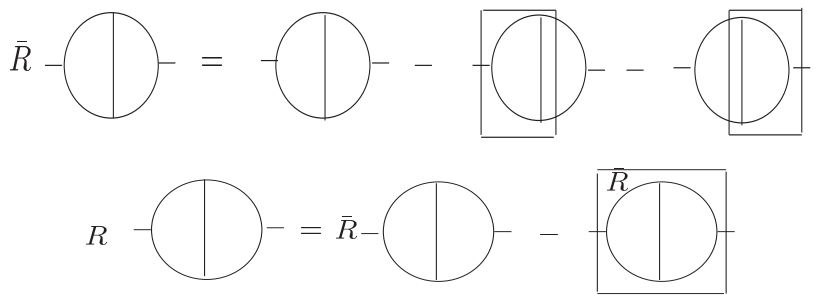


Figure 2.6: A Graphical example of the R-Operation

2.8.1 Bogoliubov's Definition of the R-operation.

Having motivated the R-Operation, the definition of R and $\bar{R}$ are now shown

$$R_B I(\mathcal{G}) \equiv (1 - K)\,\bar{R}_B I(\mathcal{G}) = \sum_{\mathcal{S}\in\mathcal{W}(\mathcal{G})} I\left(\mathcal{G}/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}_B I(\Gamma)\right) \tag{2.67}$$

$$\bar{R}_B I(\mathcal{G}) \equiv \sum_{\mathcal{S}\in\bar{\mathcal{W}}(\mathcal{G})} I\left(\mathcal{G}/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}_B I(\Gamma)\right) \tag{2.68}$$

The notation $I\left(\mathcal{G}/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} f(\Gamma)\right)$ means that when the integral I is constructed for the diagram, each of the 1PI subgraphs $\mathcal{S} \ni \Gamma \subset \mathcal{G}$ has been replaced by the modified expression $f(\Gamma)$.

This definition of the R and the $\bar{R}$-Operation is purely graphical in nature, where

the $\star$ operator acts by replacing the subgraph by its subtracted version.

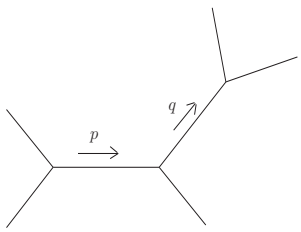
2.9 Henge Decomposition

The method of Henge Decompostion is due to Caswell-Kennedy [86, 87, 88] and is motivated from the asymptotic behaviour of Feynman integrals, which seeks to resolve the ambiguity of the momenta in a Feynman graph which has an overlapping divergence in a systematic way. Henge Decomposition works by carving up the momentum-space which the Feynman integral inhabits in some non-trivial manner with the use of step functions, where each section of momentum-space contains a well behaved integral which is independent of any arbitrary choice of momentum routing. This will lead to a series of Feynman integrals with restricted integral boundaries which when summed over has to give back the original integral:

$$I_{\mathcal{G}} = \sum_{\ell \in \mathcal{G}} I(\mathcal{G}, l) \tag{2.69}$$

The choice of dividing momentum-space in such a way leads to the fact that in each section there has to be an edge which holds the smallest momentum ℓ, which leads to the solution of the problem of overlapping divergences. This is because each section can only have one edge which has the smallest momenta ℓ, there can be no ambiguity over the rate at which each momenta in an edge which contains multiple internal momenta grow large.

To see how this will work, first consider the tree graph:



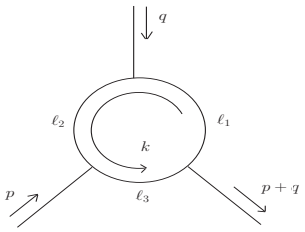
which is expressed by:

$$I_{\mathcal{G}} = \frac{1}{[p^2 + m^2][q^2 + m^2]} \tag{2.70}$$

In order to analyze the behaviour of this integral, inequalities are used to gain information on the bounds of the integral:

$$\begin{aligned} I_{\mathcal{G}} &= \sum_{\ell \in \mathcal{G}} I(\mathcal{G}, l) \\ &= I_{\mathcal{G}}(p,q) \cdot \theta(p > q) + I_{\mathcal{G}}(p,q) \cdot \theta(q > p) \\ |I_{\mathcal{G}}| &\leq \max(q,m)^{-2}\theta(p > q) + \max(p,m)^{-2}\theta(q > p) \\ &\leq \max(q,m)^{-2} + \max(p,m)^{-2} \end{aligned} \tag{2.71}$$

From this simple tree-level example, more complicated cases involving graphs with 1-loop (where we have dropped the overall constant factors) can now be considered:



from which is obtained:

$$I_{\mathcal{G}} = \int \frac{d^{2\omega}k}{[k^2+m^2][(k+p)^2+m^2][(k-q)^2+m^2]} \tag{2.72}$$

Which with the introduction of step functions to restrict the momentum-space regions gives for one particular region where ℓ_1 carries the smallest momentum $k_1 = k - q$:

$$I(\mathcal{G}, \ell_1) = \int \frac{d^{2\omega}k_1 \theta[(k_1+q)^2 - k_1^2]\theta[(k_1+q+p)^2 - k_1^2]}{[k_1^2+m^2][(k_1+q)^2+m^2][(k_1+q+p)^2+m^2]} \tag{2.73}$$

This process is then repeated for all other choices of edge to carry the smallest momentum (which corresponds to the remaining regions of momentum-space).

The choice in using the step functions to divide up the momentum-space is due to inability of applying the bounding inequality directly to the full integral. It is, however, possible to find a bound on the full integral when the bounds of each of the separate regions of momentum-space are combined.

The example can simplified by realising that in the region of integration $(k_1+q)^2+m^2$ and $(k_1+q+p)^2+m^2 \geq k_1^2+m^2$

$$I(\mathcal{G}, \ell_1) \leq \int \frac{d^{2\omega}k_1}{[k_1^2+m^2]^3} \tag{2.74}$$

The integral can split up the momentum-space further still by separating the integral into UV safe and UV sensitive regions:

$$I(\mathcal{G},\ell_1) = \int_0^m \frac{d^{2\omega}k_1}{[k_1^2+m^2]^3} + \int_m^\infty \frac{d^{2\omega}k_1}{[k_1^2+m^2]^3} \tag{2.75}$$

The bound on the integral can then be applied by replacing k_1^2 and m^2 by the larger of k_1^2 or m^2

$$I(\mathcal{G},\ell_1) \leq \int_0^m \frac{d^{2\omega}k_1}{m^6} + \int_m^\infty \frac{d^{2\omega}k_1}{k^6} \tag{2.76}$$

Setting $2\omega = 4$ and evaluating the integral in polar coordinates gives:

$$I(\mathcal{G},\ell_1) \leq \frac{3S_3}{4m^2}$$

with S_3 being the surface area of a 3-sphere. Looking at the symmetry of the graph, it can be seen that the other 2 regions will give the exact same bounds which leads to the result of the bound for the full integral being $I(\mathcal{G}) \leq c \times m^{-2}$, where c is some positive definite constant, which in the case above will be $c = \frac{9S_3}{4}$.

Continuing to the multi-loop case an additional parameter is introduced to make the recursive bounding of arbitrary Feynman graphs be in a suitable form, this additional parameter is needed for the inductive proof of the BPHZ theorem . This parameter is a small-momentum cut-off λ which all momenta in the edges will be greater than.

$$I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} I_\lambda(\mathcal{G},\ell) \tag{2.77}$$

Once the bound for $I_\lambda(\mathcal{G})$ has been found, the bound for the full integral $I(\mathcal{G})$ is obtained by setting $\lambda = 0$.

At this point a new notation is introduced which will be needed to discuss multi-loop graphs, the Henge. A henge $\mathcal{H}$ is defined to be the set of 1PI subgraphs and single vertices of $I(\mathcal{G})$ that remain after the removal of a selected edge $\ell \in \mathcal{G}$ from $I(\mathcal{G})$. This definition allows for simple graphical representation of the henge which also best illustrates this definition, figure 2.8 shows the set of all henges for a 3-loop graph with the henges represented by bold lines. The graph $\mathcal{G}$ can be mapped to a single loop, corresponding to $\ell \in \mathcal{G}|\ell \notin \mathcal{H}$, and a set of effective vertices, which are the 1PI subgraphs and single vertices (Θ) which belong to the Henge which are then shrunk to single points. This graph is designated $\mathcal{G}/\mathcal{H}$ and is graphically represented by a blob for the effective vertex as shown in figures 2.7 and 2.9.

With these definitions, the splitting of momentum-space or "Henge Decomposition"

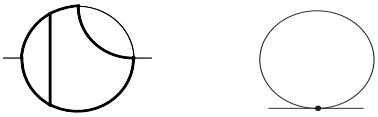


Figure 2.7: A Henge with its associated $\mathcal{G}/\mathcal{H}$.

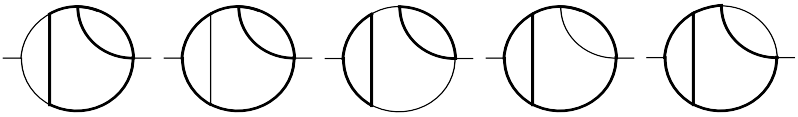


Figure 2.8: The set of all henges for a 3-loop contribution to the 2-point function of a generalised scalar theory. The choice of using a generalised theory is to demonstrate that this method is valid for all quantum field theories that can be represented by graphs.

for an arbitrary Feynman Graph to be expressed as

$$I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dk\, i_k\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} I_k(\Theta)$$

The loop is the term $i_k\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right)$, and the effective vertices are contained within $\prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} I_k(\Theta)$. The recursive nature of this definition for the integral $I(\mathcal{G})$ is apparent as for each henge $\mathcal{H}$ an edge $\ell \in \mathcal{H}$ can be selected for each $\Theta \in \mathcal{H}(\mathcal{G},\ell)$ which will then define a new henge, if one exists, in the graph Θ. This recursion ends when $\mathcal{H}(\mathcal{G},\ell) = \emptyset$, i.e. a henge can no longer be defined. This termination of the recursion is shown in figure 2.10.

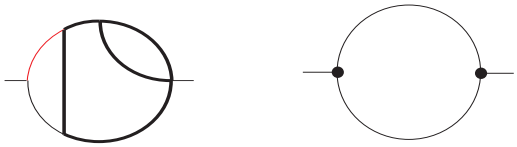


Figure 2.9: Graphical example of a particular Henge Decomposition of a 3-loop contribution to the 2-point function of a generalised scalar theory. This works by selecting any line $\ell \in \mathcal{G}$ (represented by the red lines in the above graphs), this defines a Henge $\mathcal{H}$ which is shown in bold. The subdivergence of $I(\mathcal{G})$ for this particular choice of ℓ is now contained within the Henge shown in the left graph above. The graph on the right shows the graph $\mathcal{G}/\mathcal{H}$ and is the graph corresponding to $i_k\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right)\prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} I_k(\Theta)$.

Once the smallest henge is obtained, the same methods used in the 1-loop example can be applied to get the bounds of the sub-loop.

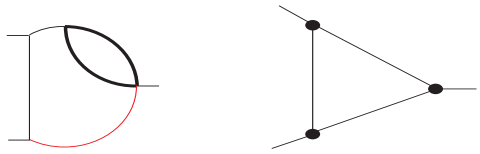


Figure 2.10: The graph on the left shows the topology of graph contained in the term $I_k(\Theta)$, which in our recursive definition forms a new graph $\mathcal{G}_1$ from which we have selected an edge $\ell \in \mathcal{G}_1$ (which is highlighted red) which defines a new henge shown by the double edges. For the Graph shown the recursion ends as we can no longer define a new $1PI$ Henge as $\mathcal{H}(\mathcal{G}_1, \ell)$ is a 1-loop 3-point function. We continue this process performing a summation of all possible choices of ℓ, and thus we have disentangled all the overlapping subdivergences within $I(\mathcal{G})$.

With these new definitions along with the methods for finding the bounds on the sub-loops, it is now possible to present the general proof of Henge decomposition for an overall convergent Feynman graph. To do so, it has to be shown that:

$$I_\lambda(\mathcal{G}) \leq c \times max(m,\lambda)^{\deg(\mathcal{G})} = \begin{cases} \lambda^{\deg(\mathcal{G})}; \\ m^{\deg(\mathcal{G})}. \end{cases} \tag{2.78}$$

What follows is an inductive proof on the number of loops L in $\mathcal{G}$, so it is assumed $I_\lambda(\Theta) \leq c \times max(m,\lambda)^{\deg(\Theta)}$ for all graphs Θ with less than L loops and which have negative overall degree. Only the case of $\lambda \geq m$ will be shown as the other bound is found in exactly the same way.

$$\begin{aligned} I_\lambda(\mathcal{G}) &= \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty d^{2\omega}k\, i_k[\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)] \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} I_k(\Theta) \\ &\leq c \times \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty d^{2\omega}k\, k^{\deg(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell))-2\omega} \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} k^{\deg(\Theta)} \\ &= c \times \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty d^{2\omega}k\, k^{\deg(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell))-2\omega+\sum_{\Theta\in\mathcal{H}(\mathcal{G},\ell)}\deg(\Theta)} \\ &= c \times \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty d^{2\omega}k\, k^{\deg(\mathcal{G})-2\omega} \\ &\leq c \times \lambda^{\deg(\mathcal{G})} \end{aligned} \tag{2.79}$$

As $i_k[\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)]$ is just a loop graph with effective vertices, this integral can be bounded with $k^{\deg(\mathcal{G}/\mathcal{H})-2\omega}$ using the tree-level inequalities derived earlier. The above proof also used the identity $\deg(\mathcal{G}) = \deg(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)) + \sum_{\Theta\in\mathcal{H}(\mathcal{G},\ell)}\deg(\Theta)$ which is assumed to be < 0.

This proof shows that if a Feynman graph $\mathcal{G}$ has an overall negative degree of

divergence and that all its 1PI subgraphs also have negative degree of divergence then it will be convergent.

As convergent integrals are only a part of the calculation of Feynman Graphs, a method for dealing with the divergent integrals will have to be considered which leads to the renormalization of the Feyman graphs within the henge decompostion procedure.

2.9.1 Kennedy-Caswell Definition of R-Operation

As in the Bogoliubov definition of the BPHZ theorem, the subtraction operator $(1-K)$ is used to remove the overall divergent part of the integral. At the 1-loop level all such subtractions are local and so can be removed simply, this is not the case in the multi-loop level as the divergent parts $KI(\mathcal{G})$ will not be local unless a way to remove subdivergences is found first.

In this representation of a Feynman Graph, the R-Operation is defined using the Henge decomposition method as this allows the isolation of all possible subdivergences in a systematic way.

$$I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dki_k \left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} I_k(\Theta) \tag{2.80}$$

With this definition the momentum decomposition of $I(\mathcal{G})$, the R-Operation can then be defined on it. As this method ensures that one of the edges carries the smallest momentum, and that this choice of an edge defines the Henge (which is shrunk to a set of effective vertices), this means that the subdivergences of the $I(\mathcal{G})$ must exist within the effective vertices.

This leads to:

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dki_k \left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} RI_k(\Theta) \tag{2.81}$$

Where the R-Operation which removes the overall divergence and the subdivergences recursively is defined as:

$$RI_\lambda(\mathcal{G}) \equiv \bar{R}I_\lambda(\mathcal{G}) - K\bar{R}I_0(\mathcal{G}) \tag{2.82}$$

Where it should be noted that the subtraction-operator acts upon $I(\mathcal{G})$ with zero small momentum cut-off λ, this is done to avoid complications involved with derivatives of the step functions which is justified when you consider that only the finite part of the integral is altered as $RI_k(\Theta)$ is finite by construction. When the recursion has finished, all the overlapping divergences which may exist within $I(\mathcal{G})$ have been resolved and the resulting overall result is manifestly finite. This is due to the Henge decomposition,

whose construction cannot contain overlapping divergences along with the use the BPH theorem which allows the systematic removal of all divergences contained within the graph. A graphical example of how R and $\bar{R}$ act upon a graph is shown in figure 2.11 and 2.12.

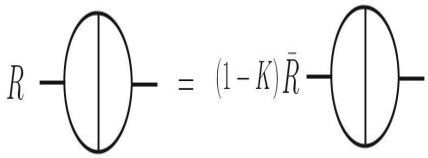


Figure 2.11: A graphical example of how$R$ acts upon a the overlapping 2-point 2-loop graph in ϕ_6^3.

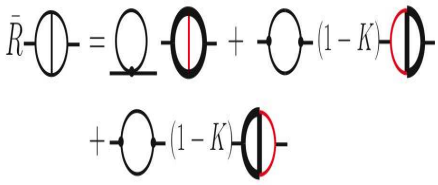


Figure 2.12: $\bar{R}$ acting upon the overlapping 2-point 2-loop graph in ϕ_6^3, where the thick line represents the henge and it can be seen that the last two terms correspond to the same topology.

2.9.2 The Equivalence of the 2 approaches

As there are now two different definitions of the R and the $\bar{R}$-Operation it is important to show these two definitions are truly equivalent.

In order for this to be done it is necessary to show that:

1. All subtractions made by Bogoliubov's representation of BPHZ are also done in the Caswell-Kennedy representation;

2. All subtractions made by the Caswell-Kennedy representation are also done by Bogoliubov's representation.

For Graphs with no loops or with just a single loop, the equivalence is trivial. For multiple loops the case is not so clear.

As such, for condition 2) to be true, it is observed that a henge $\mathcal{H}(\mathcal{G}, \ell)$ is just a special kind of spinney $\mathcal{S}$, moreover if a graph $\mathcal{G}$ has L loops in it then every henge

is an $L-1$ loop spinney, it should also be noted that every $L-1$ loop spinney is a henge. In order to show this, consider the Graph $\mathcal{G}$ from which a spinney is selected, the spinney is then shrunk to a set of effective vertices $\mathcal{G} \to \mathcal{G}/\mathcal{S}$. As both the graph $\mathcal{G}$ and the spinney $\mathcal{S}$ are $1PI$, the graph of effective vertices $\mathcal{G}/\mathcal{S}$ has to have a single loop which from the definition of a henge means that $\mathcal{H}(\mathcal{G},\ell) = \mathcal{S}$ for all $\ell \in \mathcal{G}/\mathcal{S}$. More than this, each $\mathcal{H}(\mathcal{G},\ell)$ can contain $1PI$ subgraphs which can form Spinneys. Therefore the set of all Spinneys $\mathcal{S}$ from the henge $\mathcal{H}(\mathcal{G},\ell)$ form a wood $\mathcal{W}(\mathcal{H}(\mathcal{G},\ell))$ and that for each $\mathcal{S} \in \mathcal{W}(\mathcal{H}(\mathcal{G},\ell))$ there will exist an equivalent Spinney from the proper wood of the graph $\mathcal{G}$, $\mathcal{S} \in \bar{\mathcal{W}}(\mathcal{G})$ as such the set of all ℓ for which $\mathcal{S} \in \mathcal{W}(\mathcal{H}(\mathcal{G},\ell))$ is exactly the graph $\mathcal{G}/\mathcal{S}$. Therefore this means that every subtraction made from the Caswell-Kennedy representation of BPHZ is also included in Bogoliubov's representation.

To prove 1), a Henge decomposition is performed on the graph of effective vertices $\mathcal{G}/\mathcal{S}$ until all that is left is a set of disjoint vertices v which contains both true vertices $v_{\mathcal{G}}$ (those that are vertices in $\mathcal{G}$) and the effective vertices of the shrunken spinneys $\mathcal{S}$.

An induction on the number of loops. Assume that for any graph $\mathcal{G}$ with less than L loops $RI_\lambda(\Gamma) = R_B I_\lambda(\Gamma)$ (inductive step).

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\mathcal{S}\in\bar{\mathcal{W}}(\mathcal{G})} I_\lambda\left(\mathcal{G}/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right). \tag{2.83}$$

and

$$\begin{aligned} RI_\lambda(\mathcal{G}) &= \bar{R}I_\lambda(\mathcal{G}) - K\bar{R}I_0(\mathcal{G}), \\ &= \sum_{\mathcal{S}\in\mathcal{W}(\mathcal{G})} I_\lambda\left(\mathcal{G}/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right). \end{aligned} \tag{2.84}$$

For an L loop graph, using equations eq.(2.81) and 2.83 to obtain:

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} \int_\lambda d^{2\omega}k_\ell \, i_{k_\ell}\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} \sum_{\mathcal{S}\in\mathcal{W}(\Theta)} I_\lambda\left(\Theta/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right). \tag{2.85}$$

All the subgraphs of $\mathcal{G}$ made from all the lines and vertices within the henge $\mathcal{H}(\mathcal{G},\ell)$ can be denoted as $\mathcal{H}(\mathcal{G},\ell) = \bigcup_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} \Theta$ and thus equation (2.85) can be rewritten as:

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\ell\in\mathcal{G}} \int_\lambda d^{2\omega}k_\ell \, i_{k_\ell}\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \sum_{\mathcal{S}\in\mathcal{W}(\mathcal{H}(\mathcal{G},\ell)} I_\lambda\left((\mathcal{H}(\mathcal{G},\ell)/\mathcal{S} \star \prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right). \tag{2.86}$$

since $\prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} \sum_{\mathcal{S}\in\mathcal{W}(\Theta)}$ may be replaced by $\sum_{\mathcal{S}\in\mathcal{W}(\mathcal{H}(\mathcal{G},\ell)}\cdot \ \ell \in \mathcal{G} \sum_{\mathcal{S}\in\mathcal{W}(\mathcal{H}(\mathcal{G},ell))}$ by $\sum_{\mathcal{S}\in\bar{\mathcal{W}}(\mathcal{G}} \sum_{\ell\in\mathcal{G}/\mathcal{S}}$ as the same terms appear in each one.

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\mathcal{S}\in\bar{\mathcal{W}}(\mathcal{G}} \sum_{\ell\in\mathcal{G}/\mathcal{S}} \int_\lambda d^{2\omega}k_\ell\, i_{k_\ell}\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \sum_{\mathcal{S}\in\mathcal{W}(\mathcal{H}(\mathcal{G},\ell)} I_\lambda\left((\mathcal{H}(\mathcal{G},\ell)/\mathcal{S}\star\prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right). \tag{2.87}$$

since $\ell \in \mathcal{G}/\mathcal{S}$, $\mathcal{S} \subset \mathcal{H}(\mathcal{G},\ell)$ and thus $i_k\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) = i_k\left((\mathcal{G}/\mathcal{S})/\mathcal{H}(\mathcal{G},\ell)\right)$ and $I\left(\mathcal{H}(\mathcal{G},\ell)/\mathcal{S}\right) = I\left(\mathcal{H}(\mathcal{G}/\mathcal{S},\ell)/\mathcal{S}\right)$, so

$$\sum_{\ell\in\mathcal{G}/\mathcal{S}} \int_\lambda d^{2\omega}k_\ell\, 1_{k_\ell}\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) I_\lambda\left((\mathcal{H}(\mathcal{G},\ell)/\mathcal{S}\star\prod_{\Gamma\in\mathcal{S}} -K\bar{R}I_0(\Gamma)\right) = I_\lambda\left(\mathcal{G}/\mathcal{S}\star -K\bar{R}I_0(\Gamma)\right). \tag{2.88}$$

by the usual momentum space decomposition (some line has to carry the smallest momentum). Setting $\lambda = 0$, will lead to the desired result.

$$I_\lambda\left(\mathcal{G}/\mathcal{S}\right) = \sum_{\ell\in\mathcal{G}/\mathcal{S}} \int_\lambda^\infty dk i_k\left(\left(\mathcal{G}/\mathcal{S}\right)/\mathcal{H}(\mathcal{G}/\mathcal{S},\ell)\right) \prod_{\theta\in\mathcal{H}(\mathcal{G}/\mathcal{S},\ell)} I_k(\theta)$$

This set of disjoint vertices is then mapped onto the original graph $\mathcal{G}$. For the two definitions of BPHZ to be equivalent, it is required that the following functions are true:

$$\Phi = \begin{cases} \forall \ell \in \mathcal{G}/\mathcal{S} \mapsto \Phi(\ell) = \ell \in \mathcal{G}; \\ \forall v \in \mathcal{G}/\mathcal{S} \mapsto \Phi(v) = \theta \in \mathcal{S}. \end{cases}$$

$$\Phi' = \begin{cases} i_k \mapsto i_k; \\ I_\lambda(v) \mapsto \begin{cases} \text{iff } \Phi(v)) \in v_{\mathcal{G}} \text{ then } I_\lambda(\theta(v)); \\ \text{then } -K\bar{R}I_0\Phi(v). \end{cases} \end{cases}.$$

Where the function Φ takes a graph $\mathcal{G}/\mathcal{S}$ and maps it to a graph $\mathcal{G}$ and the function Φ' maps the the lines and disjoint vertices onto the graph $\mathcal{G}$.

What this means is that when the disjoint vertices are mapped on to the graph $\mathcal{G}$, lines in the shrunken graph remain lines, true vertices remain true vertices but effective vertices, made by shrinking the spinney to a point, get replaced with $-K\bar{R}I_0\Phi(v)$. As this can be done for any spinney, the entire Wood is covered.

$$\Phi' \equiv \mathcal{G}/\mathcal{S}\star\prod_{\theta\in\mathcal{S}} -K\bar{R}\theta(\mathcal{S})$$

$$\bar{R}_B(\mathcal{G}) = \sum_{\mathcal{S}\in\bar{\mathcal{W}}} \Phi'(\mathcal{G}/\mathcal{S})$$

As a spinney $\mathcal{S}$ is a purely graphical object, it can contain no momenta thus per-

forming the Henge decomposition on $\mathcal{G}/\mathcal{S}$ covers all of momentum space and as such we have performed a full henge decomposition proving that all subtractions made by Bogoliubov's representation of BPHZ are contained within the henge decomposition representation of the BPHZ theorem.

It has thus been shown that both representations of the BPHZ theorem or indeed the same. All that is left to do is to show that all subtractions made by the Caswell-Kennedy representation of BPHZ combinatorially form counterterms.

2.9.3 Equivalence to Counterterms

Now that the case has been made for the use of local subtractions, it is now important to show that they are equivalent to counterterms in order to justify their use. What makes this proof necessary is that the equivalence between BPHZ local subtractions and counterterms is not obvious for two reasons:

1. Symmetry factors arising from identical particles have to be handled in the correct way.

2. A given countergraph may include subtractions from several different graphs, which is to say that a countergraph corresponds to the cancellation of divergences for a scattering process rather than on an individual Feynman graph basis.

In order to solve this problem the generating functional will be used to ensure all symmetries are preserved, which leaves the proof that the combinatorial factors for the BPHZ subtractions are such that they are exactly the same as the subtractions generated by the counterterm.

$$Z(J) = \int d\phi e^{-S(\phi)+J\phi} = \mathcal{N} exp\left[-S_I(\frac{\delta}{\delta J})\right] e^{\frac{1}{2}J\Delta J} \tag{2.89}$$

A perturbative expansion of the generating functional can be made in terms of the number of vertices in a graph $\mathcal{G}$

$$Z(J) = \mathcal{N}\sum_{n=0}^{\infty}\frac{(-)^n}{n!}\left[S_I\left(\frac{\delta}{\delta J}\right)\right]^n e^{\frac{1}{2}J\Delta J} = \mathcal{N}\sum_{n=0}^{\infty}\frac{(-)^n}{n!}\sum_{\mathcal{G}_n} I\left(\mathcal{G}_n(J)\right); \tag{2.90}$$

where the final sum is over all graphs $\mathcal{G}_n$ which contain exactly n vertices and source term J is attached to each of the external legs. The combinatorial wieghts for each graph is then handled automatically as the weight is defined by how many times the graph appears in the usual graphical expansion.

The renormalized generating functional is then defined as

$$\begin{aligned} RZ(J) &\equiv \mathcal{N}\sum_{n=0}^{\infty}\frac{(-)^n}{n!}\sum_{\mathcal{G}_n} RI\left(\mathcal{G}_n\right) \\ &= \mathcal{N}\sum_{n=0}^{\infty}\frac{(-)^n}{n!}\sum_{\mathcal{G}_n}\sum_{\mathcal{S}\in\mathcal{W}(\mathcal{G}_n)} I\left(\mathcal{G}/\mathcal{S}\star\prod_{\Gamma\in\mathcal{S}} -K\bar{R}I(\Gamma)\right). \end{aligned} \tag{2.91}$$

$\sum_{\mathcal{G}_n} RI\left(\mathcal{G}_n\right)$ can be expressed in terms $\sum_{\mathcal{G}_n} -K\bar{R}I\left(\mathcal{G}_n\right)$ as the sum over all partitions of the n vertices by a set of disjoint vertices. A given cover consists of r_j sets of j vertices where there is an insistence that $\sum_j^n r_j = n$.

The idea is that each $\Gamma \in \mathcal{S}$ includes some subset of the n vertices in $\mathcal{G}_n$, and these subsets are necessarily disjoint. A sum over all the ways (if any!) of choosing edges to connect the vertices into 1PI subgraphs will still have to be made.

This leads to the identity

$$\sum_{\mathcal{G}_n}\sum_{\mathcal{S}\in\mathcal{W}(\mathcal{G}_n)}\prod_{\Gamma\in\mathcal{S}} -K\bar{R}I(\Gamma) = \sum_{\substack{r_0,\ldots,r_n \\ r_0+\cdots+r_n=n}} \frac{n!}{\prod_{j=0}^n (j!)^{r_j}\, r_j!}\prod_{j=0}^{n}\left[\sum_{\mathcal{G}_j} -K\bar{R}I(\mathcal{G}_j)\right]^{r_j} \tag{2.92}$$

Using this identity on $RZ(J)$ and using that $\left[\sum_{\mathcal{G}_j} -K\bar{R}I(\mathcal{G}_j)\right]^{r_j}$ can be replaced by $\left[\sum_{\mathcal{G}_j} -K\bar{R}I(\mathcal{G}_j(\frac{\delta}{\delta J}))\right]^{r_j} e^{\frac{1}{2}J\Delta J}$, as the function $-K\bar{R}I(\mathcal{G}_j)$ can be viewed as a function of the sources J attached to its external legs. Because of the K all these sources are at the same point in configuration space. The following is then obtained:

$$\begin{aligned} RZ(J) &= \mathcal{N}\sum_{n=0}^{\infty}\frac{(-1)^n}{n!}\sum_{\substack{r_0,\ldots,r_n \\ r_0+\cdots+r_n=n}} \frac{n!}{\prod_{j=0}^n (j!)^{r_j}\, r_j!}\prod_{j=0}^{n}\left[\sum_{\mathcal{G}_j} -K\bar{R}I(\mathcal{G}_j(\frac{\delta}{\delta J}))\right]^{r_j} e^{\frac{1}{2}J\Delta J} \\ &= \mathcal{N}\prod_{j=0}^{\infty}\sum_{r_j=0}^{\infty}\frac{1}{r_j!}\left[\frac{1}{j!}\sum_{\mathcal{G}_j} K\bar{R}I(\mathcal{G}_j(\frac{\delta}{\delta J}))\right]^{r_j} e^{\frac{1}{2}J\Delta J} \\ &= \prod_{j=0}^{\infty}\exp\left[\frac{1}{j!}\sum_{\mathcal{G}_j} K\bar{R}I(\mathcal{G}_j(\frac{\delta}{\delta J}))\right] e^{\frac{1}{2}J\Delta J} \\ &= \mathcal{N}\exp\left[K\bar{R}e^{S_I(\frac{\delta}{\delta J})}\right] e^{\frac{1}{2}J\Delta J}. \end{aligned} \tag{2.93}$$

which leads to, as claimed

$$RZ(J) = \int d\phi e^{-S_B(\phi)+J\phi} \tag{2.94}$$

where the action is

$$S_B(\phi) = \frac{1}{2}\phi\Delta^{-1}\phi - K\bar{R}e^{S_I(\phi)} \tag{2.95}$$

As can be seen in figures 2.13 and 2.14, there is no one-to-one correspondance between BPHZ subtractions and counterterms but the combinatorial factors arrange themselves correctly. And it is with this relation to countergraphs which makes the BPHZ theorem so powerful, as figure 2.14 shows that subtractions made by BPHZ are part of a counter graph for ϕ^3 2-point function, but that the counter graph contains information about all possible contributions to the 2-point function, whereas BPHZ allows the calulation of processes on a graph by graph basis.

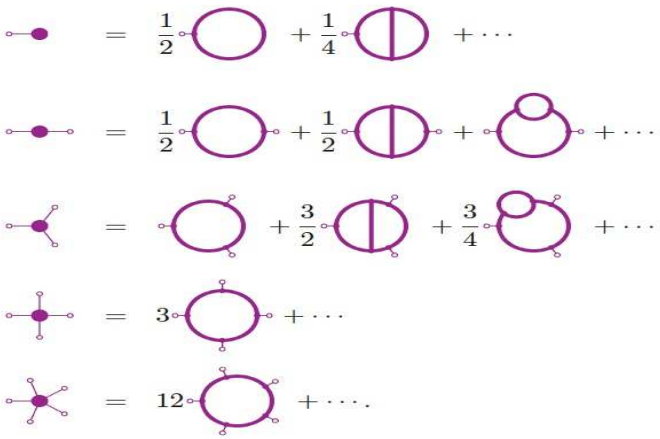


Figure 2.13: Graphical representation of counter terms (left) and the associated subtraction terms (right) in ϕ^3 theory in 2ω [89, 90].

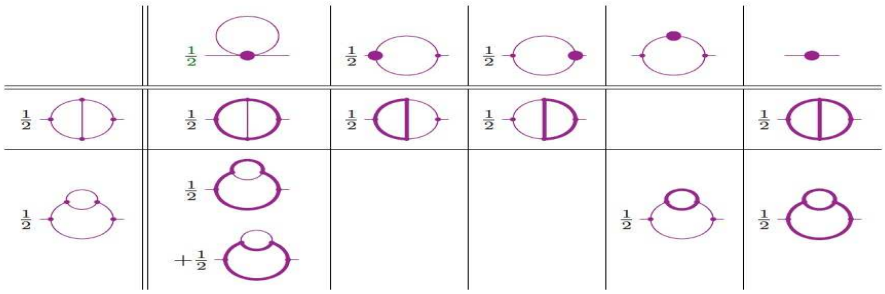


Figure 2.14: This figure shows the necessary henge graph subtractions needing to cancel the divergences in the graphs (y-axis) which correspond to the counter graphs (x-axis) [89, 90].

2.10 Sector Decomposition

Sector decomposition was first introduced by Hepp to deal with the problem of overlapping divergences and is tied intrinisically to Feynman parametric representation of integrals and dimensional regularization, as such the renormalization scheme of $\overline{MS}$ is a natural choice for this method. Sector decomposition takes a constructive approach to the problem of overlapping divergences by imposing a complete ordering on the Feynman parameters on the parametric representation of a Feynman integral. By imposing such an order we are splitting the Feynman parameter space into distinct regions where there are manifestly no overlapping divergences.

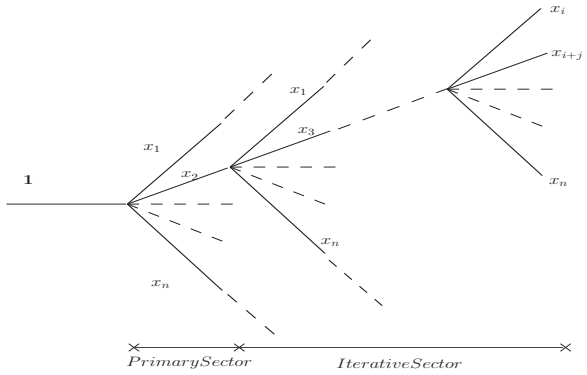


Figure 2.15: A graphical illustration of a sector decomposition tree to help visualize the process of method.

The method of sector decomposition is split into four steps:

- Primary sector - which removes the δ function from the parametric integral.
- Iterative sector - which proceeds to split the integral into unique topological sectors.
- Subtractions are made on each of the sector integrals and the pole terms are isolated.
- The coefficients of the poles are calculated.

These are mainly done by numerical methods as the form of the finite part and the sheer number of sector integrals to be done, demonstrated in fig.(2.15), means sector decomposition is ideally suited for automated processes.

2.10.1 Sector Decomposition: Primary Sector

The first step or primary sector decomposition is to impose an order on all the Feynman parameters in (2.59) with the aim of the eliminating the δ-function. For N feynman

parameters with N choices in how to remove the δ-function, these are our primary sectors. Our decomposition will take the form

$$\int d^N x = \sum_{l=1}^{N} \int d^N x \prod_{j=1}^{N} \theta(x_l \geq x_j) \tag{2.96}$$

with the θ-function defined to be

$$\theta(x_l \geq x_j) = \begin{cases} 1 & \text{if } x \geq y \text{ is true;} \\ 0 & \text{otherwise.} \end{cases}$$

which when when integrated over x_l and mapping the integral over the unit cube to a sum of N $(N-1)$-dimensional parameter integrals (2.59) becomes of the form

$$I_{\mathcal{G}_l} = \frac{\Gamma\left(\frac{1}{2}\right)^{2\omega} \Gamma(N - \omega L)}{\prod_{i=1}^{N} \Gamma(\alpha_i)} \int_0^1 \prod_{i=1}^{N-1} dx_i dx_i x_i^{\alpha_i - 1} \delta\left(1 - \sum_{l=1}^{N} x_l\right) \frac{\mathcal{U}_l^{N-(L+1)\omega}}{\mathcal{F}_l^{N-\omega L}} \tag{2.97}$$

where $I_{\mathcal{G}} = \sum_l^N I_{\mathcal{G}_l}$

To demonstrate how this method works an example will be taken from ϕ_6^3

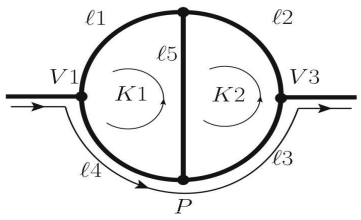


Figure 2.16: The canonical 2-point 2-loop graph from ϕ^3theory.

Using fig.(2.16) as the example, the following for the $\mathcal{U}$ and $\mathcal{F}$ functions are obtained:

$$\mathcal{U} = x_1x_5 + x_4x_5 + x_1x_2 + x_1x_3 + x_4x_2 + x_4x_3 + x_2x_5 + x_3x_5 \tag{2.98}$$

$$\begin{aligned} \mathcal{F} &= p^2 \left[x_1x_4x_5 + x_1x_4x_2 + x_1x_4x_3 + x_2x_3x_1 + x_2x_3x_4 + x_2x_3x_5 + x_1x_5x_3 + x_4x_5x_2\right] \\ &\quad + \mathcal{U}\left(x_1 + x_2 + x_3 + x_4 + x_5\right) m^2 \end{aligned} \tag{2.99}$$

and the choice for the primary sector is as follows

$$\begin{aligned} \mathbb{I} &= \theta(x_1 > x_2, x_3, x_4, x_5) + \theta(x_2 > x_1, x_3, x_4, x_5) + \theta(x_3 > x_1, x_2, x_4, x_5) \\ &\quad + \theta(x_4 > x_1, x_2, x_3, x_5) + \theta(x_5 > x_1, x_2, x_3, x_4) \end{aligned} \tag{2.100}$$

From the symmetry of the graph we conclude that sectors corresponding to x_1, x_2, x_3, x_4 are all identical so that only two choices for the primary sector remain

$$\mathbb{I} = 4\theta(x_3 > x_1, x_2, x_4, x_5) + \theta(x_5 > x_1, x_2, x_3, x_4) \tag{2.101}$$

In the example, the calculation on the primary sector $\theta(x_3 > x_1, x_2, x_4, x_5)$ will be done first. Our remapping then takes the form of a change of variables

$$t_1 = \frac{x_1}{x_3}; t_2 = \frac{x_2}{x_3}; t_4 = \frac{x_4}{x_3}; t_5 = \frac{x_5}{x_3} \text{ for } t_1, t_2, t_4, t_5 \in [0, 1]$$

The $\mathcal{U}$ and $\mathcal{F}$ functions then become with variable x_3 factored out

$$\mathcal{U} = x_3^2(t_1 + t_4 + t_5 + t_1t_{25} + +t_4t_{25} + t_2t_5) = x_3^2\mathcal{U}_3 \tag{2.102}$$

$$\begin{aligned} \mathcal{F} &= x_3^3 \left(p^2 \left[t_1t_4t_5 + t_1t_4t_2 + t_1t_4 + t_2t_1 + t_2t_4 + t_2t_5 + t_1t_5 + t_4t_5t_2\right]\right. \\ &\quad \left. +\mathcal{U}_{x_3}(1 + t_1 + t_2 + t_4 + t_5)m^2\right) \\ &= x_3^3\mathcal{F}_3 \end{aligned} \tag{2.103}$$

where the notation $x_{ijkl} = x_i + x_j + x_k + x_l$ is used. Substituting (2.102) and (2.103) into (2.59) and then integrating over x_3 which eliminates the δ-function, gives

$$I_{\mathcal{G}_{\theta(x_3>x_1,x_2,x_4,x_5)}} = 4\pi^\omega\Gamma(5 - 2\omega)\int_0^1 dt_1dt_2dt_4dt_5\frac{\mathcal{U}_3^{5-3\omega}}{\mathcal{F}_3^{5-2\omega}} \tag{2.104}$$

An important thing to note is that by choosing x_3 to be the variable with the largest value, x_2 has been made UV safe, and by safe it means it cannot be dangerous due to the symmetry of the topology.

The primary sector of $\theta(x_5 > x_1, x_2, x_3, x_4)$ will now be evaluated The change of variables used is

$$t_1 = \frac{x_1}{x_5}; t_2 = \frac{x_2}{x_5}; t_3 = \frac{x_3}{x_5}; t_4 = \frac{x_4}{x_5} \text{ for } t_1, t_2, t_3, t_4 \in [0, 1]$$

The $\mathcal{U}$ and $\mathcal{F}$ functions become (with x_5 factored out)

$$\mathcal{U} = x_5^2(t_1 + t_4 + t_2 + t_3 + t_1t_{23} + +t_4t_{23}) = x_5^2\mathcal{U}_5 \tag{2.105}$$

$$\begin{aligned} \mathcal{F} &= x_5^3 \left(p^2 \left[t_1t_4t_3 + t_2t_4 + t_2t_3 + t_1t_4t_{23} + t_2t_3t_{14}\right]\right. \\ &\quad \left. +\mathcal{U}_{x_5}(1 + t_1 + t_2 + t_4 + t_5)m^2\right) \\ &= x_5^3\mathcal{F}_5 \end{aligned} \tag{2.106}$$

Substiting (2.105) and (2.106) into (2.59) and then integrating over x_5 the following result is obtained

$$I_{\mathcal{G}_{\theta(x_5>x_1,x_2,x_3,x_4)}} = \pi^{\omega}\Gamma(5-2\omega)\int_0^1 dt_1 dt_2 dt_3 dt_4 \frac{\mathcal{U}_5^{5-3\omega}}{\mathcal{F}_5^{5-2\omega}} \tag{2.107}$$

2.10.2 Sector Decomposition: Iterative Sectors

Having dealt with the δ-function, next up is the iterative sectors. The iteration of sector decomposition is taken by taking the minimal set of parameters which cause $\mathcal{U}_i$ and $\mathcal{F}_i$ to vanish, this set in general is not unique and there has been resent progress in mapping this problem to Hironakas polyhedra game [91] to not only choose optimal sets but to avoid infinite recursions. Once a minimal set has been found the integrals are then remapped over the unit sphere to a set of $(N-1)$ $(N-2)$-dimensional integrals making sure that the choice of mapping preserves the limits of integration $[0,1]$. This iterative process terminates when $\mathcal{U}_{x_i>\ldots}$ and $\mathcal{F}_{x_i>\ldots}$ are of the form

$$\mathcal{U}_{x_i>\ldots} = 1 + H(x) \qquad \text{and} \qquad \mathcal{F}_{x_i>\ldots} = (S^2 + H(x))$$

where $H(x)$ is a polynomial in x and S^2 is the kinematic invariants which are assumed to be positive. At this stage there will no longer be any overlapping divergences and all the non-overlapping pole terms will be contained within the Feynman parameters of the form $x_j^{\mu+\varepsilon\nu}$ exterior to the $\mathcal{U}_{x_i>\ldots}$ and $\mathcal{F}_{x_i>\ldots}$ functions .

At this point subtractions are made by Taylor series which then allows expansion in ε up to the order of interest.

For the example the choices for the iterative sectors are shown in fig. 2.17.

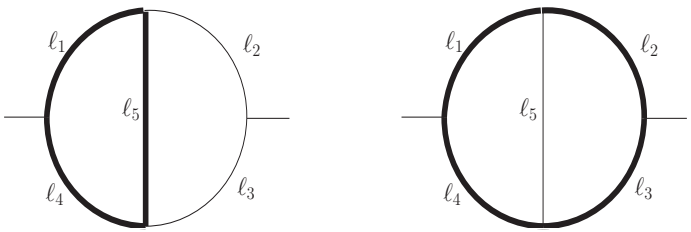


Figure 2.17: he iterative sector graphs for primary sectors $\theta(x_3 > x_1, x_2, x_4, x_5)$ and $\theta(x_5 > x_1, x_2, x_3, x_4)$ respectively are highlighted in bold.

From this, the primary sector $\theta(x_3 > x_1, x_2, x_4, x_5)$ there are two possible choices for the iterative sector

$$\theta(x_3 > x_1, x_2, x_4, x_5) = 2\theta(x_3 > t_1 > t_2, t_5) + \theta(x_3 > t_5 > t_1, t_2)$$

where again due to symmetry the iterative sectors belonging to ℓ_1 and ℓ_2 are the same.

For the primary sector $\theta(x_5 > x_1, x_2, x_3, x_4)$ there is only one choice of iterative sector again due to symmetry

$$\theta(x_3 > x_1, x_2, x_4, x_5) = 4\theta(x_5 > t_3 > t_1, t_2, t_4)$$

Taking the sector $\theta(x_3 > t_1 > t_2, t_5)$, and recalling (2.104) and with the change of variables

$$z_1 = t_1; z_2 = t_2; z_4 = \frac{t_4}{t_1}; z_5 = \frac{t_5}{t_1}; \qquad z_1, z_2, z_4, z_5 \in [0, 1]$$

Where $t_2 = z_2$ due to this variable, as previously mentioned, no longer being dangerous. Applying the mapping, $\mathcal{U}_3$ and $\mathcal{F}_3$ functions become

$$\mathcal{U} = z_1(1 + z_4 + z_5 + z_2 + z_1z_5 + z_4z_2 + +z_5z_2 + z_1z_4z_5) = z_1\mathcal{U}_{(3,1)} \tag{2.108}$$

$$\begin{aligned} \mathcal{F} &= z_1 \left(p^2 \left[z_1^2 z_4 z_5 + z_1 z_4 z_2 + z_1 z_4 + z_2 + z_2 z_4 + z_2 z_5 + z_1 z_5 + z_1 z_5 z_2\right]\right. \\ &\quad \left. + \mathcal{U}_{(3,1)}(1 + z_1 + z_1 z_2 + z_4 + z_1 z_5) m^2\right) \\ &= z_1 \mathcal{F}_{(3,1)} \end{aligned} \tag{2.109}$$

Substiting (2.108) and (2.109) into (2.104) and taking care to change the measure of the integral, yields

$$I_{\mathcal{G}_{\theta(x_3>z_1>z_2,z_5)}} = 8\pi^\omega \Gamma(5 - 2\omega) \int_0^1 dz_1 dz_2 dz_4 dz_5 z_1^2 \frac{z_1^{5-3\omega}}{z_1^5 - 2\omega} \frac{\mathcal{U}_{(x_3;z_1)}^{5-3\omega}}{\mathcal{F}_{(3,1)}^{5-2\omega}} \tag{2.110}$$

At this point the iteration stops as the $\mathcal{U}_{(3,1)}$ is now of the form $\mathcal{U}_{x_i>\ldots} = 1 + H(x)$ and thus safe, the integral is now free of overlapping divergencies and may now have its singular structure evaluated. Substituting in the dimension of the theory $\omega = 3 - \varepsilon$, the Γ function then becomes

$$\begin{aligned} \Gamma(-1 + 2\varepsilon) &= \frac{(-1 + 2\varepsilon)}{(-1 + 2\varepsilon)} \Gamma(-1 + 2\varepsilon) = \frac{\Gamma(2\varepsilon)}{(-1 + 2\varepsilon)} \\ &= \frac{2\varepsilon}{2\varepsilon} \frac{\Gamma(2\varepsilon)}{(-1 + 2\varepsilon)} = -\frac{\Gamma(1 + 2\varepsilon)}{(1 - 2\varepsilon)2\varepsilon} \end{aligned} \tag{2.111}$$

Using this result (2.110) becomes

$$\begin{aligned} I_{\mathcal{G}_{\theta(x_3>z_1>z_2,z_5)}} &= -4\pi^{\omega}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\int_0^1 dz_1dz_2dz_4dz_5 z_1^{-1+\varepsilon}\frac{\mathcal{U}_{(3,1)}^{-4+3\varepsilon}}{\mathcal{F}_{(3,1)}^{-1+2\varepsilon}} \\ &= -4\pi^{\omega}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\int_0^1 dz_1dz_2dz_4dz_5 z_1^{-1+\varepsilon}\left[B_{(3,1)}(z_1,z_2,z_4,z_5)\right. \\ &\quad \left.-[B_{3,1)}(0,z_2,z_4,z_5)+B_{(3,1)}(0,z_2,z_4,z_5)\right] \\ &= -4\pi^{\omega}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\left(\int_0^1 dz_1\frac{1}{z_1^{1-\varepsilon}}\int_0^1 dz_2dz_4dz_5 B_{(3,1)}(0,z_2,z_4,z_5)\right. \\ &\quad \left.+\int_0^1 dz_1dz_2dz_4dz_5\frac{B_{(3,1)}(z_1,z_2,z_4,z_5)-B_{(3,1)}(0,z_2,z_4,z_5)}{z_1^{1-\varepsilon}}\right) \\ &= -4\pi^{\omega}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\left(\frac{1}{-\varepsilon}\int_0^1 dz_2dz_4dz_5 B_{(3,1)}(0,z_2,z_4,z_5)\right. \\ &\quad \left.+\int_0^1 dz_1dz_2dz_4dz_5\frac{B_{(3,1)}(z_1,z_2,z_4,z_5)-B_{(3,1)}(0,z_2,z_4,z_5)}{z_1^{1-\varepsilon}}\right) \end{aligned} \tag{2.112}$$

Where $B_{(3,1)}(z_1,z_2,z_4,z_5)=\frac{\mathcal{U}_{(x3,1)}^{-4+3\varepsilon}}{\mathcal{F}_{(3,1)}^{-1+2\varepsilon}}$ and the result $\int_0^1\frac{1}{-\varepsilon}\frac{d}{dz_1}z_1^{-\varepsilon}=-\frac{1}{\varepsilon}\left(1^{-\varepsilon}-0^{-\varepsilon}\right)=\frac{1}{-\varepsilon}$ which uses $\lim_{u\to 0}u^{-\varepsilon}=\lim_{u\to 0}e^{-\varepsilon lnu}$ has been used.

Turning to the iterative sector $\theta(x_3>t_5>t_1,t_2,t_4)$, the mapping is

$$z_1=\frac{t_1}{t_5};z_2=t_2;z_4=\frac{t_4}{t_5};z_5=t_5; \qquad z_1,z_2,z_4,z_5\in[0,1]$$

The map $\mathcal{U}_3$ and $\mathcal{F}_3$ to the new sector and the functions become

$$\mathcal{U}=z_5(1+z_1+z_4+z_2+z_1z_{25}+z_4z_{25})=z_5\mathcal{U}_{(3,5)} \tag{2.113}$$

$$\begin{aligned} \mathcal{F} &= z_5\left(p^2\left[z_1z_4z_5^2+z_1z_4z_2z_5+z_1z_4z_5+z_2z_{14}+z_2+z_1z_5+z_1z_5z_2\right]\right. \\ &\quad \left.+\mathcal{U}_{(3,5)}(1+z_1z_5+z_2+z_3z_4+z_5)m^2\right) \\ &= z_3\mathcal{F}_{(3,5)} \end{aligned} \tag{2.114}$$

Substituting (2.113) and (2.114) into (2.104) to obtain

$$I_{\mathcal{G}_{\theta(x_3>z_5>z_1,z_2)}}=4\pi^{\omega}\Gamma(5-2\omega)\int_0^1 dz_1dz_2dz_4dz_5z_3^2\frac{z_3^{5-3\omega}}{z_3^5-2\omega}\frac{\mathcal{U}_{(3,5)}^{5-3\omega}}{\mathcal{F}_{(3,5)}^{5-2\omega}} \tag{2.115}$$

which by analogy with (2.110) to get

$$\begin{aligned} I_{\mathcal{G}_{\theta(x_3>z_5>z_1,z_2)}} &= -2\pi^{\omega}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\left(\frac{1}{-\varepsilon}\int_0^1 dz_1dz_4B_{(3,5)}(z_1,z_2,z_4,0)\right. \\ &\left. + \int_0^1 dz_1dz_2dz_4dz_5\frac{B_{(3,5)}(z_1,z_2,z_4,z_5)-B_{(3,5)}(z_1,z_2,z_4,0)}{z_1^{1-\varepsilon}}\right) \end{aligned} \tag{2.116}$$

Moving to the final iterative sector $\theta(x_5 > t_3 > t_1, t_2, t_4)$, which has a mapping of

$$z_1 = \frac{t_1}{t_3}; z_2 = \frac{t_2}{t_3}; z_3 = t_3; z_4 = \frac{t_3}{t_5}; \qquad z_1, z_2, z_3, z_4 \in [0,1]$$

Mapping $\mathcal{U}_5$ and $\mathcal{F}_5$ to the new sector to get

$$\mathcal{U} = z_3(1 + z_1 + z_4 + z_2 + z_{14}z_3 + z_{14}z_2z_3) = z_3\mathcal{U}_{(5,3)} \tag{2.117}$$

$$\begin{aligned} \mathcal{F} &= z_3\left(p^2\left[z_1z_4z_3 + z_1z_3 + z_4z_2z_3 + z_2z_3 + z_1z_2z_4z_3^2 + z_1z_4z_3^2 + z_2z_3^2z_1 + z_2z_3^2z_4\right]\right. \\ &\left. + \mathcal{U}_{(5,3)}(1 + z_1z_3 + z_2z_3 + z_3 + z_4z_3)m^2\right) \\ &= z_3\mathcal{F}_{(5,3)} \end{aligned} \tag{2.118}$$

Substituting (2.117) and (2.118) into (2.107) to obtain

$$I_{\mathcal{G}_{\theta(x_5>z_3>z_1,z_2,z_4)}} = 4\pi^{\omega}\Gamma(5-2\omega)\int_0^1 dz_1dz_2dz_3dz_4z_3^2\frac{z_3^{5-3\omega}}{z_3^5-2\omega}\frac{\mathcal{U}_{(5,3)}^{5-3\omega}}{\mathcal{F}_{(5,3)}^{5-2\omega}} \tag{2.119}$$

By analogy with (2.110) results in the following

$$\begin{aligned} I_{\mathcal{G}_{\theta(x_5>z_3>z_1,z_2,z_4)}} &= -\frac{2}{\Gamma(5)}\frac{\Gamma(1+2\varepsilon)}{(1-2\varepsilon)\varepsilon}\left(\frac{1}{-\varepsilon}\int_0^1 dz_1dz_2dz_4B_{(x_3;z_5)}(z_1,z_2,z_4,0)\right. \\ &\left. + \int_0^1 dz_1dz_2dz_4dz_5\frac{B_{(x_3;z_5)}(z_1,z_2,z_4,z_5)-B_{(x_3;z_5)}(z_1,z_2,z_4,0)}{z_3^{1-\varepsilon}}\right) \end{aligned} \tag{2.120}$$

At this point the singular structure of fig.(2.16) has been removed and no overlapping divergence has been encountered and is now ready to be numerically integrated. As such Sector Decomposition is a powerful method for dealing with the problematic divergences but, as can be seen, the number of integrals which have to be calculated is increased dramatically.

2.11 Sectored Feynman parameter space - momentum space identity

From the exploration of the henge decomposition representation of BPHZ and the sector decomposition method it should be clear that there are some remarkable similarities between the two techniques, most strikingly when viewed graphically. This similarity is the main motivational point to see how close the relation between these two methods truly is, that is the relation between complete ordering of the Feynman parameters with an ordering the henge decomposed momentum space integral.

What this shows is that when a complete ordering of the Feynman parameters is done, there is a corresponding ordering of momentum space propagators such that there is always a smallest momentum, and each subesquent term is larger than the previous, otherwise known as complete henge decomposition. This shows that there is one-to-one mapping between individual sectors and henge decompositions, which means that one moving from one representation to another, the manifestly disentangled nature of the integrals is not spoiled.

Looking at an integral restricted to the region of parameter space in which the parameters are totally ordered, that is $i < j \Rightarrow x_i > x_j$, gives

$$S_N \equiv \Gamma(N) \int_0^1 dx_1 \cdots \int_0^1 dx_N \, \delta\left(1 - \sum_{k=1}^{N} x_k\right) \theta\Big(1 > x_1 > \cdots > x_N > 0\Big) \left[\sum_{j=1}^{N} x_j Q_j\right]^{-N},$$

where the notation $\theta\left(\alpha_1 > \cdots > \alpha_N\right) \equiv \prod_{j=1}^{N-1} \theta\left(\alpha_j - \alpha_{j+1}\right)$ is used.

It is convenient to revert to the form of the Feynman parameter (Schwinger representation) identity of eq. (2.27) with $\alpha_j = 1 \; \forall j$, for which our sector integral becomes

$$S_N = \Gamma(N) \int_0^\infty dt_1 \cdots \int_0^\infty dt_N \, \delta(1 - t_1) \, \theta\Big(t_1 > \cdots > t_{N-1} > t_N > 0\Big) \left[\sum_{j=1}^{N} t_j Q_j\right]^{-N}.$$

Making a further shift in the variables $z_i \equiv t_i - t_{i+1}$ for $1 \leq i \leq N$, where $t_{N+1} \equiv 0$ is fixed for convenience. This transformation has unit Jacobian and inverse $t_i = \sum_{k=i}^{N} z_k$, hence

$$S_N = \Gamma(N) \int_0^1 dz_1 \cdots \int_0^1 dz_N \, \delta\left(1 - \sum_{k=1}^{N} z_k\right) \left[\sum_{j=1}^{N} \sum_{k=j}^{N} z_k Q_j\right]^{-N},$$

where the ordering of the t_i has been translated into the limits on the integrals over the z_i.

Noting that

$$\sum_{j=1}^{N}\sum_{k=j}^{N} z_k Q_j = \sum_{j=1}^{N}\sum_{k=1}^{N} z_k Q_j \theta(k \geq j) = \sum_{k=1}^{N}\sum_{j=1}^{N} z_k Q_j \theta(j \leq k) = \sum_{k=1}^{N}\sum_{j=1}^{k} z_k Q_j,$$

yields

$$S_N = \Gamma(N) \int_0^1 dz_1 \cdots \int_0^1 dz_N \, \delta\left(1 - \sum_{k=1}^{N} z_k\right) \left[\sum_{k=1}^{N} z_k \sum_{j=1}^{k} Q_j\right]^{-N} = \prod_{\ell=1}^{N} \frac{1}{\sum_{j=1}^{k} Q_j}$$

upon using eq. (2.29).

This result is important as it shows that upon a complete ordering of the Feynman parameters z, the quadratic form denominator Q_t corresponding to the largest Feynman parameter z_t is guaranteed to be smaller than all the other denominators. Which corresponds exactly to the line carrying the smallest momentum in the Henge decomposition method.

If the $Q_i \geq 0 \ \forall i$, for instance they are Euclidean quadratic forms such as $Q_i \equiv p_i^2 + m^2$, then the sector integrals are of the form

$$\begin{aligned}
S_1 &= \int_0^1 dx_1 \frac{\delta(1-x_1)}{x_1 Q_1} = \frac{1}{Q_1}, \\
S_2 &= \int_0^1 dx_1 \int_0^1 dx_2 \frac{\delta(1-x_1)\theta(x_1 > x_2)}{[x_1 Q_1 + x_2 Q_2]^2} = \frac{1}{Q_1(Q_1+Q_2)} \leq \frac{1}{Q_1^2}, \\
S_2 &= \int_0^1 dx_1 \cdots \int_0^1 dx_3 \frac{\delta(1-x_1)\theta(x_1 > x_2 > x_3)}{[x_1 Q_1 + x_2 Q_2 + x_3 Q_3]^3} = \frac{1}{Q_1(Q_1+Q_2)(Q_1+Q_2+Q_3)} \leq \frac{1}{Q_1^3}, \\
S_3 &= \int_0^1 dx_1 \cdots \int_0^1 dx_4 \frac{\delta(1-x_1)\theta(x_1 > x_2 > x_3 > x_4)}{[x_1 Q_1 + x_2 Q_2 + x_3 Q_3 + x_4 Q_4]^4} \\
&= \frac{1}{Q_1(Q_1+Q_2)(Q_1+Q_2+Q_3)(Q_1+Q_2+Q_3+Q_4)} \leq \frac{1}{Q_1^4},
\end{aligned} \tag{2.121}$$

where the final inequality automatically holds in the sector under consideration.

Chapter 3

BPHZ Feynman Diagram Evaluation

In this chapter, the computer code is decribed with the emphasis on a detailed description of the main graphical algorithm and an explanation of the data structures of the code. The choice of Maple [92] as the language of the code for implementation of the BPHZ algorithm was due to its symbolic manipulation. Over time the parts of the code that required symbolic manipulation have been replaced and it would now be possible to write this code in any non-symbolic language. The underlying code was written by A. D. Kennedy, with the of this thesis being a complete rewrite of how the numerator tensor data structures work to allow non-scalar Feynman graphs to be computed as well as the handling of spinor structures and all necessary work around this. The symbolic elimination for the Taylor series was replaced with method based upon the graphical Galler-Fisher algorithms used extensively throughout this code as well as corrections in momentum space integration routines.

3.1 Overview

This computer program is the culmination of the work done in this thesis. The problem of UV divergences within quantum field theory can be completely solved using the ideas discussed in the previous chapter and it is with this code that those ideas are implemented to fully automate the removal of UV divergences from Feynman graph calculations. The remaining finite part of the integral will be dealt with with some suitable numerical integrator. It takes a diagram which can either be put into the code by hand or taken from the output file of some Feynman Graph generator such as the program QGRAF [93]. This information gets passed to the **Static Diagram Analysis** which examines the topology of the graph by analysing the connectivity; checks to see if the graph is 1PI and routes the momenta through the graph. This information is

then stored in external pseudofunctions as the information will not be changed further. Once the external pseudofunctions are created, the information gets passed on to the **Forestry** which determines whether the graph is divergent via powercounting and generates the correct Taylor series subtractions as necessary. **Forestry** then applies the R-Operation to the graph functions. The last part of the code is the **Parametric Representation** which takes the graph functions: applies the Feynman parameterisation; Manipulates the tensor structure into a more convenient form; performs the momentum-space integration and finally allows the user to output the parametric integrand symbolically in Maple, or generate the output code into a suitable language(in this case C) for the finite part of the integral. The output information is then evaluated using the VEGAS Monte-Carlo algorithm.

This code is written in a recursive functional way due to the recursive nature of the R and $\bar{R}$ algorithms as well as the overall elegance this method has when applied in Maple. As such, there is an extensive use of procedures and related operations in the code. The main structures used will be explained with the specific meanings of a few important Maple commands relegated to an appendix.

3.2 Definitions & Declarations

The first section of the code sets up the naming conventions and data structures for representing a Feynman Graph. The information about a diagram can either be static or dynamic. The static information for a given graph, is the information which is not changed once it has been found, this includes the topology of the graph, the types of edges in the graph, the external and loop momenta to the graph and the set of vertices for the graph. The dynamic information is the all the information which changes as the code progresses, such as the momenta associated with each edge which can change under derivatives etc. To input a Feynman diagram we use a pseudofunction diagram made up of line, externalline , and vertex pseudofunctions; internally the diagram is represented as a graph made up of edges which carry the dynamic state of the graph, and a collection of tables indexed by the line name holding the static information.

3.2.1 Table indexing procedures

Tables are used extensively throughout the program, the method used for storing and retrieving data tables is known as “Hashing” and is briefly explained in the following subsection. The Hash-tables used will then be defined and an example of how one is set up in Maple is given.

Hashing

If the function $f(K)$ is considered; which contains the location of the argument K along with its associated data in the table, then there are only limited number of such functions $f(K)$ which would give a unique location for argument K. If, however, the idea of $f(K)$ having to map to a unique address in the table is abandoned then the benefit of an average search time of $O(1)$ is kept but without have to have K unique function evaluations for K addresses in the table, this is the idea behind "Hashing". The hash function $f : K \to \mathrm{Z}/(n)$ where K is a set and n is the size of your hash table. Any such function f will suffice, although it works best if it is very discontinuous, so "nearby" elements of K (such as people with the same surname in a telephone directory) are mapped to "distant" elements of $\mathrm{Z}/(n)$. This does not solve the problem completely but it helps to break up clusters of data. A collision occurs if $f(k_1) = f(k_2)$, but if the table is kept about half full and f is uniformly random then the probability of having a collision is about $1-(1-1/n)^N = 1-\exp[N\ln(1-1/n)] \approx 1-\exp(-N/n) \approx N/n \approx 1/2$. The remaining collisions that occur can be solved via a process called chaining. When collision occurs a sequential search is then implemented to resolve the different data. This sequential is fast as the lists will be very small. For more information on these ideas, the reader is advised to read the appropriate sections in "*The Art of Computer Programming I*" by D.E. Knuth [94].

Hash-Tables

In this code there are four kinds of index functions used:

- **index/err**: In which if an unassigned element is called it returns an error;
- **index/null**: All unassigned elements implicitly contain the empty sequence NULL;
- **index/sparse2**: The unassigned elements of this table contain the sequence $[0, 0]$, this for when a table of sequences is needed;
- **index/edge**: This table is used to ensure that only edges pseudofunctions exist in the table.

All the hash tables are set up in the same way, for example in the case of **index/sparse2** :

```
‘index/sparse2‘ := proc(idx::list, tbl::table, entry::list)
  if nargs = 2 then
```

```
    if assigned(tbl[op(idx)]) then tbl[op(idx)]
    else 0$2
    fi
  else tbl[op(idx)] := op(entry);
  fi
end proc:
```

The structure of this is simple, the arguments of the procedure are a list of indices, or keys, the hash table and a list of expressions which are the entries for the table. If arguments passed to this procedure are only two, the index list and hash table, it checks to make to see if an entry exists for the key provided by index list. If an entry exists, it returns the value of the entry, if no entry exists it returns the value $(0, 0)$ (or an error or NULL etc. for the other index functions). If three arguments are passed to the procedure, a new entry, provided by the entry list, is placed into the table indexed by the index list.

3.2.2 Names

The labeling of lines, vertices, momenta and Feynman parameters are always of the form of a letter followed by a number, as is the normal convention. Each type of line, vertex, momenta and Feynman parameter is designated by a specific letter followed by a unique number to identify each one individually as is usual in most Feynman graph calculations. These lines are converted from strings into symbols so that they can be used as variables. The convention of this code is that internal lines are always names such as **I123**, external lines **E123**, vertices **V123**, external momenta **p123**, loop momenta **k123**, and any other momenta **q123**. Additionally, temporary line names of the type **QE123** and **QI123** are used to distinguish line names involved in subtraction operations and finally the Feynman parameters as **x123**.

This naming scheme is enforced by the procedure **alphaNum** which takes in a line name, vertex name, momentum name or Feynman parameter name and checks that the letter of this object is in alphaset, that is the set of letters designated for each type of object name such as "I" and "E" for a line name, and that the remainder of the name only consists of digits.

Additionally, a procedure **alphaLine** is used to allow for the construction of the temporary line names used in the taylor subtraction scheme routines and differs only in that instead of checking that numbers following a line name are digits, converts the string of digits into a single symbol.

A distinction is made in this code between lines and edges. A line is a component of the topology of the input graph and is assigned a line name, whereas an edge is function of the line name which contains all the necessary information associated with

the line such as numerator and denominator structures.

Due to the use of Henges in the code, it is necessary to introduce an ordering on the set of lines in order to avoid overcounting the number of henges, as every henge H occurs for every ordering of the momenta in G/H.. This is chosen to be only dependent on the integer address that points to line expressions, effectively making it arbitrary. One condition is applied, the line **I0** is always guaranteed to contain the smallest momentum.

From this two functions are constructed which will return the smallest or the largest linename from a nonempty list of linenames, **linenameMin** and **linenameMax**. These are found by comparing any two entries in the list of line names and then comparing the bigger/smaller of the two with another entry and so on, until the biggest/smallest entry of the list is found.

3.2.3 Feynman Rules

The Feynman rules for the theory to which the diagram belongs are defined by the linetypes and vertextypes allowed in the graph. These at present are defined by the user, but a future iteration of the code could get these directly from the Lagrangian of the theory. The properties of the propagators and vertices of a given type are held in a collection of arrays. The masses of various linetypes are stored in an array *mass* indexed by the linetype and initialised to *index*/*err*:

```
mass[scalar] := 'm':
```

An array indexed by linetype is also set-up which relates linetypes with corresponding fields they can represent:

```
lineFieldTypes[scalar] := scalarField, scalarField:
```

As the scalar field itself is self-adjoint, but for fermion fields the entries would correspond to field and anti-field.

And finally an array indexed by vertextype and fieldtype is created which is assigned to a specific number which represents the "slot" into which the line is allowed to connect to the vertex:

```
vertexLineTypes[ThreeScalarVertex,scalarField] := 1,2,3:
```

Which for non-scalar theories such as QED each slot in a vertex would correspond uniquely to a photon, electron or positron line.

The slots of the vertices are explained in detail later in section 3.5.2.

The naïve power counting degree of the propagator is needed for use in finding the overall degree of divergence of the graph. This degree of divergence is constructed by storing the power counting degree for the denominator (*denomDegree*). These degrees require that the propagator $\Delta(k) = \frac{1}{\mathrm{D}(k)}$ satisfies the bounds $|D(k)| > c\max(k,m)^{-\mu}$ where $-\mu$ is the degree bound of the denominator. The numerator degree of divergence is obtained from the **numeratorMomenta** dynamic data structure.

```
denomDegree[scalar] := -2:
```

This is all the information required by the code to apply particular Feynman rules of a theory to the underlying code.

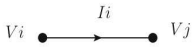
3.2.4 Data Structures of the code

The data structures of the code now need to be defined. The first thing that needs to be mentioned is the momentum associated with each edge in the graph. For the purposes of the code, the **momentum** flowing through a line or entering through an external line must be a sum of external (**p**) and loop (**k**) momentum names with coefficients in Z/(3) (i.e., 0 or ± 1).

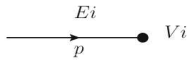
External Pseudofunctions

The external pseudofunctions contain the static data structures for the construction of the graph topology. The graph topology is constructed from internal **line**s, **externalline**s and **vertex**es and is currently the only input needed for the calculation of a particular graph by the code. To avoid confusion, the input topological structure is referred to as a *diagram* and the constructed data structure corresponding to this diagram is called a *graph*. The *diagram* can by type checked to ensure that the only information contained within the diagram are the set of internal lines, external lines and vertices.

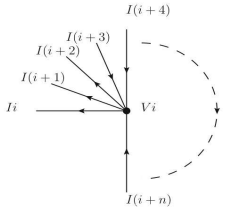
The internal lines are represented by a line pseudofunction which consists of a linename, the initial and final vertexnames, and the linetype and represented graphically by (for the scalar linetype):



External lines are represented by externalline pseudofunctions which consists of a linename, the vertexname to which it is attached, the momentum entering the graph, and the linetype. Momentum conservation is required and is checked in the analyze diagram section. Which is represented graphically by:



The vertices of the graph are represented by the vertex pseudofunctions which consists of a vertexname and a vertextype. which graphically is:



Examples

As stated above, the input for the program is the topology of the graph which is to be calculate. This *diagram* consists of a set internal lines, external lines and the vertices and theses diagrams can be easily generated as output from programs such as QGRAF [93] as well as being constructed by hand.

The following 2-pt 2-loop scalar ϕ^3 diagram, figure 3.1:

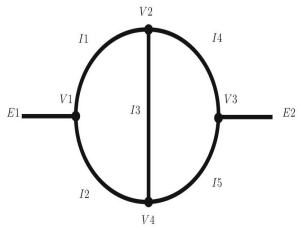


Figure 3.1: 2 loop 2 point scalar ϕ^3 diagram

is constructed with static data structures as follows:

```
diag2 := diagram(
  seq(ThreeScalarVertex(cat('V',i),
      subs(diag2vertices, cat('V',i))), i = 1..4),
  scalar(E1,V1,p1),
  scalar(E2,V3,-p1),
  scalar(I1,V1,V2),
  scalar(I2,V4,V1),
  scalar(I3,V2,V4),
  scalar(I4,V2,V3),
  scalar(I5,V3,V4));
```

Using this input along with the static information of the Feynman rules, the calculation is then performed.

3.2.5 Internal Pseudofunctions

A graph is the internal data structure representing a subgraph of a Feynman diagram (which as per usual can be the graph itself): the dynamic data associated with it is represented explicitly while the static data is accessed through global arrays indexed by the linenames. A graph pseudofunction describes a subgraph within a diagram: it consists of a *numerical coefficient*, a *tensor structure*, an *edge table* containing all the edges in the graph, a list of *external line names* (lines that are not in the subgraph but are attached to vertices that are in it) and the list of *internal line names* of lines belonging to the subgraph. The structure also stores the diagram's *hash code* so one can check whether it is being used in the correct context, and the value *maximin* (named after a Roman emperor) which tracks henges visited so far.

```
‘type/graph‘ := proc (a)
  type(a, specfunc(anything, graph))
    and nops(a) = 8
    and type(op(1,a), anything) \# coefficient
    and type(op(2,a), KTensor)  \# tensor structure
    and type(op(3,a), table) \# all edges in graph
    and type(op(4,a), list(linename)) \# external lines of subgraph
    and type(op(5,a), list(linename)) \# internal lines of subgraph
    and type(op(6,a), string)   \# diagHash
    and type(op(7,a), linename) \# maximin
    and type(op(8,a), anything) \# debug list of nested Taylor series subtractions
end proc:
```

The tensor structure is represented as a tensor pseudofunction consisting of a sum of products of Kronecker delta tensors. Spinor structures are temporarily stored inside the **KTensor** until the trace is complete.

The spinor structures are held in a special pseudofunction in order to avoid confusion with the Kronecker delta's.

The Kronecker deltas themselves are stored in a nested set of pseudofunctions, with the top level **KTensor** consisting only of sums of Kronecker deltas, which can only contain products of Kronecker deltas.

The products of the Kronecker deltas can can also contain sums of Kronecker deltas, this method is employed to mesh well with the recursive methods used on the graph

topologies. This nesting is later 'flattened', so the result is only a sum of products of Kronecker delta functions:

The definition of a Kronecker delta tensor is somewhat abused in this code. It can take the form of an actual Kronecker delta, indexed by a pair of linenames with a positive integer associated to it, which is the numerator momenta vector index, $\delta_{I1_1,I2_1}$, which always ties together internal edges. Or it can take the form of an external momenta with a free index with the index being represented by linename and positive integer from the numerator momenta, p_{E1_2}. By convention external momenta tensors are always indexed with external line names.

Using the clifford algebra anticommutation relation:

$$\{\gamma_\mu, \gamma_\nu\} = \gamma_\mu\gamma_\nu + \gamma_\nu\gamma_\mu = 2\delta_{\mu\nu}\mathbf{1}, \tag{3.1}$$

spinors can be represented by a Kronecker delta connecting two dirac γ matrices. As the code requires all internal γ's to be traced over, the formal use of the γ matrices is eschewed in favour of a Kronecker delta function indexed by a pair of linename and numerator index akin to that of Kronecker delta function used above for the numerator momenta, as the actual structure of the γ matrices is never needed. External γ's could be calculated in a future version of the code using projection operators.

The need for this special delta function is due to the deltas' acting upon different spaces, and it is left to a following module in the code to perform the traces which will reduce the spinors to momentum-space delta functions.

Each *edge* consists of the momentum $\mathbf{q}$ associated with the line, the power of the quadratic form occuring in the denominator of the propagator associated with the line, and a list $n_1 \ldots n_k$ corresponding to the momentum vectors in the numerator. These numerator vectors may be written explicitly as $q_{l,n_1} \ldots q_{l,n_j}$ where l is the linename and j is the numerator degree: in other words the tensor indices used are built from the linename and an index selecting one of the j numerator vectors. A special kind of edge, denoted as *sEdge* (subtracted edge), are temporarily created in the Taylor subtraction scheme routines to allow the code to distinguish between edges, as functions of the output variables, and sEdges, as functions of the temporary new variables used in the elimination of momenta resulting from Taylor subtracting various henges of the initial diagram.

```
`type/edge` := proc (a)
  type(a, specfunc(anything, 'edge')) or type(a, specfunc(anything, 'sEdge'))
```

```
    and nops(a) = 3
    and type(op(1,a), momentum)   \# line's momentum
    and type(op(2,a), negint)     \# degree of denominator
    and type(op(3,a), NumeratorMomenta) \# numerator momenta
end proc:
```

It should be noted that the edge pseudofunction does not include the denominator mass associated with propagators as this is a static quantity, and is not modified by the R-operation (just as the graph topology is unchanged). The correct masses are inserted when the Feynman parameterization procedure is invoked and the *Feynman Parameter Graph* is created. In other words the mass associated with a propagator i

The **numeratorMomenta** is a list of integers, the first giving the maximum index occuring and the rest corresponding to the tensor indices of the numerator. It should be noted that the number of indices present may be less than the maximum index as they may have been removed by differentiation during Taylor subtraction. In essence, the maximum numerator index keeps track of the maximum index used on the edge and the remaining list of indices are the indices of the numerator momenta on the edge. When new numerator momenta are added, the maximum index is increased by 1 for each new numerator momenta and then the index is added to the back of the list of numerator indices ($op(2..-1)$).

```
`type/NumeratorMomenta` := proc (a)
  type(a, specfunc(nonnegint, 'NumeratorMomenta'))
    and nops(a) >= 1
    and type(op(1,a), nonnegint)  \# maximum index used
end proc:
```

The convention used is to add each new index to the back of the sequence of numerator indices.

3.2.6 Feynman parameter pseudofunctions

In constructing the Feynman parameter representation of a graph we need to expand the numerator momenta and associate them with the appropriate external or loop momenta; we do this by building an FPGraph pseudofunction.

```
`type/FPGraph` := proc (a)
  type(a, specfunc(anything, FPGraph))
    and nops(a) = 3
    and type(op(1,a), anything) \# numerical coefficient
```

```
    and type(op(2,a), KTensor)  \# tensor structure
    and type(op(3,a), list(nonnegint))  \# powers of loop momenta in numerator
end proc:
```

3.2.7 Output routines

The procedures present in this part of code just extract the necessary information from the data structures from the code and presents them in a more readable fashion (i.e. it removes a lot of the packaging from the data structures). Such as when displaying a scalar edge it takes the momenta from the edge and squares it and adds in the appropriate mass-squared term. This process is mainly for debugging as using the print procedures is computationally expensive. As with the userinfo statements, the output routines will not be shown.

3.3 Galler-Fischer Algorithm

The Galler-Fischer algorithm and variants thereof, form an essential part of the graph manipulation aspect of the Feynman Integral automization code and as such a thorough examination of the algorithm is desirable. The Galler-Fischer algorithm [95] finds the connected components of an arbitrary graph by the equivalence classes of the graph which are determined by the pairwise equivalence relations. Two sets of objects are considered equivalent if they obey the following rules:

- Transitivity - if $a \equiv b$ and $b \equiv c$ then $a \equiv c$;
- Symmetry - if $a \equiv b$ then $b \equiv a$;
- Reflexivity - $a \equiv a$.

As has been seen in the previous section, a graph can be represented by a set of edges and each edge can be represented as pairs of vertices $\langle ij \rangle$, which are labelled by V concatenated with some small number. A sparse array $Father$ indexed by vertices is created and is updated as the ancestors (a vertex from which the edge originates) is found for each vertex within a given graph. The property of Transitivity will mean that for each set of connected components there will be a common ancestor, the ancestor $A(i)$ is i if $Father[i] = 0$ or $A(Father[i])$ otherwise. If the ancestors are unequal $A(i) \neq A(j)$ then we set $F[A(i)] \leftarrow A(j)$. Once the set of edges corresponding to the graph has been run through, the ancestor of any given vertex will label all the connected components that vertex is attached to.

The Galler-Fischer algorithm is then extended to find various other useful graphical features such as finding the irreducible edges within a connected graph, i.e. the edges which cannot be cut and still have the graph connected. This is done by introducing a

$\mathbf{Z_3}$ chain-valued (sum of edges with the coefficient ± 1 or 0) array $FatherPath$ indexed by vertices that contains the edges connecting i to $Father[i]$, and defining the route $R(i)$ from i to $A(i)$ to be zero if $i = A(i)$ or the chain $R(i) = FatherPath[i] \oplus R(Father[i])$ otherwise. Upon examining the edge $\langle ij \rangle$ it is noted that the chain $C_{ij} \equiv R(j) \oplus \langle ij \rangle \ominus R(i)$ connects $A(i)$ to $A(j)$, so if $A(i) \neq A(j)$ as well as updating $Father[i]$ as above $FatherPath[i] \leftarrow C_{ij}$ is also set. If on the other hand, $A(i) = A(j)$ then a closed loop C_{ij} has been discovered, and all edges occurring in this loop are marked as irreducible.

Routing Momenta is done by multiplying each loop by a symbol (canonically K) for the corresponding loop momentum, and for each vertex i at which the external momentum p enters the graph the route $R(i)$, which contains all the lines in a given route through the graph, is multiplied by a momentum p. The momentum flowing through any given edge is then the sum of all momentum coefficients of the given edge.

Henges can be found by starting with a 1PI graph $\mathcal{G}$ and removing an edge ℓ, which produces the set of edges $\mathcal{G} - \{\ell\}$. The Apply the Galler-Fischer algorithm again on this new graph and then partitioning result the lines into two sets $\mathcal{G} = I \cup R$ of irreducible lines and reducible lines. Applying the unextended Galler-Fischer algorithm once more to the partition I into disconnected pieces $I = \{\theta\}$. The resulting set is the required henge $\mathcal{H}(\mathcal{G}, \ell)$ and the single loop $\mathcal{G}/\mathcal{H}(\mathcal{G}, \ell) = R \cup \{\ell\}$.

The Galler-Fischer algorithm (including loops) can be expressed in the following way:

Algorithm 1 Galler-Fischer Algorithm

```
for edge in graph do
  sRoot, sPath ← ancestor [edge.start]
  eRoot, ePath ← ancestor [edge.end]
  r ← −sPath + edge + ePath
  if sRoot = eRoot then
    add r to loops
  else
    ancestor [sRoot] ← eRoot, r
  end if
end for
```

The ancestorRoute function is constructed as follows:

where the **fatherPath** is initialised with $index/sparse2$. The variables a and r are the ancestor, which is the vertex labelling the connected component of the diagram defined by the edges that have been inspected so far, and r is the $Z/(3)$ module of edges connecting the argument vertex to its ancestor a respectively. It also should be noted that to obtain the **ancestor** function, the first argument of **ancestorRoute** is

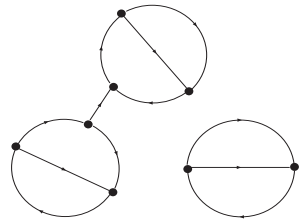
Algorithm 2 ancestorRoute pseudofunction

```
ancestorRoute := proc(v)local a, p, r;
a, p := fatherPath[v];
if a = 0 then
  v, 0
else
  a, r := procname(a);
  fatherPath[v] := a, r + p # optimization
end if
end proc;
```

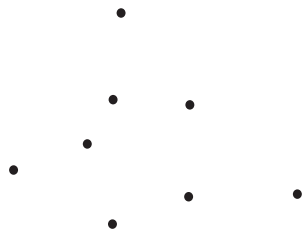
taken, or in the parlance of Maple:

$$\mathbf{ancestor} := \mathbf{v} \rightarrow \mathbf{op}(\mathbf{1}, [\mathbf{ancestorRoute}(\mathbf{v})]);$$

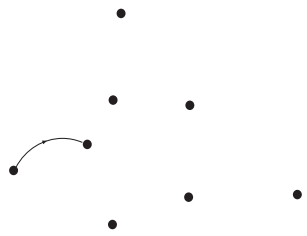
To help elucidate how this algorithm works, the following graph will be analysed:



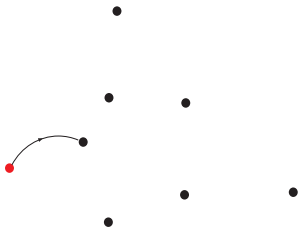
The graph is then represented in the code as a series of verticies:



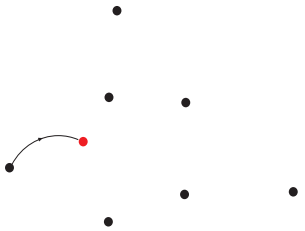
An edge is then selected, for each *edge* in *graph* do



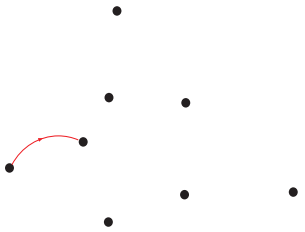
As the edges are directed lines, the starting root is then selected as the first argument of the edge $\langle ij \rangle$, $sRoot, sPath \leftarrow ancestor\,[edge.start]$



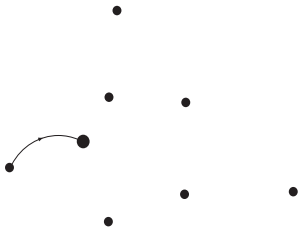
The second argument of edge $\langle ij \rangle$ is then selected to be the end root of this edge, $eRoot, ePath \leftarrow ancestor\,[edge.end]$



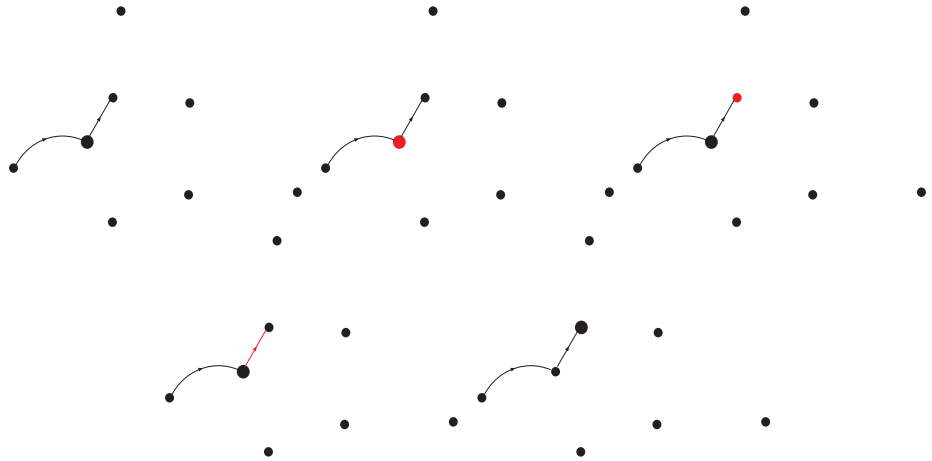
The route r of the graph is then defined as $r \leftarrow -sPath + edge + ePath$,



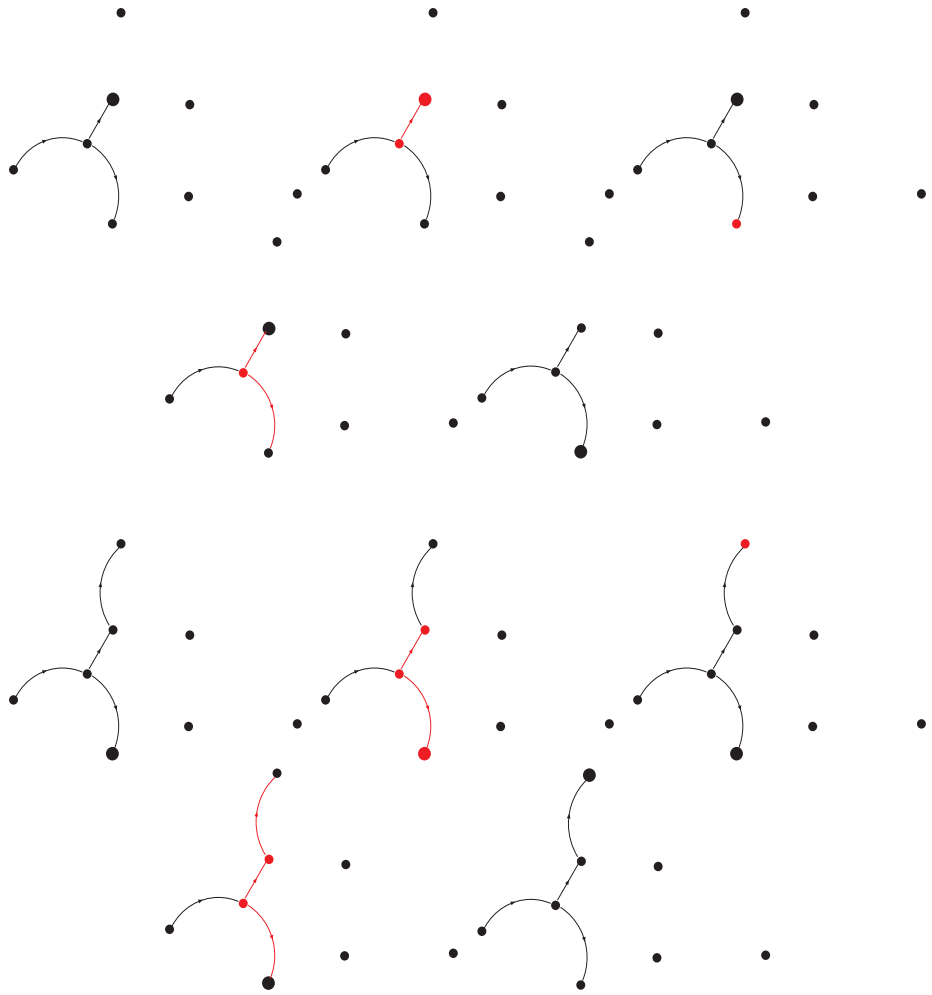
As the $sRoot \neq eRoot$ the source or ancestor of the edge is then defined to be end root of the edge; $ancestor\,[sRoot] \leftarrow eRoot, r$,

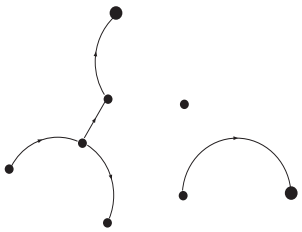


This process is repeated for the second selected edge:

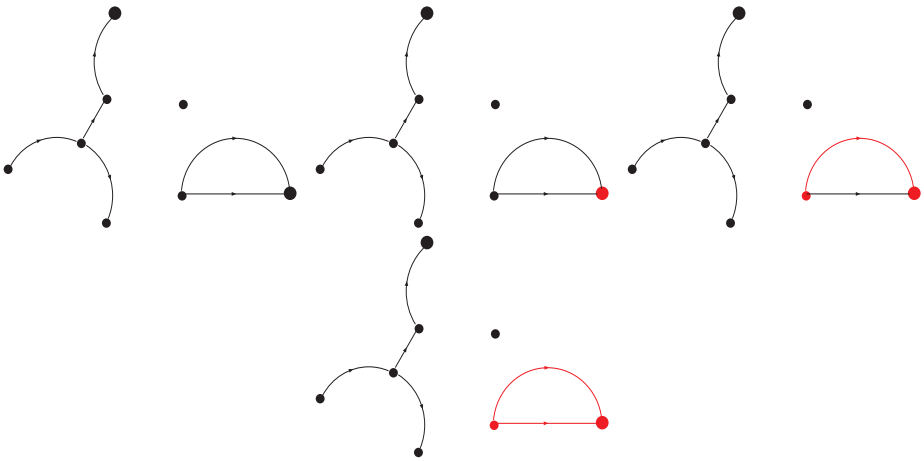


and so on:

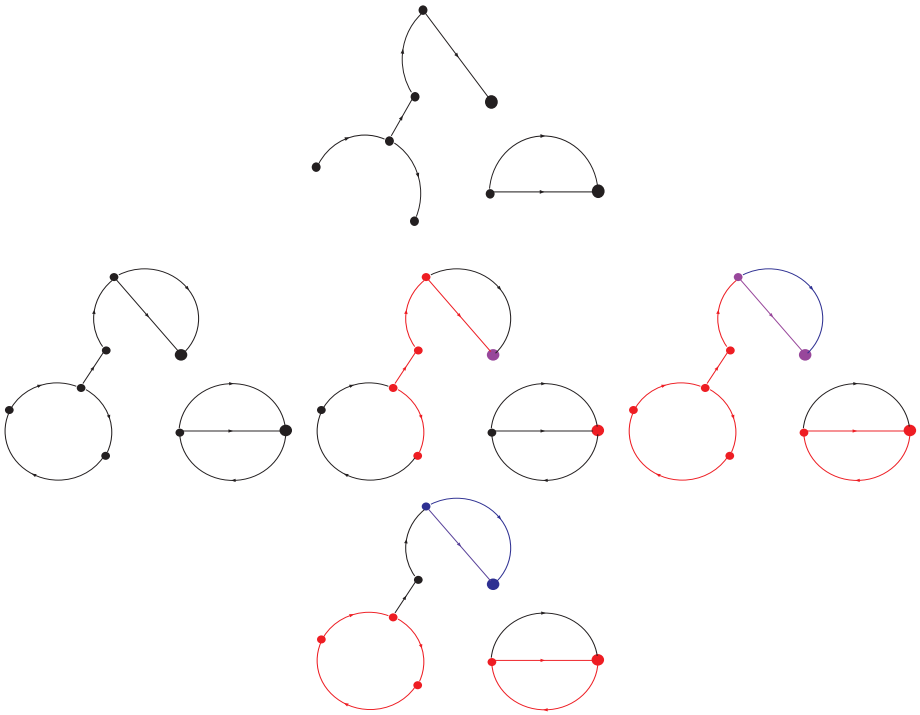




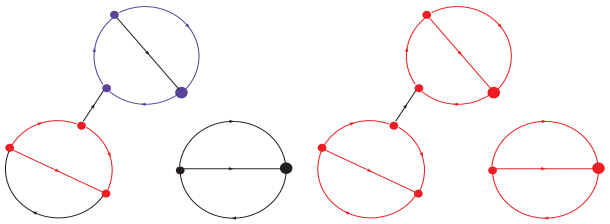
When an edge is selected which will form a loop, it can be seen that $sRoot = eRoot$ and thus the code selects the closed route of r and adds this value to a the list $loops$,add r to $loops$:



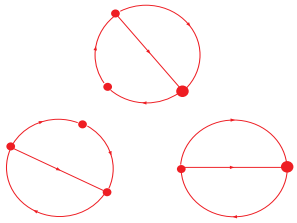
This process then continues untill the entire graph is covered.



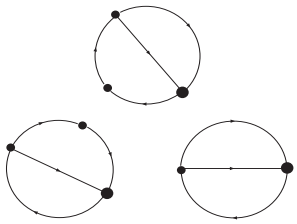
The 1PI pieces are found by running through the Galler-Fischer algorithm once and marking all the irreducible edges, removing the unmarked reducible edges and the running the Galler-Fischer algorithm once for the remaining 1PI and possibly disconnected pieces.



Where the above graphs show the results of running the Galler-Fischer algorithm once



delete *unmarkedEdges*



run *GallerFischer* once more.

Which is exactly the 1PI, as required.

3.4 Tensor Storage and Simplification

This section of the code has the routines for simplifying the tensor structures of the graph. It is split into three sections: the first has the routines for inserting momentum-space and spinor-space delta functions into the KTensor data structure as well as routines for reducing the KTensor structure to its simplest form. The second section ensures that conventions for the ordering of delta functions are enforced and the final section evaluates the products of the delta functions. The idea behind these routines, is to allow the insertion of tensor objects into the numerator without having to worry about order or whether it is a product or sum of tensor objects. It works by effectively

wrapping the existing tensor structure in a pair of braces and multiplying the new tensor objects to this, and when another more tensor objects are inserted the same happens again, ending up with multiple layers of sums and products of tensor objects . These would all be resolved in the tensor flattening routines which collapses down the layers into a single sum of products and the matching indices are tied together and resolved with the tensor simplification routines.

3.4.1 KTensor Flattening Rountines

As defined in section 3.2.5, the tensor structures of the graph are held in the *KTensor* and take the form of sums of products of Kronecker delta functions. However due to the recursive nature of much of the code, a way to correctly place every new delta function into the code is needed. This means that the *KProd* procedure has to accept not just the Kronecker deltas but also *KTSums*. This can lead to the KTensor taking the form of the below example:

$$KTensor(KTSum(KProd(4, \delta(I1, 1, I2, 1), \delta(I3, 1, I4, 1), \\ KTSum(KProd(1, \delta(E1, 2, E2, 2)), KProd(1, \delta(I2, 1, I4, 1)))))). \tag{3.2}$$

Although this is actually a fairly simple example, it aptly demonstrates how the KTensor is constructed.

Due to the non-trivial form the KTensor can take, care must be taken in how additional delta functions are inserted into it. The routine **insertKDelta** is designed to insert momentum-space Kronecker deltas into the KTensor with an analogous procedure for the insertion of the spinor deltas. The procedure takes the KTensor and a list of objects which can take the form of a product of delta functions, an integer times a product of deltas or a sum of products of deltas which will be inserted into the KTensor. If an integer times a product of deltas is inserted, the enforced the convention is:

$$KTSum(KProd(n, \delta_{ab}, \ldots, \delta_{cd}, KTSum(\ldots)))$$

where n is an integer, the indices a, b, c, d are the linename-numerator index index pair and the inner $KTSum$ is the old KTensor structure. For a product of deltas, the above convention is enforced with the integer $n = 1$. For an inserted KTSum, the procedure removes the KTSum(KProd()) wrapping and inserts the deltas as before.

As this method of having layers of the KTSums and KProds can lead to very large

terms which can be computationally expensive to move around, and as the terms will need to simplified before the Kronecker delta functions can evaluated. A method of 'flattening' these layers has been developed that effectively expands the various sums and products found in the KTensor into a single sum of products <algorithm 3¿.

Algorithm 3 Tensor Flattening Routines

KTSumExpand := **proc**(**alpha** :: **KTSum**)
$map(op@KProdExpand, alpha)$
endproc :
KProdExpand := **proc**(**alpha** :: **KProd**)
$coF, sigma, delta := op(1, alpha), selectremove(ks- > op(0, ks) =' KTSum', [op(2.. - 1, alpha)]);$
$KPRec1(map(KTSumExpand, sigma), delta)$

KPRec1 := **proc**(**sigma** :: **list**(**KTSum**), **delta** :: **list**(**KroneckerDelta**))
if sigma = [] **then**
$KTSum(KProd(coF, op(delta)));$
else
$KPRec2(op(1, sigma), KPRec1([op(2.. - 1, sigma)], delta))$
end if
endproc :
KPRec2 := **proc**(**Sigma** :: **KTSum**, **delta** :: **KTSum**)
$map(KPRec3, Sigma, delta)$
endproc :
KPRec3 := **proc**(**Pi** :: **KProd**, **delta** :: **KTSum**)
$op(map(KPRec4, delta, Pi))$
endproc :
KPRec4 := **proc**(**Pi**, **PiPrime**)
$KProd(op(1, PiPrime) * op(1, Pi), op(2.. - 1, Pi), op(2.. - 1, PiPrime))$
endproc :

endproc :

The **KTSumExpand** procedure uses the composite operator @ which applies the procedure **KProdExpand** to each element of KTSum and then removes the outer KTSum wrapping from the returned result of **KProdExpand**. Using the example of the KTensor (3.2), as the KTSum only has one element **KTSumExpand** will return:

```
KTSum(op(KProdExpand(KProd(4, KroneckerDelta(I1, 1, I2, 1),

  KroneckerDelta(I3, 1, I4, 1), KTSum(KProd(1, KroneckerDelta(E1, 2, E2, 2)),

  KProd(1, KroneckerDelta(I2, 1, I4, 1)))))))
```

For each term passed to **KProdExpand** function, the function splits this term into a list of KTsum parts which it sends back through **KTSumExpand**, a numerical coefficient taken from the KProd and the remaining Kronecker Delta terms from the KProd term. In the example, this gives

```
"alpha in KPRodExpand is",  KTSum(KProd(1, KroneckerDelta(E1, 2, E2, 2)),
                                  KProd(1, KroneckerDelta(I2, 1, I4, 1)))
 "coF in KProdExpand is", 4
"delta in KProdExpand is", KroneckerDelta(I1, 1, I2, 1),
                                KroneckerDelta(I3, 1, I4, 1)
```

which gets passed to the **KPRec1** function, after the KTSum term is recursively passed back through the **KTSumExpand** function to ensure that each element of the list only contains a sum of products of delta functions. This list is then passed on to the next module. In the example, the inner KTSum gets sent back through **KTSumExpand** to give:

```
KProd(1, KroneckerDelta(E1, 2, E2, 2))
```

and

```
KProd(1, KroneckerDelta(I2, 1, I4, 1))
```

at which point the recursion terminates as there are no more layers of KTSum left.

These delta functions of these two terms are then sent on to the **KPRec1** function with a KTSum term of [], which returns

```
KTSum(KProd(1, KroneckerDelta(E1, 2, E2, 2)))
```

and

```
KTSum(KProd(1, KroneckerDelta(I2, 1, I4, 1)))
```

Both of these terms now get the op applied to them and the **KTSumExpand** returns:

```
KTSum(KProd(1, KroneckerDelta(E1, 2, E2, 2)),
              KProd(1, KroneckerDelta(I2, 1, I4, 1)))
```

Now that the inner KTSum has been resolved, this is then passed on to the **KPRec1** procedure along with Kronecker delta functions of the outer most layer.

The recursion of $KPRec2(op(1, sigma), KPRec1([op(2..-1, sigma)], delta))$ is designed to create a single sum of products of delta functions. This is done by recursively reducing the number elements in the list of KTSums until it is empty, and a final KTSum is constructed out of the Kronecker Deltas with the **if** statement $\mathbf{if\ sigma} = []\mathbf{then KTSum}(\mathbf{KProd}(\mathbf{coF}, \mathbf{op}(\mathbf{delta})))$. This new KTSum is then

passed to **KPRec2** along with the nth element of the list KTSums. **KPRec2** will return a single KTSum which will be passed back into **KPRec2** with then $(n-1)$th element of the list of KTSums. This process will continue until all elements of the list KTSums has been used up and a single KTSum term is returned. In the example, there is one element in the list of KTSums:

```
[KTSum(KProd(1, KroneckerDelta(E1, 2, E2, 2)),
               KProd(1, KroneckerDelta(I2, 1, I4, 1)))]
```

so the Kronecker Delta terms passed from **KProdExpand** are formed into a KTSum:

```
KTSum(KProd(4,KroneckerDelta(I1, 1, I2, 1), KroneckerDelta(I3, 1, I4, 1)))
```

and both of these terms are passed on to the **KPRec2** procedure.

The procedure **KPRec2, KPRec3** and **KPRec4** merge the two KTSum terms into a single KTSum. This is done by first mapping the function **KPRec3** over the KTSum terms *Sigma* and *delta*, which has the effect applying **KPRec3** to each element in *Sigma*, a KPRod term, and having *delta* as the 2nd argument to the function, or more explicitly $KTSum(KPRec3(op(1, Sigma), delta), ..., KPRec3(op(-1, Sigma), delta)$. In the example, taking the first element of *Sigma* the following is obtained:

```
KTSum(KPrec3(KProd(1, KroneckerDelta(E1, 2, E2, 2)),
     KTSum(KProd(4, KroneckerDelta(I1, 1, I2, 1),
                                    KroneckerDelta(I3, 1, I4, 1)))
```

KPrec3 maps the function **KPrec4** over the arguments of the KTSum term and takes the KProd term as its second argument. The op removes the outer KTSum wrapping to return a list of KPRod terms. In the example, using the first element of the KTSum term, the two following KProd terms are passed to the function **KPRec4**:

```
KProd(4, KroneckerDelta(I1, 1, I2, 1), KroneckerDelta(I3, 1, I4, 1))
```

and

```
KProd(1, KroneckerDelta(E1, 2, E2, 2))
```

In the **KPrec4** procedure, the numerical coefficients of the two KProd terms are multiplied together and remaining Kronecker Delta functions are multiplied together to give a list of Kronecker Deltas in single KProd term. The example gives:

```
KProd(4, KroneckerDelta(I1, 1, I2, 1), KroneckerDelta(I3, 1, I4, 1),
                KroneckerDelta(E1, 2, E2, 2))
```

This is done for every pair combination of the two KTSum terms passed into the function **KPRec2**.

After the flattening procedures have been run through, all that remains is a single KTSum term which only contains KProd terms containing Kronecker delta functions. The final result of the example is given below:

```
KTSum(KProd(4, delta(I1, 1, I2, 1), delta(I3, 1, I4, 1),
          delta(E1, 2, E2, 2)), KProd(4, delta(I1, 1, I2, 1),
          delta(I3, 1, I4, 1), delta(I2, 1, I4, 1)))
```

3.4.2 Canonicalization

The Canonicalization procedures use the symmetry on the indices of the Kronecker delta function and commutativity of the Kronecker delta itself to enforce a specific ordering on the Kronecker Delta and KProd terms, as the ordering does become important in some aspects of the code.

The **canonicalKD** function enforces the convention of having the linenames which index the Kronecker Delta's in a lexicographical order or if the linenames are identical it imposes an ordering on the numerator index such that for a Kronecker Delta $\delta_{\alpha_i,\beta_j}$ the indices have the following constraints: $\alpha < \beta$ or if $\alpha = \beta$ then $i < j$ for α and $\beta = E_\mu$ or I_μ with $\mu, i, j \in \mathcal{Z}^+$ the set of all positive integers.

The **canonicalKD** function only works on true Kronecker Delta functions as no ordering is need on the free indexed momentum terms P_{μ_i}.

The convention that is imposed on a product of Kronecker Deltas is that the canonicalized Kronecker Delta with lowest lexicographic configuration is always on the left of the expression and the one with highest is on the right if the list of Kronecker Delta functions. The free indexed numerator momenta or always on the right of the Kronecker Delta functions in a lexicographical order. If any two Kronecker Deltas or numerator momenta have the same lexicographical ordering then the one with the lowest numerator index is on the left.

The **compareKD** procedure takes two Kronecker delta or numerator momenta and checks to see if they are in the conventional order and returns a boolean variable. It does this by first checking the number of elements in each argument. If the number of elements are different it checks to see if the numerator momenta is on the right of the Kronecker and returns a value of "true" if it is and false if it isn't. If the number of elements are the same for each argument it then checks the lexicographical ordering on the first linename (which have already been canonicalised) and returns true or false depending on whether the arguments are in order. If the first linenames are the same lexicographically then the second linenames are checked (this of course does not happen for the numerator momenta) and as before a true or false is if one is greater than the

other. If the second set of line names are the same then the numerator index are compared and a inequality is returned.

3.4.3 tensorSimplify: Tensor simplification routines

In this section are two routines for the simplification of tensor structures **tensorSimplify** and **tensorSimplify2**. Both of these routines work by using the Galler-Fischer algorithm <algorithm 1> to find the connected components, i.e. matching up the indices of the Kronecker Delta functions and numerator momenta. The matched indices are then contracted. The essential difference between these two routines is that **tensorSimplify** is designed to be used at the end of the program and will complain if there are any unmatched indices, whereas **tensorSimplify2** is designed to be used within the code to keep the size of KTensor manageable by contracting all matched indices and leaves all free indices. The tensor simplification routines work by taking in a KProd term and extracting the numerical coefficient into the variable $coeF$ and the Kronecker Deltas and free numerator momenta into the variable $kt1$. The indices of the Kronecker Deltas are then formed into a set of $IndexPair$s which consist of the a linename with numerator momenta index, for example the Kronecker delta $\delta_{I1_1,I2_1}$ would form the index pairs of $IndexPair(I1,1)$ and $IndexPair(I2,1)$. The free numerator momenta are formed into a set of $IndexTriple$s which consist of momentumname, linename and numerator index, the linename and numerator index are also put into the set of all indexPairs. The Galler-Fischer algorithm is then applied to the $IndexPair$ and $IndexTriple$.

tensorSimplify

As the routing of the connected pieces is unimportant, the **ancestor** function is used rather than the **ancestorRoute** function <algorithm 2>: The elements of the KProd are then split $coeF, kt1 := op(1, kt), [op(2..-1, kt)]$, with each kronecker deltas and free index numerator momenta being sent around <algorithm 4> in a **do** loop to form the $IndexPair$ and $IndexTriple$:

The resulting $IndexPair$ and $IndexTriple$ are then passed to the Galler-Fischer algorithm, which looks for closed loops ($\delta_{ij}\delta_{ij}$) it replaces the Kronecker Deltas with a factor of 2ω the space-time dimension. The Kronecker Deltas which form a closed loop are then stored in the list $loopRoots$. At this point the next Kronecker Delta or free index momenta is passed through algorithms the Galler-Fischer algorithms until all elements of $kt1$ have been used.

As the result of **tensorSimplify** requires that all that remains is a constant times dot products of momenta, all Kronecker Deltas that do not form closed loops have

Algorithm 4 indexPairs: tensorSimplify

if $nops(kd) = 4$ **then**
  $s := IndexPair(op(1..2, kd))$;
  $t := IndexPair(op(3..4, kd))$;
  $indexPairs := indexPairsunion\{s, t\}$
else if $nops(kd) = 3$ **then**
  $s := IndexPair(op(1..2, kd))$;
  $t := IndexTriple(op(1..3, kd))$;
  $indexPairs := indexPairsunion\{s\}$;
  $momenta := momentaunion\{t\}$;
else
  ERROR("Bad KroneckerDelta structure", kd)
end if

to contract with the free indices on the numerator momenta. This is achieved by <algorithm 5> which uses the ancestor function on the set of sets of $IndexPair$s and $IndexTriple$s and removes all Kronecker Deltas that form closed loops from this set. The numerator momenta indices must then match up with each other in dot products or an error is returned. Finally the resulting constant times dot products are multiplied by the numerical coefficient of the KProd and the final result is then moved into the overall factor of the Feynman Parametric graph ($op(1, FPGraph)$).

Algorithm 5 tensorSimplify pair matching

$roots := select(v- > ancestor(v) = v, indexPairsunionmomenta)$;
$roots := rootsminus\{loopRoots\}$;
$momPairs := map(a- > select(p- > ancestor(p) = a, momenta), roots)$;
if $nops(momPairs) > 0$ and $map(nops, momPairs) \neq \{2\}$ **then**
  ERROR(sprintf("Unmatched momenta
end if
if $nops(momPairs) \neq nops(roots)$ **then**
  ERROR(sprintf("Paths
end if
$res1 := res1 * ` * `(op(map(pq- > dotProduct(op([1, 3], pq), op([2, 3], pq)),$
    $[op(momPairs)])))$;
$res := res1 * coeF$;

Below is an example of an input KProd and the resulting dot product.

```
tensorSimplify: Simplifying tensor expression
   KProd(8, KroneckerDelta(I100, 2, I101, 1), KroneckerDelta(E100, 2, p),

     KroneckerDelta(E100, 3, p), KroneckerDelta(I101, 1, E100, 2),

     KroneckerDelta(I100, 2, E100, 3))
```

```
tensorSimplify: Roots are {IndexTriple(E100,3,p)} of which loopRoots are {}
tensorSimplify: Simplified tensor expression
                              8 dotProduct(p, p)
```

tensorSimplify2

As the purpose of **tensorSimplify2** is not to resolve the KTensor structure but to simplify it, there are a few deviations from the structure of **tensorSimplify**. The basic premise **tensorSimplify2** is the same as **tensorSimplify**, after the coefficient of the KProd has been retrieved, the Kronecker Deltas and numerator momenta form **IndexPair**s and **IndexTriple**s which are then looped over and passed through a Galler-Fischer algorithm to find closed loops.

In order to have simplifications of the type $\delta_{ij}\delta_{jk} = \delta_{ik}$, it is necessary to keep track of repeated indices. This is done by modifying Galler-Fischer algorithm such that as the algorithm is forming the**IndexPair**s and **IndexTriple**s, if it comes across an **IndexPair** or **IndexTriple** it has seen before it constructs a set containing these **repeatedPairs**

As there will be unmatched **IndexPair**s and **IndexTriple**s the pair matching algorithm <algorithm 5> has to be modifed:

The set **freePairPairs** is then the set "vertices" $\langle i, j \rangle$ of the numerator graph where i and j are **IndexPair**s and the set of **IndexTriple**s with loops and repeated indices removed. The set of the pair of **IndexPair**s form the the Kronecker Delta functions and the the set of the **IndexTriple** form the free indexed numerator momenta. The set of **freePairPairs** in conjunction with the numerical coefficient *coeF* times any factors of 2ω from closed loops form a new KProd which is inserted into the KTensor of the *Graph* or *FPGraph*. An example of this procedure it given below:

```
tensorSimplify2: Simplifying tensor expression
   KProd(1, KroneckerDelta(k1, 1, k1, 2),
     KroneckerDelta(I101, 1, k1, 2), KroneckerDelta(I100, 2, k1, 1),
     KroneckerDelta(E100, 3, p), KroneckerDelta(E100, 2, p))
tensorSimplify2: Roots are {IndexPair(k1,2), IndexTriple(E100,3,p),
                           IndexTriple(E100,2,p)} of which loopRoots are {}
tensorSimplify2: Roots are {IndexPair(k1,2), IndexTriple(E100,3,p),
                           IndexTriple(E100,2,p)}
tensorSimplify2: freePairs are {IndexPair(I100,2), IndexPair(I101,1),
                           IndexTriple(E100,3,p), IndexTriple(E100,2,p)}
tensorSimplify2: freePairPairs are {{IndexPair(I100,2), IndexPair(I101,1)},
                           {IndexTriple(E100,3,p)}, {IndexTriple(E100,2,p)}}
tensorSimplify2: Simplified tensor expression
```

```
KProd(1, KroneckerDelta(E100, 3, p), KroneckerDelta(E100, 2, p),
  KroneckerDelta(I100, 2, I101, 1))
```

The use of **tensorSimplify2** can dramatically decrease the number of terms in the KTensor and therefore can decrease the memory usage needed to move such objects between lists and tables as required by the code. This will become increasingly important with higher loop order graphs the user may wish to evaluate.

3.5 Static Diagram Analysis

3.5.1 Analysis of the topology of a diagram

The purpose of this procedure is to take the basic *diagram* input and create a list of graph pseudofunctions which contain all the information necessary for renormalization and calculation of the graph using the Feynman rules of the appropriate theory. This is done by analyzing the topology of the graph by determining if a given graph has the desired properties of connectivity and being 1PI. Provided that these checks are passed, the module then finds provides a momentum routing through the graph and creates a sequence of graph pseudofunctions which contains all the dynamical infromation about the diagram along with a collection of global tables which contain the static information of the diagram. **AnalyzeDiagram** first creates a graph pseudofunction that has the purely topological information of the graph, this is called the **protoG** or proto-graph, at which point the numerator procedures are called which take the protoG pseudofunction and create a list of graphs with the correct numerator functions inserted. A list of graph pseudofunctions is created as structures of the type $\frac{\gamma_\mu(\not{a}+m)\gamma_\nu(\not{b}+m)}{a^2+b^2}$ are explicitly expanded out, with each element of the list being one of the sum of terms from the expansion which then allows the terms containing an odd number of γ-matrices to be killed. The momentum-slash terms $\not{a}$, however, are always kept as sums of momenta.

The following global variables are invoked:

```
global denomDegree, diagHash, externalMomenta, lineEnds,
            lineType, lineVertexTypes, loopMomenta, numDegree,
            numLoops, numberOfLoops, vertexCoordinates, vertexLineEnds,
            vertexLineTypes, vertexType, vertices;
```

The global sets of externalMomenta and loopMomenta, which are just the set of momentum names, and vertices, which as the name implies is the set of vertices in the diagram, are all initialised to the empty set. The global tables VertexType and lineType contains the vertex and line types indexed by the vertex and line names. LineEnds contains the vertexnames associated with the two ends of the indexing linename.

After the diagram pseudofunction is passed to the **analyzeDiagram** procedure it is first checked to check to see if it is of the correct type and creates a hash code for the diagram which is used to make certain checks in the code. As has been discussed already in section 3.3, the Galler-Fischer algorithm, <algorithm 1> , finds the topological properties with the use of pairwise equivalence relations of vertices. So it is necessary to first construct the vertex and internal pseudofunctions.

Each vertex in the diagram pseudofunction is checked against the set of all vertices to see if it is duplicate, if it is not it is then added to this set. Each type of vertex in the diagram pseudofunction is the entered into a table vertexType which is indexed by the vertex name, so the various properties associated with Feynman rules of each vertex type can be accessed later <algorithm 6>.

Algorithm 6 Process vertex pseudofunctions

```
for v in select(type, d, vertex) do
  vType := op(0, v);
  vName, vCoordinates := op(v);
  if member(vName, vertices) then
    ERROR(sprintf("Duplicate vertex %d", vName))
  end if
  vertices := verticesunion{vName};
  vertexType[vName] := vType;
  vertexCoordinates[vName] := vCoordinates
end for
```

Likewise, the line pseudofunctions take take the internal lines of the diagram pseudofunction and enters the line type in to a table lineType indexed by line name. The lines are then checked against the set of all vertices to see if all the internal lines are connecting the vertices of the diagram and returns an error if it finds a line connecting to a vertex not in the set of all vertices. The internal lines are then attached to the vertices by calling the **setVertexLinEnds** function and the Galler-Fischer algorithm is then applied to the diagram.

The graph is then separated into its connected components by finding the ancestor of each connected graph, which is defined to the be the **componentRoots** <algorithm 7>. The **componentVertices** are then the set of vertices within each connected component and finally a new diagram pseudofunction is created for each connected piece, i.e. it splits the original diagram pseudofunction into new diagram pseudofunction for each disconnected diagram.

Once the connected pieces of the diagram have been found, the external lines pseudofunctions are created and attached to the appropriate vertices. The external momenta is routed through the graph by taking the moment name from each external line pseudofunctions and multiplying this factor to each line in the ancestorRoute route (the

Algorithm 7 Find connected components

```
componentRoots := select(v- > ancestor(v) = v, vertices);
componentVertices := map(a- > select(v- > ancestor(v) = a, vertices),
        componentRoots);
componentDiagrams :=
      map(vset- > select(pf- > foldl('or','false',
                  op(map(member, pf, vset))), d),
      componentVertices);
if nops(componentDiagrams) ≠ 1 then
  ERROR(sprintf(cat("Diagram has %d (dis)connected components:",
       seq(sprintf("
  n
  t Component %d is %%a", j),
            j = 1..nops(componentDiagrams))),
       nops(componentDiagrams),
  op(map(d − > map(pf − > op(1,pf), d), componentDiagrams))))
end if
```

internal lines which connect the vertex attached to the external line and the ancestor vertex), this is stored as a sum terms in the variable **allMom**. A table momentum is created indexed by (external) line name and contains the appropriate value of the momentum in that line. Edge pseudofunctions are then created for the external lines, with momentum taken from the external line pseudofunction and the denominator degree from the Feynman rules. The numerator momenta is empty at this point as only the topology of the diagram is considered, numerator factors are substituted in later. The momentum routing is completely arbitrary and is just the father path from the Galler-Fischer algorithm. However, due to its arbitrary nature, the routing can lead to the number of terms needing to be differentiated with respect to the external momenta to be quite large, a possible future optimization would be to find the shortest route between the external fields (i.e. the path which passes through the fewest number of edges).

The sequence of external line names *extLines*, is used in the graph pseudofunction to keep track of lines external to section of the graph which is being analyzed at the time.

The table *consMom* enters the external line momenta for the ancestor of the external line and is used to check the momentum conservation of the graph:

Internal edge pseudofunctions are constructed in a similar way, with the loop momenta variables for the graph pseudofunction being constructed by concatenating the letter "k" with a number equal to the number of elements in the loopMomenta set +1. This loop momentum variable is then converted into *type/symbol* and added to the set

Algorithm 8 External line pseudofunctions and external momentum routing

$lineEnds[lName] := vName;$
$setVertexLineEnds(vName,$
$\quad map(field- > vertexLineTypes[vertexType[vName], field],$
$\quad \{lineFieldTypes[lType]\}),$
$lName);$
$externalMomenta := externalMomentaunionindets(mom);$
$a, r := ancestorRoute(vName);$
$allMom := allMom + mom * r;$
$momentum[lName] := mom;$
$consMom[a] := consMom[a] + mom;$
$edges[lName] := edge(mom, denomDegree[lType], NumeratorMomenta(0));$
$extLines := extLines, lName$

of loopMomenta.

$$allMom := allMom + momentumName() * loop$$

Once all the edge pseudofunctions are created and stored in the edge table, a check is then performed on the topological information of the graph to see if it is consistent with the Feynman rules defined earlier (section 3.2.3). As this program is only concerned with the calculation of loop graphs, a final check is made that the connected pieces of the graph or 1PI. If they are not, it means either that the graph is strictly tree-level or that graph is disconnected.

Algorithm 9 1PI Graph check

$numberOfLoops[[intLines]] := nops(loops);$
$loopLines := `union`(map(op@indets, loops));$
$reducibleEdges := remove(i- > member(i, loopLines), [intLines]);$
if $nops(reducibleEdges) \neq 0$ **then**
ERROR(sprintf("Diagram is not 1PI. Edge(s) %q are one particle reducible.",
op(reducibleEdges)))
end if

Once these checks have been performed, a graph pseudofunction is created:

$$protoG := graph(1, KTSum(KProd(1)), edges, [extLines], [intLines], diagHash, \\ 'I0', GR()); \tag{3.3}$$

As stated at the beginning of this section, this *protoG* graph pseudofunction contains only the topological structure of the graph. This protoG forms the basis of the graph calculations and is sent to the **setNumeratorFactors** procedure which inserts

the appropriate numerical structure for the graph:

$sNL := setNumeratorFactors(protoG)$;

The procedure ends by returning a list of graphs ready for renormalization. $[sNL]$

An example of the workings of the *analyzeDiagram* procedure is given below for the the standard scalar 2-loop graph (figure 3.1) is given below:

```
analyzeDiagram: Route from V4 to V5 is I4
analyzeDiagram: Route from V7 to V5 is I5+I4
analyzeDiagram: Lines I6+I5+I4 form a loop
analyzeDiagram: Route from V5 to V6 is I7
analyzeDiagram: Lines I8+I7+I5+I4 form a loop
analyzeDiagram: Momentum in line I4 is p3+k1+k2
analyzeDiagram: Momentum in line I5 is k1+k2
analyzeDiagram: Momentum in line I6 is k2
analyzeDiagram: Momentum in line I7 is p3+k1
analyzeDiagram: Momentum in line I8 is k1
analyzeDiagram: Slot 1 of ThreeScalarVertex vertex V4 connected to line I4
analyzeDiagram: Slot 2 of ThreeScalarVertex vertex V4 connected to line I5
analyzeDiagram: Slot 3 of ThreeScalarVertex vertex V4 connected to line E4
analyzeDiagram: Slot 1 of ThreeScalarVertex vertex V5 connected to line I4
analyzeDiagram: Slot 2 of ThreeScalarVertex vertex V5 connected to line I6
analyzeDiagram: Slot 3 of ThreeScalarVertex vertex V5 connected to line I7
analyzeDiagram: Slot 1 of ThreeScalarVertex vertex V6 connected to line I7
analyzeDiagram: Slot 2 of ThreeScalarVertex vertex V6 connected to line I8
analyzeDiagram: Slot 3 of ThreeScalarVertex vertex V6 connected to line E5
analyzeDiagram: Slot 1 of ThreeScalarVertex vertex V7 connected to line I5
analyzeDiagram: Slot 2 of ThreeScalarVertex vertex V7 connected to line I6
analyzeDiagram: Slot 3 of ThreeScalarVertex vertex V7 connected to line I8
analyzeDiagram: 2 loop diagram
     [graph(1, KTSum(KProd(1)), edges, [E4, E5], [I4, I5, I6, I7, I8],

       "diagram_diag2", I0, GR())]
```

where the edge table is:

$$\begin{aligned} edgeTable\,(PROP(I8) &= 1/(k1^2+m^2), PROP(I7) = 1/((p3+k1)^2+m^2), \\ PROP(I5) &= 1/((k1+k2)^2+m^2), PROP(E5) = 1/(p3^2+m^2), \\ PROP(I6) &= 1/(k2^2+m^2), PROP(E4) = 1/(p3^2+m^2), \\ PROP(I4) &= 1/((p3+k1+k2)^2+m^2)\big) \end{aligned} \tag{3.4}$$

3.5.2 Ordered Vertex Line Ends

For non-scalar field theories, the direction of an edge entering a vertex can be important as it is used to distinguish fields from anti-fields (such as electron and positron in QED). In order to achieve this, each vertex is given a number of slots determined by the interactions of the field theory in question (for example, there will be 3 slots in each vertex for ϕ^3-theory and QED), furthermore, each slot can only accept one type of field which are defined in the Feynman rules (section 3.2.3). This is implemented by taking a vertex name, a line name and a set of slot numbers which are obtained from the Feynman rules of the theory. With these three inputs, a new table is set-up, *vertexLineEnds*, which equates the line name to an element of the table addressed by the vertex name and slot number. If the table element has an exisiting entry when the procedure is called, an error is returned.

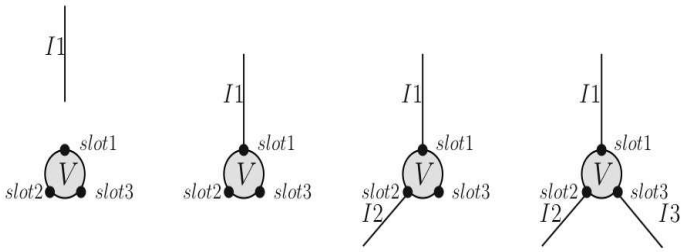


Figure 3.2: From left to right shows diagrammatically how a scalar vertex has its edges assigned to slot numbers

An example of how the **setVertexLineEnds** procedure works is given below for both ends of an edge $I4$:

```
setVertexLineEnds: Trying to connect scalar line I4 into slots {1, 2, 3}
of ThreeScalarVertex vertex V4
setVertexLineEnds: scalar line I4 connects to slot 1
 of ThreeScalarVertex vertex V4
setVertexLineEnds: Trying to connect scalar line I4 into slots {1, 2, 3}
 of ThreeScalarVertex vertex V5
setVertexLineEnds: scalar line I4 connects to slot 1
 of ThreeScalarVertex vertex V5
```

As this is a scalar theory, the edge fits into the first available slot on the vertices which in this case is *slot 1* for vertices $V4$ and $V5$.

3.5.3 Numerators

After the *protoG* has been constructed, it is sent into the **setNumeratorFactors** procedure which takes a graph pseudofunction, protoG, and returns a list of graph pseudofunctions corresponding to the full algebraic expression of the the input diagram with the numerator structure expanded and the spinor structure simplified.

The code deals with scalar propagators of type $\frac{1}{k^2+m^2}$, boson propagators of type $\frac{\delta_{\mu\nu}}{k^2} + \frac{\xi k_\mu k_\nu}{k^4}$ and fermion propagators of type $\frac{\not{k}+im}{k^2+m^2}$ along with their derivatives. The momenta numerator structures of these propagators are stored as vector indices in the the numerator momenta list with the remaining numerical coefficients and mass-terms of the numerator being incorporated into the overall factor of the graph. Momenta contained within vertices are offloaded into an adjoining fermion lines.

In the case of the boson term $\frac{\delta_{\mu\nu}}{k^2}$, the numerator contains no momenta but the delta function ties together the momenta of the adjoining vertices. In order to get the appropriate vector index for the fermions when attached to a vertex with a boson propagator attached the *getVectorIndex* procedure <algorithm 10> is used.

Algorithm 10 Vector Index Assignment

getVectorIndex := **proc**(**lN** :: **linename**, **lEnd** :: **literalBoolean**)
option remember; local maxIndex, num, edge; global newET;
$edge := edges[lN]$;
$num := op(3, edge)$;
$maxIndex := op(1, num) + 1$;
$num := subsop(1 = maxIndex, num)$;
$edge := subsop(3 = num, edge)$;
$edges[lN] := edge$;
$maxIndex$;
endproc :

The **Vector Index Assignment** algorithm, <algorithm 10 >, takes a linename and a boolean variable corresponding to the start and end of the linename and increases the maximum index of the numerator momenta list by 1, the *option remember* is used which adds an additional argument to the procedure, the result of the procedure, and returns this value whenever the same linename and boolean variable are called. This optimization is used as the *getVectorIndex* procedure will be called multiples with the same inputs.

As both the fermion and boson numerators contain two distinct terms, it is necessary to expand the graph in to all possible combinations of the the fermion and boson numerators. This is done by taking the list of internal lines and for each linename within this list, it forms an edge-numerator pair $eNPair()$ which pairs the linename with the value *red* and *blue* (the "colours" are just for debugging convenience, any marker would

do) , for example $eNPair(I1, Red), eNPair(I1, blue)$. A list of Numerator Graphs, $numG$, is then formed which contain all the possible eNPair combinations for the list of linenames, this list will contain 2^I $numG$'s containing I $eNPair$'s with I being the number of internal lines.

Algorithm 11 Numerator Expansion Procedure

f := **proc**(**L**)
$map(L1 \rightarrow NumG(op(L1)), subs(DUMMY = NULL, frec(L, DUMMY)))$
endproc :

frec := **proc**(**L**, **out**)
$`if`(nops(L) > 0, frec([op([2..-1], L)], h([[eNPair(op(1, L), red)],$
$[eNPair(op(1, L), blue)]], out)), out)$
endproc :

h := **proc**(**L1**, **L2**)
local result, i, j:
$result := []$:
for i in $L1$ **do for** j in $L2$ **do**
$result := [op(result), [op(i), op(j)]]$:
end for: end for:
endproc :

For example, a graph with two internal lines $I1$ and $I2$ will produce the following sequence:

```
[NumG(eNPair(I2, red), eNPair(I1, red)), NumG(eNPair(I2, red),
  eNPair(I1, blue)), NumG(eNPair(I2, blue), eNPair(I1, red)),
   NumG(eNPair(I2, blue), eNPair(I1, blue))]
```

This list is then passed on to the Internal Line Numerator Structure Setup procedure which then replaces the numG's with the appropriate graph pseudofunction.

For the special case of a purely scalar diagram, the *setNumeratorFactors* will return a graph pseudofunction list containing a single element which has the exact value of the *protoG* graph pseudofunction created in the *analyzeDiagram* procedure.

To ensure this each of the internal lines are checked to see if they are *'type/scalar'* and returns the input argument if it is true. Graphs with scalar propagators along with the fermion and boson propagators is not yet implemented.

The fields external to the graph are best dealt with, in the context of the program, with a projection operator to tie them together. The resulting delta functions are then inserted into the *KTensor*, which up until this point would have been empty. Currently

only two external bosons are allowed in the program, with a proper projection operator algorithm yet to implemented.

The internal lines are first sent to the numerator expansion procedure, <algorithm 11>, and for each element in the returned list a new copy of *edge table*, newET, is created. For each linename, the appropriate edge is retrieved from the edge table and the constituent parts of the edge (momenta, denominator degree and numerator momenta) are assigned to new variables for optimization reasons (the information will be called multiple times, so assigning it to a local variable will avoid having to have the information retrieved every time from the look-up tables). The current maximum numerator vector index is also assigned to a new local variable. With this done, each linename in the eNPair is checked to see if it is of *'type/spinor'* or *'type/boson'*.

For the spinor lines, the vertices attached to the start and end of the spinor line are retrieved and these are used to find the boson lines attached to the vertices (canonically assigned to the number 3 slot in the 3-vertex interaction), $sVLine$ and $eVLine$, and boolean variables are assigned to new variables sVB and eVB which return *true* if the boson line starts at the start/end vertex and $false$ if it ends there. Once this is done, the code then checks the *red/blue* variable of the $eNPair$ which represent the two possible numerator structures for the spinor line. *Blue* represents the numerator structure $\not{k}$ and so the max numerator vector index is increased by 1 and this new max vector index is added on to the back of the list of numerator vector indices which is then inserted into the edge and the edge is entered into a new edge table. As stated in the definitions and declarations section of the code, the Dirac γ-matrices are not explicitly stored and can be represented by Kronecker delta functions connecting two γ matrices. As there are γ-matrices associated with each vertex the γ in the $\not{k}$ numerator can be represented by a pair of spinor delta functions. The arguments for the spinor delta structures are the boson line attached to the vertex at the start or end of the spinor line with the appropriate vector index and the the spinor line itself with the newly increased maximum vector index, remembering that the γ-matrices associated with each vertex has been offloaded onto the adjoining boson line. For the *red* $eNPair$ spinor lines, the numerator this represents is im, as such this factor is assigned to a variable which will be included in the overall factor of the graph once all the linenames in the graph have been evaluated. The Spinor structure is simply a single spinor delta connecting the γ matrices associated with the vertices at each end of the edge. This modified edge is then entered into the edge table.

For the boson lines, as they have no spinor structure, the $newSpin$ sequence entry is $NULL$. The *blue* $eNPair$s correspond with the numerator structure $\delta_{\mu\nu}$ and this Kronecker delta is constructed and inserted into the $KTensor$ structure for the graph. The maximum vector index in the numerator momenta is increased by the value of the vector index of the end of the boson line (which is canonically larger than the start of

the boson line), but these new indices are not added into the list of vector indices of the numerator momenta as there is no numerator momenta associated with this line. The *red eNPair*s correspond with the gauge fixing numerator structure $\xi\frac{k_\mu k_\nu}{k^4}$. The ξ is included overall factor of the graph, and maximum index of the numerator momenta is increased by 2 (one for each numerator momenta present) and the these are added to the list of numerator indices. The new edge created is then added into the copy of the edge table for this graph.

Once each internal edge of a graph from the graph sequence has been passed through, the overall coefficient is collected and the **KTensor** structure is sent to the **makespinor** procedure. **makespinor** constructs a spinor graph and simplifies the resulting Kronecker deltas into a single *KTensor* which is then simplified. If there are odd numbers of γ-matrices **makespinor** will return zero and the that element of the graphsequence will be replace with the value $NULL$. A graph pseudofunction, *newG*, is then created with the new edge table, KTensor and overall coefficient included and forms part of the sequence of graph pseudofunctions which are returned to the **analyzeDiagram** module.

3.5.4 Spinor tracing

The spinor structure of a graph can be represented graphically akin to the Feynman rules for QFT using a group theory technique of Birdtracks [83, 88] , the rules for which can be found in figure 3.3.

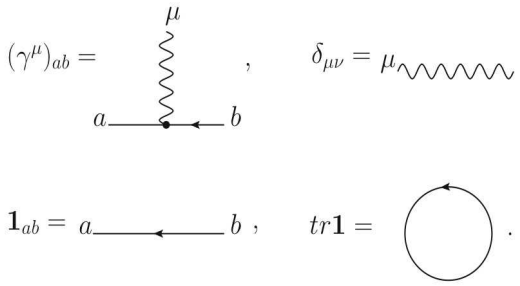


Figure 3.3: Spinography rules [88]

With these rules, a spinor graph can be constructed which contains all the spinor structures of the graph pseudofunction and due to its graphical nature, the Gallar-Fischer algorithm can be applied.

The **makespinor** procedure constructs takes two arguments *KSTensor* which holds the spinor structures of the graph and a *KTSum* which holds the information from the external field projection operation and any Kronecker delta functions from internal

boson lines. It uses these to construct a spinor graph and using the Gallar-Fischer algorithm < algorithm 1 >, closed loops can be found and replaced with the appropriate Kronecker delta functions using the spinor tracing routines < algorithm 14>.

As the order in which the spinor deltas are in is important, the ancestor route algorithm <algorithm 2> and the Galler-Fischer algorithm which keeps track of the route through the graph <algorithm 1> are used. The arguments for the ancestor route function are *indexPair*s of the spinor delta functions vis-à-vis the indexPairs of the momentum-space Kronecker delta functions in the tensor simplify routines <algorithm 4>.

Once the all the spinor loops and associated routes have been found the list of spinor deltas in the the spinor loops are converted into a set which removes any duplicates and this set is sent to the ordered pairs procedure <algorithm 3.5.4>

$spinorLoopSets := map(e- > \{op(1..-1, e)\}, [spinorLoops]);$
$oST := map(orderedPairs, spinorLoopSets);$

The **ordered pairs** procedure takes the set of spinor deltas in the spinor loops and creates a table *dex* labelled by the first indexPair of the spinor delta and returns the second indexPair from the same spinor delta. As the set of spinor delta functions has to form a closed loop, when the *dex* table is called with the second $indexPair$ it will return the first $indexPair$ of another spinor delta function, the first $indexPair$ of the first spinor delta function is assigned to a variable a which is used to find when the loop is completed . Using the *dex* table a sequence of $indexPair$s is created which are in the appropriate order and the sequence terminates when the closed loop is complete, this occurs when the *dex* table returns the same $indexPair$ as the first $indexPair$ examined, i.e. the variable a. The sequence of ordered $indexPairs$ is then sent to the **SpinorTrace** procedure <algorithm 14>.

The Order pairs procedure then returns a sequence of sums of products of Kronecker delta functions corresponding to the results of the spinor tracing routines.

Each element of the sequence from the **ordered pairs** procedure is then examined, if there are an odd number of γ-matrices the returned result is set to 0. The the integer and Kronecker delta elements of the ordered pairs sequence are formed into a $KTSum$ data structure and are returned to the **setNumeratorFactors** procedure.

The spinor tracing is achieved by first re-writting the Clifford algebra identity (3.1) using the Birdtrack notation rules introduced in figure 3.3.

Using the Clifford algebra in figure 3.4 with the rule for drawing traces from the Birdtrack rules figure 3.3, the spinor trace for a 2-γ spinor graph is obtained figure 3.5.

Applying figure 3.4 to a 4-γ spinor graph leads to figure 3.6.

This has reduced 4-γ spinor graph to a sum of 2-γ spinor graph times a Kronecker

Algorithm 12 Ordered Pairs

```
orderedPairs := proc(loop :: set(SpinorDelta))
local dex, s, l, a, b, oPair;
for s in loop do
  dex[op(1..2, s)] := op(3..4, s)
end for
a := op(1..2, op(1, loop));
b := dex[a];
l := indexPair(a);
while indexPair(a) ≠ indexPair(b) do
  l := l, indexPair(b);
  b := dex[b]
end while
oPair := expand(SpinorTrace([l]));
oPair
endproc :
```

Algorithm 13 Spinor Loop Contractions

```
for ST in oST do
  if ST = 0 then
    newST := 0
  else if type(op(1, ST),'integer') then
    ST2 := insertKDelta([op(ST)], kt)
  else
    ST1 := KTSum(op(map(KProd@op, ST)));
    ST2 := KTSum(op(map(e-> KProd(op(op(e, ST1)), kt), $1..nops(ST))))
  end if
  newST := KTSum(op(newST), ST2)
end for
if member(0, newST) then
  res := 0
else
  res := op(newST)
end if
res
```

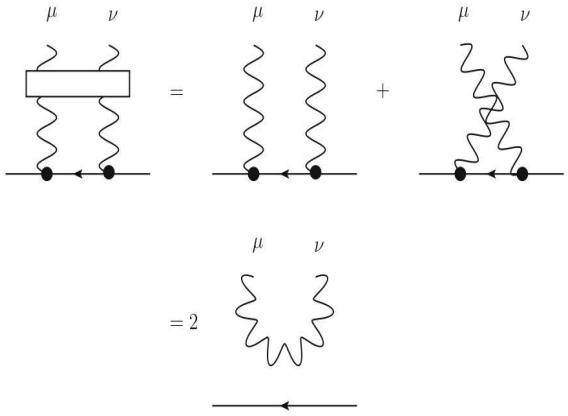


Figure 3.4: Clifford Algebra identity in Birdtrack notation

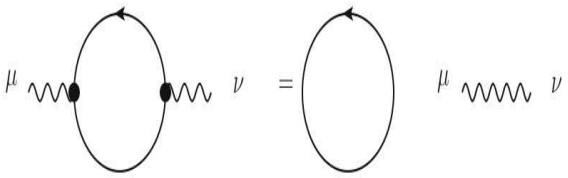


Figure 3.5: Trace of 2 spinors

delta function.

Taking the trace of figure 3.6 and using the cyclic property of a trace, figure 3.7 is obtained.

Immediately it can be seen that result of figure 3.7 with 3.5 leads to figure 3.8.

Successive application of figure 3.4 and 3.5 along with cyclic property of the trace leads to a method for taking the trace of an n-γ spinor graph.

The best way to see how the trace algorithm works is to look again at the 4-γ spinor graph figure 3.6. The γ-matrix on the left of the spinor graph is selected and using the Clifford algebra this gamma can then be commuted to the right, with each commutation the graph is replace with a new spinor graph which is a Kronecker delta times an $n-2$ spinor graph and the original graph with the two γ lines crossed, each time it crosses another γ-matrix there is a factor of -1 is applied to the graph. This is repeated until the first γ line has passed through every other γ line. As the graph is traced over, using the cyclical property of the trace this is just the original graph again. The same procedure is done recursively for the sum of $n-2$-γ spinor graphs until there is only a sum of products of Kronecker delta functions times a 2-γ spinor graph. Figure 3.5 is then used to reduce the final 2-γ spinor graph to a Kronecker delta

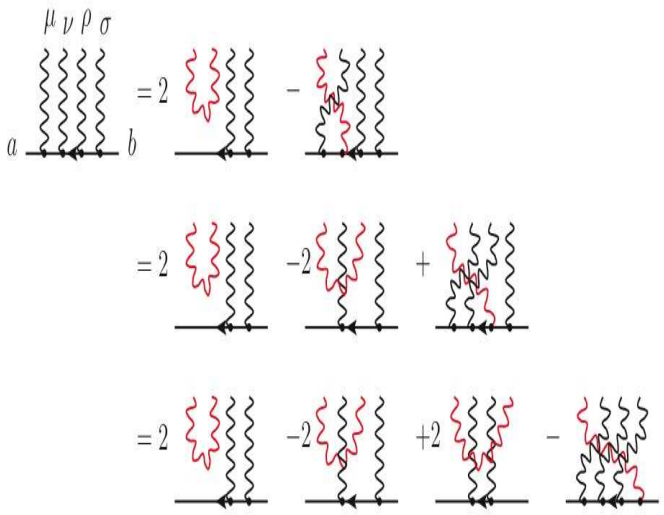


Figure 3.6: Clifford algebra applied to 4 spinors

times a trace which is equal to the dimension of the space-time.

The spinor trace routine in this code <algorithm 14>, does the above procedure by using two routines, **SpinorTrace** and **pairprime**. SpinorTrace takes a list of *indexPair*s and returns 0 if the list has an odd number of elements; returns the $2 * omega$, the dimension of space-time, if the list is empty as this corresponds to the trace of unity and calls the pairprime routine on anything else. Pairprime takes three arguments, the first corresponds to the γ line which is being commuted through the graph, the second is a list of indexPairs which represent the γ lines which have yet to be passed through and the third is a list of indexPairs which corresponds to the γ lines which have already been passed through. If the list of γ lines to be passed through is empty, it returns 0 otherwise the routine constructs a Kronecker delta from the indexPair of the commuting γ line and the first indexpair in the list of the γ lines to be passed through and calls SpinorTrace on the list of remaining indexPairs. This is the construction of the Kronecker delta times the $n-2$ spinor graph from figure 3.6. Pairprime also calls itself with a negative sign attached with the first indexPair in the list of γ lines to be passed through moved to the list of indexPairs of γ lines that have been passed through. This is the crossed graph in figure 3.6.

To help illustrate this, the 4-γ spinor graph is gone through explicitly below.

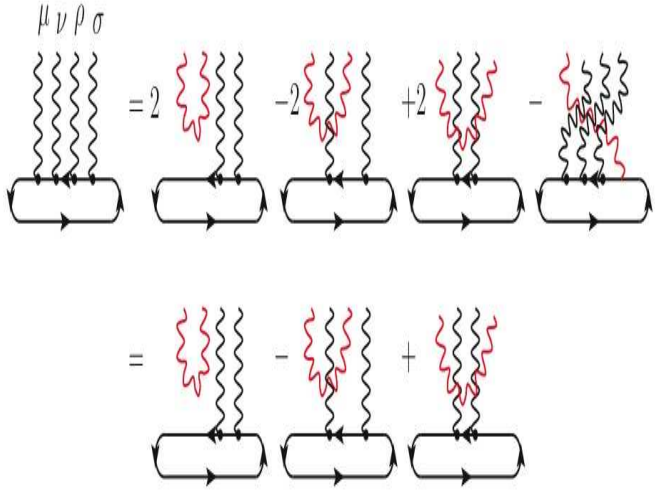


Figure 3.7: Cyclic property of traces

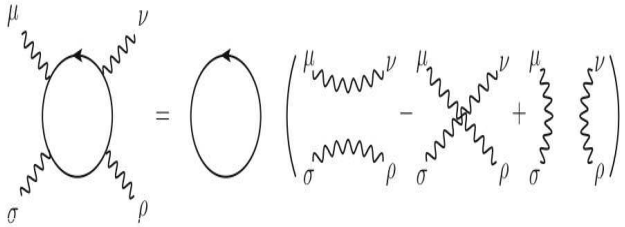


Figure 3.8: Trace of 4 spinors

$$\begin{aligned}&\textbf{SpinorTrace}\left([\mu,\nu,\rho,\sigma]\right)\\&\quad\rightarrow \textbf{pairprime}\left(\mu,[\nu,\rho,\sigma],[]\right)\\&\quad\rightarrow \delta_{\mu\nu}\times\textbf{SpinorTrace}\left([\rho,\sigma]\right)-\textbf{pairprime}\left(\mu,[\rho,\sigma],[\nu]\right)\end{aligned}\tag{3.5}$$

Algorithm 14 Spinor Tracing Routines

```
SpinorTrace := proc(x :: list)localtrone;
trone := −2 ∗ omega;
if type(nops(x), odd) then
  0
else if x = [] then
  trone
else
  pairprime(op(1, x), [op(2.. − 1, x)], [])
end if
endproc :
pairprime := proc(a, x, y)
if x = [] then
  0
else
  KroneckerDelta((op(1, a), op(2, a)), (op(1, op(1, x)), op(2, op(1, x)))) ∗
  SpinorTrace([op(y), op(2.. − 1, x)]) − pairprime(a, [op(2.. − 1, x)], [op(y), op(1, x)])
end if
endproc :
```

$$\begin{aligned}&\mathbf{SpinorTrace}\,([\rho,\sigma])\\&\quad\rightarrow \mathbf{pairprime}\,(\rho,[\sigma],[])\\&\quad\rightarrow \delta_{\rho\sigma}\times\mathbf{SpinorTrace}\,([])-\mathbf{pairprime}\,(\rho,[],[\sigma])\\&\quad\rightarrow \delta_{\rho\sigma}\times tr\mathbf{1}\end{aligned}\tag{3.6}$$

$$\begin{aligned}&\mathbf{pairprime}\,(\mu,[\rho,\sigma],[\nu])\\&\quad\rightarrow \delta_{\mu\rho}\times\mathbf{SpinorTrace}\,([\mu,\sigma])-\mathbf{pairprime}\,(\mu,[\sigma],[\nu,\rho])\end{aligned}\tag{3.7}$$

$$\begin{aligned}&\mathbf{SpinorTrace}\,([\nu,\sigma])\\&\quad\rightarrow \mathbf{pairprime}\,(\nu,[\sigma],[])\\&\quad\rightarrow \delta_{\nu\sigma}\times\mathbf{SpinorTrace}\,([])-\mathbf{pairprime}\,(\nu,[],[\sigma])\\&\quad\rightarrow \delta_{\nu\sigma}\times tr\mathbf{1}-0\end{aligned}\tag{3.8}$$

$$\begin{aligned}
&\textbf{pairprime}\,(\mu,[\sigma],[\nu,\rho]) \\
&\quad\rightarrow \delta_{\mu\sigma}\times\textbf{SpinorTrace}\,([\nu,\rho])-\textbf{pairprime}\,(\mu,[],[\nu,\sigma,\rho]) \\
&\quad\rightarrow \delta_{\mu\sigma}\times\textbf{SpinorTrace}\,([\nu,\rho])-0
\end{aligned} \tag{3.9}$$

$$\begin{aligned}
&\textbf{SpinorTrace}\,([\nu,\rho]) \\
&\quad\rightarrow \textbf{pairprime}\,(\nu,[\rho],[]) \\
&\quad\rightarrow \delta_{\nu\rho}\times\textbf{SpinorTrace}\,([])-\textbf{pairprime}\,(\nu,[],[\sigma]) \\
&\quad\rightarrow \delta_{\nu\rho}\times tr\mathbf{1}-0
\end{aligned} \tag{3.10}$$

$$\rightarrow \delta_{\mu\sigma}\delta_{\nu\rho}\times tr\mathbf{1}$$

$$\rightarrow \delta_{\mu\rho}\delta_{\nu\sigma}tr\mathbf{1}-\delta_{\mu\sigma}\delta_{\nu\rho}tr\mathbf{1}$$

$$\rightarrow tr\mathbf{1}\left(\delta_{\mu\nu}\delta_{\rho\sigma}-\delta_{\mu\rho}\delta_{\nu\sigma}+\delta_{\mu\sigma}\delta_{\nu\rho}\right)$$

There are a number of different algorithms for the calculation of spinor traces which calculate them in a cleverer and more efficient way [88, 36] . However for the purposes of this code, the current algorithm is sufficient as other bottlenecks will occur in the code long before this algorithm becomes problematic.

3.6 Forestry

This module takes the list of graph pseudofunctions generated by the **analyzeDiagram** module which correspond to the full dimensionally regulated integral of the Feynman diagram in question and applies the BPHZ renormalization procedure. It returns the original Feynman integral with all the necessary local subtractions to produce a finite integral.

This module is split into two main sections: **Subtraction Operations** and **The R Operation**.

The R Operation section provides a Henge decomposition of the Feynman Integrals and applies the Kennedy-Caswell R-Operation to these Henged integrals. As the R-Operation is done in momentum-space the natural choice of the subtraction operation is the Taylor series subtraction. This is done in the Subtraction Operations section of the code along with routines for the naïve power counting of the graphs and the

necessary differentiation operators for the edges in the graph pseudofunction.

3.6.1 Subtraction Operations

This section is split into 4 subsections which provide the naïve power counting, differentiation operators and Taylor series subtraction. The elMom section provides routines for the elimination of momentum variables in a completely arbitrary manner.

Power Counting degree: degGraph

The naïve power-counting degree of a (sub)graph is computed by the procedure. **degGraph**:

deGraph is simply the formula:

$$\deg(\mathcal{G}) \equiv 2\omega L - \sum_{i=1}^{n} deg(edge[i])$$

where n is the number of edges in the edge table of the graph pseudofunction. $\deg(edge[i])$ is calculated by taking the denominator degree which is stored in the edge and adding to this the number of vector indices in the numerator -1 which removes the contribution that the maximum vector index will make to the numerator degree.

It should be noted, that this algorithm is only valid whilst the integral is in the momentum-space representation and alternatives will be needed once the Fenman parameterization is done.

Differentiate graph with respect to external momenta: dGraph

As the Taylor series subtraction operator **T** requires the derivatives of the edges in a graph with respect to the external momenta times some necessary powers of the external momenta in the numerator, it is convenient to do both of these operations in the same procedure. In order to do this, **dGraph** takes a graph pseudofunction and differentiates the internal edges with respect to the external momenta of the graph and inserts the appropriate powers of the external momenta required by Taylor's theorem into the numerator of the external edges. The second argument to **dGraph** is just a number which is multiplied into the overall coefficient of each graph generated, this is just for convenience in introducing the negative sign needed to convert the Taylor series into a subtraction and the factors of $(d!)^{-1}$ that occur in Taylor's theorem. To help understand how **dGraph** works consider the graph with internal legs $\Delta(p_1)\Delta(p_2)\Delta(p_3)$ and external legs $\Delta(q_1)\Delta(q_2)$. **dGraph** acting on such a graph returns a sequence of graphs with internal legs of the form $\left(\frac{d}{dp_{1,\mu}}\Delta(p_1)\right)\Delta(p_2)\Delta(p_3)$ and external legs of the form $\Delta(q_1)\,q_{1,\nu}\Delta(q_2)$ tied together with a tensor $\delta_{\mu,\nu}$. The derivative of a propagator with respect to an external momentum is always either zero or $\pm$ 1 as the line

momenta are Z/(3) linear combinations of each other; such derivatives could in general be obtained from a lookup table for the Feynman rules, but it shall be assumed for the present that all propagators $\Delta(p)$ just have a quadratic form in p in the denominator and a product of vectors such as $p_{\mu_1}p_{\mu_2}p_{\mu_3}$ in the numerator. The momentum vectors corresponding to any line momentum are always kept in the numerator of the corresponding line, and the tensor indices correspond to the pair of the lineName and an integer in the **NumeratorMomenta** list; this is why the external momentum q_1 is associated with the external leg with propagator $\Delta(q_1)$ in our simple example. It is important to note that **dGraph** does *not* set the external momenta to zero in the internal lines: in this sense it really is just a differentiation operator. Also note that **dGraph** differentiates *all* the internal lines with respect to *all* the external lines, so in the example the output would be a sequence of 6 graphs if those that vanished were not omitted. Of course **dGraph** works when given derivatives of propagators in its input too, so it can be used to compute multiple derivatives. A possible future optimization would be to avoid computing mixed second derivatives such as $\frac{d^2}{dp_1 dp_2}g = \frac{d^2}{dp_2 dp_1}g$ twice.

The graph coefficient, KTensor, edge table, external linename list and internal linename list are assigned to local variables to avoid having to repeatedly call the look-up tables. The factor from the Taylor series subtraction is multiplied into the overall graph coefficient to make a new graph coefficent.

For each external **momentumName** a copy of the edge table is made to create a unique edge table for every graph pseudofunction differentiated by that external momentName. Into this edge table, a new external edges is inserted with an additional numerator vector index, replacing the exisiting external edge of the same name.

Algorithm 15 External momenta insertion into external edge numerator

$ee := edges[eln]$;
$eMom, eDd, eN := op(ee)$;
$eNd := op(1, eN)$;
$newEN := NumeratorMomenta(eNd + 1, op(2.. - 1, eN), eNd + 1)$;
$newEe := subsop(3 = newEN, ee)$;
$newEEdges := setEdge(edges, eln, newEe)$;

The denominator momentum argument is then differentiated with respect to the external momentName, creating an element of $Z/(3)$ <algorithm 16>. If anything else is returned an error is produced.

Providing a $Z/(3)$ value of ± 1 is produced by the differentiation of the momentum variable in the edge, the rest of the edge is then modified to take into account the differentiation of the denominator of the propagator <algorithm 17>. This is done by increasing the maximum vector index in the numerator by 1 and adding this new maximum index to the list of the vector indices held in the numerator momenta data

Algorithm 16 Denominator momenta differentiation

$ie := edges[iln]$;
$iMom, iDd, iN := op(ie)$;
$z := diff(iMom, eMom)$;
if $abs(z) > 1$ **then**
  ERROR("Bad momentum derivative")
else if $z \neq 0$ **then**
  call ⟨algorithm 17⟩
  call ⟨algorithm 18⟩
end if

structure. The denominator power is decreased by 2 and the old denominator power along with the $Z/(3)$ factor are multiplied to the existing overall graph coefficient c. A Kronecker delta function which ties the differentiated linename with its maximum vector index and the external linename with its maximum vector index which is doing the differentiation is then added into the $KTensor$ structure. Finally these modified data structures are substituted into the graph pseudofunction and added to a sequence of graph pseudofunctions. It should be noted that the act of differentiation does not change the subgraph structure (i.e. the topology).

Algorithm 17 Denominator differentiation

$iNd := op(1, iN)$;
$newIN := NumeratorMomenta(iNd + 1, op(2.. - 1, iN), iNd + 1)$;
$newIe := subsop(2 = iDd - 2, 3 = newIN, ie)$;
$newC := c * iDd * z$;
$newT := insertKDelta([KroneckerDelta(iln, iNd + 1, eln, eNd + 1)], t)$;
$newG := subsop(1 = newC, 2 = newT, 3 = setEdge(newEEdges, iln, newIe), g)$;
$res := res, newG$;

The numerator momentum vectors are then differentiated (if any exist) <algorithm 18> . This achieved by removing one of the vector indices from the numerator momenta data structure and inserting a Kronecker delta function which ties the affected internal linename and removed vector index with the external linename with its maximum vector index which is performing the differentiation. The graph coefficient is updated to include the $Z/(3)$ factor and all of these changes are substituted into a graph pseudofunction which forms an element of a sequence of such pseudofunctions.

This process is then repeated for all necessary internal edges and all external momenta.

Part of the output of the procedure is given below.

```
dGraph: Differentiating graph
graph(-2*delta[I7[1], E4[1]], ct,
```

Algorithm 18 Numerator differentiation

for k from 2 to $nops(iN)$ **do**
$newIN := subsop(k = NULL, iN)$;
$newIe := subsop(3 = newIN, ie)$;
$newC := c * z$;
$newT := insertKDelta([KroneckerDelta(iln, op(k, iN), eln, eNd + 1)], t)$;
$newG := subsop(1 = newC, 2 = newT, 3 = setEdge(newEEdges, iln, newIe), g)$;
$res := res, newG$;
end for

```
  [E4, E5], [I4, I5, I6, I7, I8], "diagram_diag2", I5,

  GR(TSUB(I7, I8, TSUB(I5, I4, I6))))
dGraph: Differentiating line I4 with momentum k1 w.r.t. line E4
    with momentum QE4
dGraph: Differentiating line I5 with momentum k1 w.r.t. line E4
    with momentum QE4
dGraph: Differentiating line I6 with momentum k1 w.r.t. line E4
    with momentum QE4
dGraph: Differentiating line I7 with momentum QE4+K1 w.r.t. line E4
    with momentum QE4
dGraph: Differentiating denominator
  graph(4*delta[I7[2], E4[2]]*delta[I7[1], E4[1]], ct,

    [E4, E5], [I4, I5, I6, I7, I8], "diagram_diag2", I5,

    GR(TSUB(I7, I8, TSUB(I5, I4, I6))))
dGraph: Differentiating numerator
  graph(-1-delta[I7[1], E4[2]]*delta[I7[1], E4[1]], ct,

    [E4, E5], [I4, I5, I6, I7, I8], "diagram_diag2", I5,

    GR(TSUB(I7, I8, TSUB(I5, I4, I6))))
dGraph: Differentiating line I8 with momentum K1 w.r.t. line E4
    with momentum QE4
```

with edge tables for each of the line names being:

$$\begin{aligned} PROP(E4) &= \frac{QE4_1}{(QE4^2+m^2)}, PROP(E5) = \frac{1}{(QE4^2+m^2)}, PROP(I8) = \frac{1}{(K1^2+m^2)}, \\ PROP(I4) &= \frac{1}{(k1^2+m^2)}, PROP(I5) = \frac{1}{(k1^2+m^2)}, PROP(I6) = \frac{1}{(k1^2+m^2)}, \\ PROP(I7) &= \frac{(QE4+K1)_1}{((QE4+K1)^2+m^2)^2} \end{aligned}$$

$$\begin{aligned} PROP(E4) &= \frac{QE4_1 * QE4_2}{(QE4^2+m^2)}, PROP(E5) = \frac{1}{(QE4^2+m^2)}, PROP(I8) = \frac{1}{(K1^2+m^2)}, \\ PROP(I4) &= \frac{1}{(k1^2+m^2)}, PROP(I5) = \frac{1}{(k1^2+m^2)}, PROP(I6) = \frac{1}{(k1^2+m^2)}, \\ PROP(I7) &= \frac{(QE4+K1)_1 * (QE4+K1)_2}{((QE4+K1)^2+m^2)^3} \end{aligned}$$

$$\begin{aligned} PROP(E4) &= \frac{QE4_1 * QE4_2}{(QE4^2+m^2)}, PROP(E5) = \frac{1}{(QE4^2+m^2)}, PROP(I8) = \frac{1}{(K1^2+m^2)}, \\ PROP(I4) &= \frac{1}{(k1^2+m^2)}, PROP(I5) = \frac{1}{(k1^2+m^2)}, PROP(I6) = \frac{1}{(k1^2+m^2)}, \\ PROP(I7) &= \frac{1}{((QE4+K1)^2+m^2)^2} \end{aligned}$$

elMom

The purpose of the **elMom** procedure is to expresses all the momenta in the graph in terms of the momenta entering the graph from the external lines and some arbitrarily chosen loop momenta.

This is achieved by taking a list of internal linenames and external linenames and using the Galler-Fischer algorithm to provide a momentum routing through this graph, which is labelled differently to the momentum of the main graph. Due to the recursive nature of the R-Operation, an additional set of linenames is needed which list the linenames of an already subtracted subgraph.

The subtracted internal linenames, sie, use the ancestor function and a basic Galler-Fischer algorithm to find the connected components for the sie's.

This then allows the sie's and associated vertices to be removed from the list of internal linenames, so that the connected components, associated routes and loops can be found for the unsubtracted internal edges. The external momentum routing is then performed in a similar way to the analyzeDiagram analogue <algorithm 8>. The main difference being the concatenation of the letter Q on to the front of any external linename which forms the name of each external line momenta, and the setting of the

momentum sequence of equalities to be that the external linename is equal to the − of the new external line momenta.

Like with the external momenta routing, the loop momenta routing is very similar to that of the loop momenta routing done in the analyzeDiagram procedure. For each loop found in the Galler-Fischer algorithm a new loop momenta variable is assigned which is designated as K as opposed to k for loop momenta in the main graph and the numerical value associated with it is assigned with the use of a counter *knum*, which assigns a value of +1 to the maximum K value for each additional loop. Once the loop momenta has been assigned, this is combined with external momenta routing and a sequence of equalities are returned to the **T** procedure which equate unsubtracted and external linenames to the momenta running through them.

An example of how **elMom** works is given below:

```
elMom: number of loops found = 2
elMom: Routing momentum QE4 for external line E4 from
    V4 to V6 through I7+I4 giving QE4*(I7+I4)
elMom: Routing momentum -QE4 for external line E5 from
    V6 to V6 through 0 giving 0
momentumName: New loop momentum name knum = 1
elMom: Routing momentum K1 through loop I6+I5+I4
    giving K1*(I6+I5+I4)
momentumName: New loop momentum name knum = 2
elMom: Routing momentum K2 through loop I8+I7+I5+I4
    giving K2*(I8+I7+I5+I4)
elMom: Momentum in line I4 is QE4+K1+K2
elMom: Momentum in line I5 is K1+K2
elMom: Momentum in line I6 is K1
elMom: Momentum in line I7 is QE4+K2
elMom: Momentum in line I8 is K2
```

Taylor subtraction operation: T

The Taylor series subtraction operation T acts on a graph sent from the R-Operation algorithms and returns all the subtractions needed to render it overall finite. Note that T returns minus the Taylor series, so formally it corresponds to the operation $-T^{\deg(G)}$. T first computes the overall degree; if the graph is overall divergent ($0 \leq \deg(G)$) it then expresses all the momenta in the graph in terms of the momenta entering the graph from the external lines and some arbitrarily chosen loop momenta. This is done by the elMom procedure explained in section 3.6.1. T ignores the momentum corresponding to the last external line because it is always completely constrained by momentum

conservation. The change of variables also makes it easy to generate the momentum substitution that evaluates the Taylor series at the subtraction point. At present the subtraction point is at zero external momentum, but changing the substitution zMom to have some subtraction point p_{sub} should be all that is needed to generalise this. Finally T uses dGraph to compute the Taylor series up to the necessary order, evaluating each term at the subtraction point and reexpressing the external legs' momenta in terms of the original (global) momentum variables.

Due to the recursive nature of the Henges and the R-Operation, a method has to be employed to keep track of edges that sit in the nested subtractions of a graph. This is done by renaming the *edge* to a sEdge or subtracted edge, so when these sEdges are encountered by the code they are ignored when performing further subtractions. In fact the **subsIEMom** procedure, <algorithm 19> , does more than just rename the edge it also maps the variables used in the subtraction procedures back to the global variables of the graph. The mapping of the variables is done by the procedure taking in the edge table of the new variables used in Taylor subtractions routines and making a copy of this table, then for each internal edge within the table the momentum is set to a local variable $newMom$. Another local variable, $subtractedNewMom$, inserts the subtraction point of the Taylor series into $newMom$. The momentum terms which have been set to zero in the edge are isolated by subtracting $subtractedNewMom$ from $newMom$, these terms are then remapped to the global variables and are subtracted from the global momentum in the original edge which was sent to the **T** procedure from the R-Operation procedures. This modified edge is then inserted into a new edge table and renamed as a $sEdge$.

To help illustrate how the remapping of the variables is done, a scrap of output from <algorithm 19> is provided below.

```
subsIEMom: newMom for line I6 is -QI5-K1
subsIEMom: subtractedNewMom for line I6 is -K1
subsIEMom: subtractionNewMom for line I6 is -QI5
subsIEMom: extMom in subsIEMom is {QI5 = -k1-k2, QI4 = p3+k1+k2, QE5 = -p3}
subsIEMom: subtractionOldMom for line I6 is k1+k2
subsIEMom: oldMom for line I6 is k1
subsIEMom: subtractedOldMom for line I6 is -k2
subsIEMom: Edge I6: 1/(k1^2+m^2) --> 1/(k2^2+m^2)
```

As the fields and anti-fields are distinguished by the direction of the edge entering a vertex, this orientation needs to be kept track of when dealing with subgraphs of some larger graph. The orientation algorithm <algorithm 20> , assigns a value of ± 1 to the

Algorithm 19 Substituted Edges

```
subsIEMom := proc(oldEdges :: table)
local e, oldMom, newEdges, newMom, subtractedOldMom, subtractedNewMom,
subtractionNewMom, subtractionOldMom, newMomSubsSeq, sEdges;
newMomSubsSeq := NULL;
newEdges := copy(oldEdges);
for e in ie do
  newMom := op(1, newEdges[e]);
  subtractedNewMom := subs(zMom, newMom);
  subtractionNewMom := newMom − subtractedNewMom;
  subtractionOldMom := subs(extMom, subtractionNewMom);
  oldMom := op(1, edges[e]);
  subtractedOldMom := oldMom − subtractionOldMom;
  newEdges[e] := subsop(0 = sEdge, 1 = subtractedOldMom, newEdges[e]);
  newMomSubsSeq := newMomSubsSeq, e = subtractedOldMom
end for
for e in ee do
  if op(0, e) = sEdge then
    error "Subtracted edge, sEdge = %1, present in external edges", op(0,e)
  end if
  newEdges[e] := subsop(1 = op(1, edges[e]), newEdges[e]);
end for
newEdges
endproc;
```

external edge of the graph to indicate its direction. The internal vertices are obtained by finding the *lineEnds* of all the internal lines in the subgraph. The lines external to the subgraph are then analyzed and if the line is going to a vertex which is part of the internal vertices, a value of 1 is given, if the line is going to a vertex which is not in the set of the internal vertices, a value of -1 is given.

Algorithm 20 External Line Orientation

orientation := **proc**(**line** :: **linename**)
local internalVertices, toEnd, inwards;
$internalVertices := map(ien- > lineEnds[ien], op(aie))$;
$toEnd := op(-1, [lineEnds[line]])$;
if $nops(select(v- > member(v, internalVertices), [lineEnds[line]])) \neq 1$ **then**
error "Line %1 does not have a single end in subgraph %2", line, aie
end if
$inwards := `if`(member(toEnd, internalVertices), 1, -1)$;
inwards
endproc;

An example of the <algorithm 20> is provided below.

```
orientation: Internal vertices are {V5, V6, V7}
orientation: Orientation of external edge E5 is 1
orientation: Internal vertices are {V5, V6, V7}
orientation: Orientation of external edge I5 is -1
orientation: Internal vertices are {V5, V6, V7}
orientation: Orientation of external edge I4 is 1
```

With the above two procedures in place, the Taylor series of the argument graph can proceed. First, the degree of divergence is found , if the degree of divergence is (strictly) less than zero an error is produced as the graph is convergent and **T** should not have been called on it. For $\deg(\mathcal{G}) \geq 0$ the Taylor series is then calculated. It is worth noting that the Forestry procedure does not know about symmetric integration and will calculate linear terms as well as constant and quadratic subtraction terms.

The momentum variables in the edge table of the argument graph are then mapped to a new set of variable which express everything in terms of the momenta of the external legs <algorithm 21>. The components of the edges are saved to some local variable and the edges which form part of some nested subtraction are removed from the list of internal edges, this list along with the list of *sEdge* linenames and external edge linename are sent to the **elMom** procedure to express the momentum variables in terms of the new external momentum variables. The edge list is then copied and the modified edges are then inserted into the copy, this is needed as the same edge list can

be involved in mutltiple derivatives and is also need to re-express the new momentum variables in terms of the global momentum variables.

Algorithm 21 Change of Momentum variables

$edges, ee, aie := op(3..5, g)$;
$sie, ie := selectremove(e- > op(0, edges[e]) = sEdge, aie)$;
$newMomSubs := elMom(sie, ie, ee)$;
$extMom := NULL$;
for e in ee **do**
  $newMom := convert(cat("Q", e), ' symbol')$;
  $oldMom := op(1, edges[e]) * orientation(e)$;
  $extMom := extMom, newMom = oldMom$
end for
$extMom := extMom$;
for eq in $newMomSubs$ **do**
  $momu[lhs(eq)] := rhs(eq)$
end for
$newEdges := copy(edges)$;
for $eName$ in $\{op(ie), op(ee)\}$ **do**
  $e := newEdges[eName]$;
  $se := subsop(1 = momu[eName], e)$;
  $newEdges[eName] := se$;
end for

At this moment only zero momentum subtraction point implemented, which makes a copy of the set of new external momentum variables and sets them to zero

$zMom := map(e- > lhs(e) = 0, extMom)$;

The Taylor series itself is calculated by forming a sequence of graph pseudofunctions <algorithm 22> which sets the modified copy of the edge table to − the original edge table and calls the **subsIEMom** routine <algorithm 19> to map the momentum variables to the global momentum variables. This forms the constant subtraction term of the graph pseudofunction, **dGraph** is then called for the higher order term in the Taylor series with the appropriate factor of $1/d!$. **dGraph** returns a sequence of edge tables which are added into the sequence of graph pseudofunctions which is returned to the R-Operation procedures.

An example of the output form T is given below.

```
T: Taylor subtracting 1 loop graph Graph(EXT(E5,I5,I4),INT(I6,I7,I8))
    of degree 0
T: Subtraction term is TSUB(I6,I7,I8)
T: subtracted internal edges are [] and the remaining internal Edges
    are [I6, I7, I8]
T: Global external & loop momenta are
```

Algorithm 22 Taylor Series computation

$drv := subsop(1 = -op(1, g), 3 = newEdges, g)$;
$res := subsop(3 = subsIEMom(newEdges), drv)$;
for d from 1 to deg **do**
  $drv := op(map(dGraph, [drv], 1/d))$;
  $sub := op(map(sg- > subsop(3 = subsIEMom(op(3, sg)), sg), [drv]))$;
  $res := res, sub$;
end for
res

```
    {E5 = QE5, I5 = QI5, I4 = -QE5-QI5, I8 = K1, I7 = -QE5+K1, I6 = -QI5-K1}
T: Renaming momenta external to subgraph to
   QE5 = -p3, QI5 = -k1-k2, QI4 = p3+k1+k2
T: Renaming momentum in edge E5: -p3 = QE5
T: Renaming momentum in edge I5: k1+k2 = QI5
T: Renaming momentum in edge I4: p3+k1+k2 = -QE5-QI5
T: Renaming momentum in edge I6: k1 = -QI5-K1
T: Renaming momentum in edge I7: p3+k2 = -QE5+K1
T: Renaming momentum in edge I8: k2 = K1
T: Subtraction point is at QI4 = 0, QI5 = 0, QE5 = 0

T: Constant subtraction term is
      graph(-1, KTSum(KProd(1)), newEdges, [E5, I5, I4], [I6, I7, I8],
        "diagram_diag2", I6, GR(TSUB(I6, I7, I8)))
```

with edge table:

$$\begin{aligned} PROP(E4) &= \frac{1}{(p3^2+m^2)}, PROP(E5) = \frac{1}{(p3^2+m^2)}, PROP(I8) = \frac{1}{(k2^2+m^2)}, \\ PROP(I4) &= \frac{1}{((p3+k1+k2)^2+m^2)}, PROP(I5) = \frac{1}{((k1+k2)^2+m^2)}, \\ PROP(I6) &= \frac{1}{(k2^2+m^2)}, PROP(I7) = \frac{1}{(k2^2+m^2)}. \end{aligned} \tag{3.11}$$

3.6.2 The R operation

The procedures in this section are the direct implementation of the Kennedy-Caswell Henge demposition and BPHZ definition and the associated routines to make them work.

The purpose of these procedures is to take the sequence of graph pseudofunctions generated by the **analyzeDiagram** procedure and find all local subtractions necessary to make the integral that sequence of graph pseudofunctions represent manifestly finite. This is done by performing the Henge decomposition procedure described in the last

chapter and calculating the appropriate subtraction terms using Taylor's theorem as described in the preceding sections.

A henge $\mathcal{H}$ is defined to be the set of 1PI subgraphs and singile vertices of $I(\mathcal{G})$ that remain after the removal of a selected edge $\ell \in \mathcal{G}$ from $I(\mathcal{G})$.

In order for the Henge decomposition procedure to work, several routines which analyze the topology of the graph in question:

- **connectedComponents;**
- **countLoops;**
- **oneParticleIrreducibleComponents;**
- **Henges;**

The only difference between the **connectedComponents;**, **countLoops;** and the **oneParticleIrreducibleComponents;** procedures in the R-Operation module and that of the analyze diagram module is that they have been constructed to only need to take a list of internal line names as all other necessary information for the edges associated with the line names are already constructed in the dynamic data structures of the code.

The final two subroutines needed, manipulate the graph pseudofunctions so that individual subgraphs can be analyzed:

- **embed**
- **debme**

.

Henges

To construct the henges of the input graph G that is specified by the set of lines **iLines**, it finds the IPI components of G minus the line i (the set of lists of lines representing $G - \{i\}$ are contained in the local variable *subLines*). Because the collection of henges is represented as a **set**, duplicate henges are automatically removed <algorithm 23>. The result is returned as a sequence of Henges.

For the standard ϕ^3 2-loop example, one would find:

```
henges: Henges of [I4, I5, I6, I7, I8] are
    [[I4, I5, I6]], [[I4, I5, I7, I8]], [[I6, I7, I8]]
henges: Henges of [I4, I5, I6] are []
```

Algorithm 23 Make Henges

henges := **proc**(**iLines** :: **list**(**linename**))
local i, p, res, subLines;
$p := iLine- > remove(unapply(i = iLine, i), iLines)$;
$subLines := map(p, \{op(iLines)\})$;
$res := op(map(oneParticleIrreducibleComponents, subLines))$;
res
endproc :

embed

When the graphs contain ≥ 2 loops, they will contain subgraphs which need to have local subtractions done in order to make the graph finite. However, as the graph pseudofunction contains all the information about the graph, the **embed** routine isolates the $1PI$ subgraphs, specified by the $iNames$, of the graph and modifies the graph pseudofunction to refer to this subgraph This just involves replacing the lists of linenames specifying the subgraph structure. As the graph pseudofunction is now referring to the subgraph, the external lines have been changed. To find the new set of edges external to the subgraph, the set of vertices in the subgraph have to be found. Once this done, the set of lines ending at these vertices can be found, and the internal lines from this set can be removed <algorithm 24>.

Algorithm 24 Subgraph Embedding

embed := **proc**(**iNames** :: **list**(**linename**), **gg** :: **graph**)
local eL, eNames, g, iL, v, vertices, embVertices;
$vertices := \{op(map(e- > lineEnds[e], iNames))\}$;
$eNames := [op(map(v- > op(map(u- > vertexLineEnds[v, u],$
$(\{indices(vertexLineEnds, nolist)\}$
minus $select(type, \{indices(vertexLineEnds, nolist)\}, ' name'))))), vertices)$
minus $\{op(iNames)\})]$;
$g := subsop(4 = eNames, 5 = iNames, gg)$;
g
endproc :

To show how this works, part of the output for the standard 2-loop ϕ^3 graph figure 3.1 is shown below along with a graphical representation of the the modification to the graph pseudofunction:

```
embed: slots consists of {1, 2, 3}
embed: Embedding [I4, I5, I6] in Graph(EXT(E4,E5),INT(I4,I5,I6,I7,I8))
    ==> Graph(EXT(E4,I7,I8),INT(I4,I5,I6))
```

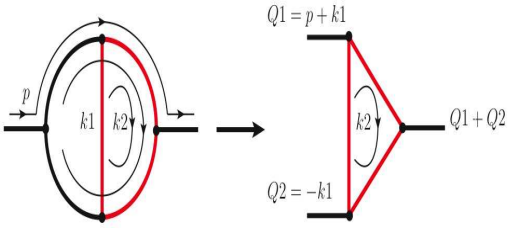


Figure 3.9: Illustrative example of how the embed procedure works. Note the embed procedure does not change the momentum variable names, this is done here for clarity.

debme

The **debme** procedure is the inverse of **embed**, it inserts the subgraph **g** into its parent **gp**. As the subgraph structure is entirely encoded in the graph pseudofunction by the list of linenames of its internal and external lines this just involves putting these lists from **gp** back into **g**. All the subtractions that may have been made are encoded in the overall coefficient, tensor structure, and edge momenta in **g** that are left unchanged <algorithm 25>.

Algorithm 25 Subgraph Re-insertion into Graph

debme := **proc**(**g** :: **graph**, **gp** :: **graph**)
$subsop(4 = op(4, gp), 5 = op(5, gp), g)$;
endproc :

Now all the technology needed to perform the R-Operation is present, all that is left is to apply them in the correct way. As a reminder, the formula for the R and the $\bar{R}$ operation is presented here:

$$RI_\lambda(\mathcal{G}) \equiv \bar{R}I_\lambda(\mathcal{G}) - K\bar{R}I_0(\mathcal{G}) \tag{3.12}$$

$$\bar{R}I_\lambda(\mathcal{G}) = \sum_{\ell \in \mathcal{G}} \int_\lambda^\infty dki_k \left(\mathcal{G}/\mathcal{H}(\mathcal{G}, \ell)\right) \prod_{\Theta \in \mathcal{H}(\mathcal{G},\ell)} RI_k(\Theta) \tag{3.13}$$

As the subtraction operation K is the Taylor series subtraction, all that is left to do is to define the R, $\bar{R}$-Operations in the code and to have a way of applying R to each of the disjoint $1PI$ subgraphs in each henge, this is done by the **RHenge** algorithm.

RHenge

RHenge takes a graph pseudofunction and a list of linenames which correspond to the internal edges of the graph pseudofunction that are contained within a henge which

in turn contain the information about $1PI$ subgraphs of that henge. The linenames not contained in the henge are then obtained and the line with the smallest momenta is found. If the smallest line from the list of lines not in the henge is smaller than the minimum line of the graph, a value of $NULL$ as the henge process will generate a subtraction term for each henge H for every ordering of the momenta in G/H, the code only generate the subtraction once (for all momenta external to H) by only doing so for the smallest line in G/H. The ordering is arbitrary. If the smallest of the linenames not in the henge is larger than the minimum line of the graph then the smallest linename of the linenames not in the henge is made the smallest line for the graph. The reason for this is that, from the previous chapter, the henge decomposition procedure requires that the lines within the Henge always carry a larger momentum value than the line which defines the henge. As the henges can be nested the line which defines the original henge has to be smaller than the lines in the henge it defines, which in turn contains a line which defines another henge which has to have be smaller than the ones in the henge it defines.

For each of the $1PI$ graphs in the henge the **embed** algorithm is called <algorithm 24> to form a new graph pseudofunction which the R-Operation is applied to. The result of R applied to the embedded subgraph is the put back into the main graph with **debme** <algorithm 25>. Finally the debug information which contains the nested subtractions of the graph is updated.

Algorithm 26 Application of the R Operation to the Henges

RHenge := **proc**(**g** :: **graph**, **h** :: **list**(**list**(**linename**)))
local sloop, minLine, hengesInOrder, res, theta;
$sloop := \{op(op(5, g))\} minus \{op(map(op, h))\}$;
$minLine := linenameMin([op(sloop)])$;
$hengesInOrder := linenameLeq(op(7, g), minLine)$;
if *hengesInOrder* **then**
  $res := subsop(7 = minLine, g)$;
  for *theta* in h **do**
    $res := op(map(gr- > op(map(debme, [R(embed(theta, gr))], gr)), [res]))$
  end for
  $op(map(gr- > subsop(8 = GR(op(sloopunion\{op(op(8, gr))\})), gr), [res]))$
else
  $NULL$
end if
endproc :

Part of the output from figure 3.1.

```
RHenge: Applying R to henge [[I4, I5, I6]]
RHenge: Lines in G/H are {I7, I8}
RHenge: Smallest line in G/H is I7, maximin = I0, hengesInOrder is true
```

```
RHenge: Applying R to 1PI subgraph [I4, I5, I6] of henge [[I4, I5, I6]]
RHenge: Applying R to henge []
RHenge: Lines in G/H are {I5, I4, I6}
RHenge: Smallest line in G/H is I5, maximin = I7, hengesInOrder is false
RHenge: Applying R to henge []
RHenge: Lines in G/H are {I5, I4, I6}
RHenge: Smallest line in G/H is I5, maximin = I0, hengesInOrder is true
```

Rbar

Now that the method for applying R to the $1PI$ graphs in each of the henges **RHenge** <algorithm 26> is defined, $\bar{R}$ is then defined to be simply the application of **RHenge** to every henge <algorithm 27>.

Algorithm 27 R-Bar Operation

Rbar := **proc**(**g** :: **graph**)
$op(map(hng- > RHenge(g, hng), [henges(op(5, g))]))$
endproc :

R

The **analyzeDiagram** produces a list of graph pseudofunctions and **R1** applies R to each of the graphs in that list.

```
R1 := proc (g::list(graph))
  op(map(R, g))
end proc:
```

The R-Operation <algorithm 28> itself is simply the Kennedy-Caswell definition R.

$$RI_\lambda(\mathcal{G}) \equiv \bar{R}I_\lambda(\mathcal{G}) - K\bar{R}I_0(\mathcal{G}) \tag{3.14}$$

It provides an error check to ensure that graph pseudofunction it is analyzing is the same as the diagram analyzed by the **analyzeDiagram** procedure. Once this check has passed, it calls $\bar{R}$ on the graph pseudofunction ($\bar{R}I_\lambda(\mathcal{G})$) and calls the Taylor series subtraction routines on $\bar{R}$ of the graph with the minimum line set to $I0$ if the degree of divergence of the graph is ≥ 0, else it returns the value of $NULL$.

R returns a sequence of graph pseudofunctions which contain the graph from **analyzeDiagram** and all appropriate subtractions.

Part of the output of **R** is shown below for the example figure 3.1:

Algorithm 28 R Operation

R := **proc**(**g** :: **graph**)
local gg, subp, unsubp, TT;
if $op(6, g) \neq diagHash$ **then**
ERROR(cat("Graph is not current diagram: ","rerun analyzeDiagram"))
end if
$unsubp := Rbar(g)$;
if $degGraph(g) \geq 0$ **then**
$TT := gr- > T(subsop(8 = GR(TSUB(op(op(8, gr)))), gr))$;
$gg := subsop(7 = I0, g)$;
$subp := op(map(TT, [Rbar(gg)]))$
else
$subp := NULL$;
end if
$unsubp, subp$
endproc :

```
R: Applying R to IPI graph Graph(EXT(E4,E5),INT(I4,I5,I6,I7,I8))
R: Evaluating unsubtracted part of Graph(EXT(E4,E5),INT(I4,I5,I6,I7,I8))
R: The degree of the graph is 2
R: Applying R to IPI graph Graph(EXT(E4,I7,I8),INT(I4,I5,I6))
R: Evaluating unsubtracted part of Graph(EXT(E4,I7,I8),INT(I4,I5,I6))
R: The degree of the graph is 0
R: Graph(EXT(E4,I7,I8),INT(I4,I5,I6)) has overall degree 0
R: Evaluating subtracted part of Graph(EXT(E4,I7,I8),INT(I4,I5,I6))
R: Parts of Graph(EXT(E4,I7,I8),INT(I4,I5,I6)) without overall subtraction
    are [], and parts with overall subtraction are
        [-Graph(EXT(E4,I7,I8),INT(I4,I5,I6))]
```

3.7 Parametric Form

Up until this point, the code has strictly worked in the momentum-space representation of the Feynman Graph. In this module of the code, the graph pseudofunction used so far is decomposed into its constituent parts and from that information, Feynman parameters are introduced to the internal lines of the graph and the Feynman parametric representation denominator is constructed along the lines of the matrix form introduced in the chapter 3 section on Feynman Parameters. The numerator of the graph is formed into a Feynman parametric graph, or **FPGraph**, this contains the overall graph coefficient, the KTensor stucture and the list of momentum vector indices. The FPGraph has its momentum shifted, in accordance with the method for converting to a purely parametric form of the integral, and is subject to the constraints of symmet-

ric integration so terms with odd numbers of momentum vector indices are dropped. The surviving terms are then symmetrized on the momentum vector indices which produces pairs of coefficients and KTensor type constructs. Finally the momentum space integration is performed and the program produces either the C-code of the remaining parametric integration or Maple code of the same thing. The Maple output is only useful for the debugging and performing the integration of the simplest diagrams and in general will should be ignored. The output of the graph contains only powers of Feynman parameters, mass terms, momentum variables some constant terms.

3.7.1 doubleFactorial

The **doubleFactorial** is needed as part of the evaluation of the momentum space integrals and is defined to be:

$$n!! = \begin{cases} n \cdot (n-2) \ldots 5 \cdot 3 \cdot 1 & \text{if } n > 0 \text{ and is odd;} \\ n \cdot (n-2) \ldots 6 \cdot 4 \cdot 2 & \text{if } n > 0 \text{ and is even;} \\ 1 & \text{if } n = 0 \text{ or } -1. \end{cases}$$

The procedure **doubleFactorial** is just the direct implementation of the above definition.

3.7.2 FP

FP constructs the Feynman parameter integral corresponding to graph g. The quadratic form occuring in the denominator is represented as $Q(k,p,m) = k.M.k - (2p.B).k + p.C.p + D$ where k is the vector of loop momenta (as specified by the $loopMomenta$ set), p is the vector of external momenta (as specified by the $externalMomenta$ set), M, B, C are matrices depending only on the Feynman parameters x_i , and $D = \sum_{i=1}^{i} x_i {m_i}^2$ is a scalar function of the Feynman parameters and the masses of the particles of the corresponding propagators.

The **FP** procedure takes the sequence of graphs generated by the **R**-Operation module and calls the procedure **FP1**, which produces a Feynman parametric integrand. If the $MSIOutput$ variable is set to *'expression'*, this will generate the Feynman Parametric integrand in Maple. If $MSIOutput$ is set to $NULL$, then it creates a directory for the C routine files inside the directory specified by the *outputFileRoot*, which is specified at the start of the code and uses the **MSICodeGen** procedure to combine the Feynman parametric integrands from **FP1** and to generate the C routines.

The **FP1** procedure takes each of the graph pseudofunctions in turn and and produces a Feynman parametric integrand for that graph.

The parametric integral is a function of the external momenta, and this external momenta is stored in the KTensor structure in the FPGraph until the momentum space the KTensor structure has its final simplification before the momentum space integration.

This external momentum, like in the graph pseudofunction, takes the form of a Kronecker delta like object (described in the definitions section) and is inserted into the FPGraph with the **addExtMom** procedure <algorithm 29>. This simple routine takes the FPGraph, the desired external momentum and a coefficient which can take the value of ± 1 which is the sign of the external momenta and is added to the FPGraph coefficient.

Algorithm 29 Add External Momenta

addExtMom := **proc**(**fpg** :: **FPGraph**, **z** :: $\{-1, 1\}$, **d** :: **KroneckerDelta**)
local c, t, t1;
$c, t := op(1..2, fpg)$;
$subsop(1 = c * z, 2 = insertKDelta([d], t), fpg)$;
endproc;

The numerator loop momenta is added to the FPGraph in a similar way as the external momenta. The **addLoopMom** routine takes the FPGraph, the sign of the loop momenta, a linename, a numerator vector index and the loop number. It updates the the list of loop momenta powers in the numerator and constructs a Kronecker delta which ties together the linename with its vector index and the loop momenta with its vector index. These are then inserted into the FPGraph pseudofunction. As the code is constructing a purely Feynman parametric representation of the Feynman integral, the loop momenta has to be shifted in the form of $k = k' + M^{-1}.B.p = k' + BoM.p/det(M)$, as explained in the previous chapter, where BoM is the matrix $B/\text{adj}M$. This is achieved by first creating an FPGraph with a power of k' added to the list of numerator loop momenta powers along with the necessary changes to the graph coefficient and KTensor. Then for each numerator external momenta, a sequence (which includes the k' term) of FPGraph pseudofunctions is created where the overall graph coefficient is updated to include the sign of the momenta and the BoM matrix which is stored symbolically and the KTensor has a new external momenta associated with the linename and vector index inserted.

For each of the internal lines a Feynman parameter is introduced and labelled x followed by the linenumber with the I removed, for example the Feynman parameter for line $I101$ will be $x101$. The Feynman parameter is then entered into a sequence of all Feynman variables. A variable *lambda* is calculated which is used to to determine the degree of divergence of the Feynman Parametric integrand and a factor of

Algorithm 30 Numerator momenta shift

```
addLoopMom := proc(fpg :: FPGraph, z :: {−1, 1}, n :: linename,
      i :: posint, ki :: posint)
local c, t, t1, kpow, kpows, d, res, zp;
c, t, kpows := op(fpg);
kpow := op(ki, kpows);
kpows := subsop(ki = kpow + 1, kpows);
d := KroneckerDelta(n, i, op(ki, loopMomenta), kpow + 1);
res := FPGraph(c ∗ z, insertKDelta([d], t), kpows); # # Now add shifts: #
for pi from 1 to numExtMom do
  zp := BoM[ki, pi];
  if zp ≠ 0 then
    p := op(pi, externalMomenta);
    d := KroneckerDelta(n, i, p);
    res := res, subsop(1 = c ∗ z ∗ SymBoM(ki, pi)/detM,
          2 = insertKDelta([d], t), fpg)
  end if
end for
res
endproc;
```

$x^{-(\text{denomPower}/2-1)}/-(\text{denomPower}/2-1)!$ is included in the overall graph coefficient. This last factor comes from the formula for Feynman parameters from the last chapter:

$$\prod_{\ell=1}^{N}\frac{\Gamma(\alpha_\ell)}{Q_\ell^{\alpha_\ell}}=\Gamma\left(\sum_{\ell=1}^{N}\alpha_\ell\right)\left(\prod_{\ell=1}^{N}\int_0^1 dx_\ell\, x_\ell^{\alpha_\ell-1}\right)\delta\left(1-\sum_{k=1}^{N}x_k\right)\left[\sum_{j=1}^{N}x_jQ_j\right]^{-\sum_{k=1}^{N}\alpha_k}.$$

The Kronecker delta function is suppressed in the code and is explicitly used in the Monte-Carlo evaluation of the parametric integral.

After the each internal line has been multiplied by a Feynman parameter, it is used along with other information from the edge table to complete the denominator of the Feynman Parametric integral $Q = k.M.k - 2k.B.p + p.C.p + D$, where the symbols have the same meaning as in chapter 3. With M being the matrix of Feynman parameters which are multiplied by terms which are quadratic in the loop momenta, B being a matrix of Feynman parameters which are multiplied by terms which are linear in the loop momenta, C a matrix of Feynman parameters which are multiplied by the external momenta and D is the mass terms times the Feynman Parameters. The momentum in the edge is formally squared and the masses of the edges are explicitly introduced here. The matrices are constructed by initialising

Two additional matrices are setup: the matrix adjoint $Madj$:= LinearAlgebra[Adjoint](M);

which is defined by the relation $M^{-1} = Madj/det(M)$ and the matrix $BoM := Madj.B$, which is the matrix B multiplied by the inverse of matrix M with the determinant of M factored out ($B.M^{-1} = BoM/det(M)$).

The momentum vectors are not explicitly stored in the code as from the previous chapter it is known that these can all be factored out and will be used up in the momentum-space integration.

The numerator of the Feynman parametric integral is stored in the FPGraph, which contains the overall coefficient of the graph, its KTensor structure and a list of numerator loop momentum vectors. The list of FPGraphs is set up and then for each line in the graph and for every numerator vector index in those lines the external momenta and loop momenta along with the necessary loop momentum shift is inserted into the FPGraphs.

Symmetric integration is achieved by taking the sum of the powers of numerator loop momenta and if this sum is *odd*, return $NULL$. If the sum is *even*, it calls the **symmetrize** procedure, which takes pairs of numerator loop momenta and returns a Kronecker delta whose arguments are the pair of numerator momenta times the appropriate element from the matrix M^{-1}, the matrix of Feynman diagram variables then returns a sequence of FPGraph pseudofunctions which have been updated for each symmetrized pair.

As the **symmetrize** procedure produces a result for all possible permutations on pairs of numerator loop momenta, there will a high degree of duplication of terms. The **tensor flattening** and **tensorSimplify2** routine are invoked and the duplicates are kept track of by creating a sparse table **Kount** which is indexed by FPGraph, everytime the same entry is called it updates the entry by 1. This allows all the duplicate FPGraphs to be replaced by a single FPGraph multiplied by the value of **Kount**. It should be noted at this point, the **graph** pseudofunction is no longer being used, having been completely replaced by the **FPGraph**.

Each of FPGraph in the list of FPGraphs has its KTensor simplified for the final time by first flattening it <algorithm 3> and then using **tensorSimplify** <algorithm 4> which returns only a constant times external momenta squared (anything else returns an error). This factor of the external momenta squared is multiplied by overall graph coefficient and this is entered into a table *rho* indexed by the number of pairs of loop momenta in the numerator. The rho table along with the denominator matrices and factors are then sent to the **MSICollect** procedure, which performs the necessary momentum space integration. **FP1** then returns a sequence of parametric integrands provided by **MSICollect**.

Algorithm 31 Numerator factors collection

```
fpgl := [FPGraph(1, KTSum(KProd(1)), [0$numLoops])];
for eName in [op(extNames), op(intNames)] do
  eMom, alpha, numMom := op(1..3, edges[eName]);
  for i in subsop(1 = NULL, numMom) do
    fpgx := NULL;
    for pi from 1 to numExtMom do
      p := op(pi, externalMomenta);
      z := coeff(eMom, p, 1);
      if abs(z) = 1 then
        fpgx := fpgx, op(map(addExtMom, fpgl, z, KroneckerDelta(eName, i, p)));
      else if z ≠ 0 then
        ERROR(sprintf("%a invalid in momentum %a of edge %s", p, eMom,
        eName))
      end if
    end for
    for ki from 1 to numLoops do
      k := op(ki, loopMomenta);
      z := coeff(eMom, k, 1);
      if abs(z) = 1 then
        fpgx := fpgx, op(map(addLoopMom, fpgl, z, eName, i, ki));
      else if z ≠ 0 then
        ERROR(sprintf("%a invalid in momentum %a of edge %s", k, eMom,
        eName))
      end if
    end for
    if nops([fpgx]) > 0 then
      fpgl := [fpgx]
    end if
  end for
end for
```

Algorithm 32 Simplify Tensors and Collect Terms

```
rhoCoeff := table(sparse);
for f in fpgl do
  rho := ` + `(op(op(3, f)))/2;
  unexpandedTnsr := insertKDelta([op(2, f)], simptnsr);
  expandedTnsr := KTSumExpand(unexpandedTnsr);
  simpTnsr := ` + `(op(map(tensorSimplify, expandedTnsr)));
  rhoCoeff[rho] := rhoCoeff[rho] + op(1, f) * simpTnsr
end for
if nops(fpgl) > 0 then
  op(map(rho- > MSICollect(c * rhoCoeff[rho], [xSeq], M, B, C, D, lambda, rho,
      Madj, BoM), map(op, [indices(rhoCoeff)])))
end if
```

FP1 is called for each graph pseudofunction provided by the **R**-Operation and the output for each graph pseudofunction is combined with all the others in the **MSI-CodeGen** procedure.

3.7.3 MSI: momentum space integration

This **MSICollect** procedure <algorithm 33> implements a general finite parametric Feynman integral formula which has had the momentum-space integration already performed:

$$(1-\bar{K})I = A\frac{(-)^{hdeg}}{hdeg!}\left(\sum_{\rho=1}^{hdeg}\frac{1}{\rho}+\frac{(1-L)}{L}\ln\det M-\ln\mu^2\right), \tag{3.15}$$

where

$$\begin{aligned} A &= \frac{\Gamma(\rho+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}}{\Gamma(\lambda)}\frac{(\mu^2)^{hdeg}}{\det M^{\omega_0+\rho+hdeg}}\times \\ &\quad\times\frac{1}{(2\rho)!}\sum_{\pi\in S_{2j}}(\text{adj } M)_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(\text{adj } M)_{\ell_{\pi(2\rho-1)}\ell_{\pi(2\rho)}}\delta_{\mu_{\pi(2\rho-1)}\mu_{\pi(2\rho)}} \\ &= \frac{\pi^{2\omega_0}(2\rho-1)!!}{2^\rho\Gamma(\lambda)}\frac{(\mu^2)^{hdeg}}{\det M^{\omega_0+\rho+hdeg}}\times \\ &\quad\times\frac{1}{(2\rho)!}\sum_{\pi\in S_{2\rho}}(\text{adj } M)_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(\text{adj } M)_{\ell_{\pi(2\rho-1)}\ell_{\pi(2\rho)}}\delta_{\mu_{\pi(2\rho-1)}\mu_{\pi(2\rho)}} \end{aligned} \tag{3.16}$$

which uses the property $\Gamma(j+\frac{1}{2})=\frac{(2j-1)!!}{2^j}\pi^{\frac{1}{2}}$ and the result $\Gamma(1/2)=\pi^{\frac{1}{2}}$ to remove the explicit Γ functions.

The finite parametric integral formula (eq. 3.15) is obtained by taking the general parametric representation formula from the previous chapter:

$$\begin{aligned} \int d^{2\omega}k\frac{k_{\mu_1}\cdots k_{\mu_{2j}}}{(k^2+m^2)^\lambda} &= \frac{\Gamma(j+\frac{1}{2})\Gamma(\frac{1}{2})^{2\omega-1}\Gamma(\lambda-\omega-j)}{\Gamma(\lambda)}\frac{(\mu^2)^{\omega-\lambda+j}}{\det M^{\omega+j+d/2}} \\ &\quad\times\frac{1}{(2j)!}\sum_{\pi\in S_{2j}}(\text{adj } M)_{\ell_{\pi(1)}\ell_{\pi(2)}}\delta_{\mu_{\pi(1)}\mu_{\pi(2)}}\cdots(\text{adj } M)_{\ell_{\pi(2j-1)}\ell_{\pi(2j)}}\delta_{\mu_{\pi(2j-1)}\mu_{\pi(2j)}} \end{aligned} \tag{3.17}$$

The variable j is renamed ρ to bring it in line with the maple code. The definition of μ^2 in the Maple code is slightly different to the one presented in chapter 3 as the Maple code version has $\det M^{-1}$ factored out of μ^2. Taking these into account and using the fact that the dimension of the integral is $\omega=\omega_0+\varepsilon$, where ω_0 is the dimension of space-time. The degree of divergence of the graph becomes $d=d_0+2\varepsilon L$, where

$d_0 \equiv 2(\omega_0 L + \rho - \lambda) = 2hdeg$ and the formula above can be re-expressed into a ε dependent form :

$$I(\varepsilon) = A\Gamma(\frac{1}{2})^{2\varepsilon L}\Gamma(-hdeg - \varepsilon L)\left(\frac{\mu^2}{\det M}\right)^{\varepsilon L} \det M^{-\varepsilon}. \tag{3.18}$$

with the constant A being the same as in (eq. 3.16).

The remaining ε-dependent Γ-function is resolved with the following formula for the expansion of the Gamma function in powers of ε:

$$\Gamma(\varepsilon - n) = \frac{(-)^n}{n!}\left(\frac{1}{\varepsilon} - \gamma + \sum_{j=1}^{n}\frac{1}{j} + O(\varepsilon)\right). \tag{3.19}$$

This formula is obtained by using the recurrence relation $\Gamma(x+1) = x\Gamma(x)$ and the expansion $\Gamma(\varepsilon) = \frac{1}{\varepsilon} - \gamma + O(\varepsilon)$. Repeated use of the recurrence relation generates $\Gamma(\varepsilon) = (\varepsilon - n)\ldots(\varepsilon - 2)(\varepsilon - 1)\Gamma(\varepsilon - n)$, which with the use of $\Gamma(\varepsilon)$ and the series expansion of the $(e - i)$ terms up to$O((\Gamma(\varepsilon)$ leads to (eq. 3.19).

This relation also gives the *renormalization scheme* of the code, which is defined to be:

$$\begin{aligned} \text{if} hdeg < 0 \quad &\rightarrow \quad (1-\bar{k})I = (-hdeg-1)! * A \\ \text{otherwise} \quad &\rightarrow \quad (1-\bar{k})I = A\frac{(-)^{hdeg}}{hdeg!}\left(\sum_{\rho=1}^{hdeg}\frac{1}{\rho} + \frac{(1-L)}{L}\ln \det M - \ln \mu^2\right) \end{aligned}$$

The arguments for **MSICollect** <algorithm 33> are **c** the overall coefficient (with external momentum inner products included from the simplified tensor structure). **xList** list of the Feynman parameters present; there is also an implicit factor of $\delta(1 - \sum_i x_i)$, but as said in the previous section this remains implicit until the numerical evaluation of the Feynman parametric integrals. **M, B, C, D** parameterization of the quadratic form in the denominator $Q(k, p, m) = k.M.k - (2p.B).k + p.C.p + D$. The matrices **M, B**, and**C** are functions of the Feynman parameters only, and the scalar **D** is a function of the propagator masses only. λ, ρ the power of k^2 in the denominator and numerator respectively. **Madj, BoM** the adjoint matrix of **M** and $det(M) * B \cdot M^{-1}$ respectively (these are completely determined by **M** and **B**, and are passed as arguments just to avoid recomputing them). **hDeg** is half the overall degree of divergence of the diagram. The symmetrized numerator tensor has already been computed and included in the coefficient **c**. **Note** that kinematic constraints on the external momenta are not applied.

Once **MSICollect** has calculated the correct parametric integrand, it is stored along

with the matrices $M, B, C, D, Madj, BoM$ and the sequence of all Feynman paragraphs used $xList$ in a Hash table indexed the matrix elements (second operand) of the matrices. The Matrix elements are used as indices for the hash table because the matrices are not '=' even though they are LinearAlgebra[Equal]. At this point the μ^2 is kept symbolically as $SmuSq$ and is only explicitly calculate at the end of the program.

Algorithm 33 Momentum Space Integration

MSICollect := **proc**(**c**, **xList**, **M**, **B**, **C**, **D**, **lambda**, **rho**, **Madj**, **BoM**)
local cc, hashIndex, hDeg, muStuff, numLoops, tabVal;
global FPHashTable;
$numLoops := nops(loopMomenta)$;
$hDeg := omega * numLoops + rho - lambda$; # degGraph not valid here
$cc := c * Pi^{(}numLoops * omega) * doubleFactorial(2 * rho - 1)/(2^{r}ho * Factorial(lambda - 1))$;
if $hDeg < 0$ **then**
  $cc := cc * (-hDeg - 1)!/detM^{(}omega + rho + hDeg)$;
  $muStuff := SmuSq^{h}Deg$
else
  $dd := cc * (-1)^{h}Deg/detM^{(}omega + rho + hDeg) * add(1/j, j = 1..hDeg)/hDeg!$;
  $cc := cc * (-1)^{h}Deg/detM^{(}omega + rho + hDeg)/hDeg!$;
  $muStuff := -SmuSq^{h}Deg * ln(SmuSq)$;
  $nuStuff := SmuSq^{h}Deg$
  $loopStuff := SmuSq^{h}Deg * ln(detM) * (1 - L) * /numLoops$;
end if
$hashIndex := op(map(m- > op(2, m), [M, B, C])), D$;
$tabVal := FPHashTable[hashIndex]$;
if $[tabVal] = []$ **then**
  $FPHashTable[hashIndex] := cc * muStuff + dd * nuStuff + cc * loopStuff, xList, M, B, C, D, Madj, BoM$
else
  $FPHashTable[hashIndex] := op(1, [tabVal]) + cc * muStuff + dd * nuStuff + cc * loopStuff, op(2.. - 1, [tabVal])$
end if
endproc :

Once all the graphs for the diagram have been passed through the **FP1** procedure and have been stored in the **FPHashTable** , the **FP** procedure calls the **MSICodeGen** routine which explicitly performs the necessary matrix multiplications and calculates the $SmuSq$ terms. If the **MSIOutput** has been set to *'expression'* then **MSICodeGen** generates a Maple output which replaces all the symbolically stored matrices with the actual matrices and produces a sum of products with any commonalities factored out. Otherwise, the C-code routine generator **MakeCodeDriver** is called.

3.7.4 Symmetrize

This section takes the numerator loop momenta of the form:

$$(O^T D^{-1})_{\ell_1,\ell'_1} \cdots (O^T D^{-1})_{\ell_{2j},\ell'_{2j}} k''_{\ell'_1\mu_1} \cdots k''_{\ell'_{2j}\mu_{2j}}$$

and replaces this with the symmerized factor of:

$$\frac{1}{(2\rho)!} \sum_{\pi \in S_{2j}} (\text{adj } M)_{\ell_{\pi(1)}\ell_{\pi(2)}} \delta_{\mu_{\pi(1)}\mu_{\pi(2)}} \cdots (\text{adj } M)_{\ell_{\pi(2\rho-1)}\ell_{\pi(2\rho)}} \delta_{\mu_{\pi(2\rho-1)}\mu_{\pi(2\rho)}}$$

which occurs once the momentum space integration has been completed (the momentum space integral itself is not done explicitly, the resulting form of a general momentum space integral is used with the appropriate factors substituted in 3.7.3). The actual form of the numerator momenta is implicit in the code and only the appropriate powers of the numerator loop momenta are kept explicitly.

The **symmetrize** procedure takes a list of non-negative integers $i_1 \ldots i_L$ where L is the number of loops in the graph. The integers are the multiplicity of the loop momentum k_j in the numerator, so the numerator is of the form ${k_1}^{i_1} \ldots {k_L}^{i_l}$, or more precisely $k_{1;k_1} \ldots k_{1;k_{i_1}} \ldots k_{L;k_{L;1}} \ldots k_{L;k_{L;i_L}}$. **symmetrize** returns a sequence of pairs of coefficients and KTensors representing the symmetrized trace on these indices. The algorithm used is brute-force: all permutations of the list of index pairs is generated and the terms are collected.

In order to **symmetrize** a list loop momenta powers, the following routines are used:

The Kronecker delta function times some element of the adjoint matrix $Madj$ is done by the routine **cp** <algorithm 34>. This procedure takes a pair of loop momentum number (e.g. $k1$ will be 1) along with the power of the loop momenta and forms the a canonicalized Kronecker delta function times the correct symbolic element of $Madj$, $SymMadj$, unless this element is zero.

From the previous chapter, it is known that when multiple numerator loop momenta are symmetrized they form sums of products of Kronecker delta functions. The procedure **zippp** takes two resultants of the **cp** procedure and multiples the two $SymMadj$ pieces together and forms a $KProd$ of the two Kronecker delta functions and returns them as a $CTPair$

Their is also a factor of $1/(2j)!$ associated with the symmetrizing of numerator loop momenta, this is provided by the procedure **kd** <algorithm 36>. In fact **kd** not only calculates the correct factorial term but also recursively calls itself as arguments for the **zippp** procedure along with **cp** to form a CTPair for every pair from the permutation list.

Algorithm 34 Form Kronecker delta functions with the appropriate matrix adjoint term

cp := **proc**(**p1**, **p2**)
local ki1, ki2, k1, k2, mu1, mu2;
$ki1, mu1 := op(p1)$;
$ki2, mu2 := op(p2)$;
if $Madj[ki1, ki2] = 0$ **then**
$[0, KProd(1)]$
else
$k1 := op(ki1, loopMomenta)$;
$k2 := op(ki2, loopMomenta)$;
if $ki1 > ki2$ or ($ki1 = ki2$ and $mu1 > mu2$) **then**
$[SymMadj(ki2, ki1)/detM, KProd(KroneckerDelta(k2, mu2, k1, mu1))]$
else
$[SymMadj(ki1, ki2)/detM, KProd(KroneckerDelta(k1, mu1, k2, mu2))]$
end if
end if
endproc;

Algorithm 35 Combine Kronecker Delta functions

zippp := **proc**(**p1**, **p2**)
local c;
$c := op(1, p1) * op(1, p2)$;
if $c = 0$ **then**
$NULL$
else
$CTPair(c, KProd(op(op(2, p1)), op(op(2, p2))))$
end if
endproc;

Algorithm 36 Find correct numerator factor $(1/(2j)!)$

kd := **proc**(**perm**)
if $perm = []$ **then**
$CTPair(1/np, KProd(1))$
else
$op(map(zippp, [procname(subsop(-1 = NULL,$
$-2 = NULL, perm))], cp(op(-2.. -1, perm))))$
end if
endproc;

The permutations of the loop momenta variables is handled by the main procedure of **symmetrize** <algorithm 37>. This procedure takes a the list of numerator loop momenta forms a *pairList* which is a sequence of pairs of the form $[i, j]$ which consist of loop momenta number i and the sequential value of the power loop momenta up to the maximum loop momenta power. For example if their are two loop momenta variables $[k1, k2]$ and the numerator has the value of $k1_\mu^2$ then the *pairList* would be $[[1, 1], [1, 2]]$. This pairList is then permute to gain a list of of every possible combination of the pairs $[i, j]$ with every other pair in the pairList sequence. The elements of the permutation list are then sent to the **kd** procedure and the results are formed into a $KTSum$ which is returned to the **FP1** procedure.

Algorithm 37 Generate All permutations from the index Paris

if $type(` + `(op(kmlpcty)), ' odd')$ **then**
  $KTSum()$
else
  $pairList := [seq([i, j]\$j = 1..op(i, kmlpcty), i = 1..nops(loopMomenta))]$;
  $perms := combinat[permute](pairList)$;
  $np := nops(perms)$;
  $res := KTSum(op(map(kd, perms)))$;
  res
end if

3.7.5 Code generation

The C-code routines are generated in the following way:

The **nameOrder** procedure <algorithm 38> takes two Feynman parameters and converts them from a symbol (e.g. $x101$ is used as single symbol in the main Maple code) and replaces it with a symbol (x) times a decimal number (101) and then compares the decimal number of the the two Feynman parameters and returns a boolean variable depending on the result.

Algorithm 38 Output file variable names

$nameOrder := proc(a :: symbol, b :: symbol)$
local i,j;
$i := convert(substring(a, 2.. - 1), ' decimal', 10)$;
$j := convert(substring(b, 2.. - 1), ' decimal', 10)$;
$is(i < j)$
endproc :

The procedure **FP** produces a sequence of Feynman parameter integrands, the **MakeCode** procedure takes each integrand in turn and converts it into C-code. The arguments which **MakeCode** takes are: the term number, which is the labeling of each

of the parameter integrands generated by **FP**; the list of Feynman parameters present; the integrand expression; the list of matrices needed to evaluate the integrand and finally the set of global variables which are set in the numerical integration routines. The global variables in general will be the mass of the fields and the kinematic invariants of the diagram.

Each of the matrices is decomposed into a sequence which equates the name of an element in the matrix with the entry of that element and the kinematic invariant p^2 is replaced with variable of momentum name *pipi*, where i is the moment name number (e.g. p3p3).

MakeCode then creates the file whose file name is the term number with the extension .c (e.g. term1.c) and deletes and replaces the file if the same file name already exists in the directory. The sequence of Feynman parameter names are then sorted into the decimal order with **nameOrder** and Maple uses its in built code generating software to convert the integrand expression into a C - readable form. The C options chosen for the output is for it to agree with ANSI C standard, for it to be optimized and for it to be single precision.

The output of a parametric integrand term is given below:

```
#include <math.h>
float term1(float x100,float x101)
{
  float SmuSq;
  float detM;
  float t10;
  float t11;
  float t14;
  float t15;
  float t2;
  float t6;
  float t8;
  extern float m;
  extern float pp;
  {
    detM = x100+x101;
    t2 = m*m;
    t6 = x100*x100;
    SmuSq = (pp*x100+t2*detM)*detM-t6*pp;
    t8 = 0.3141593E1*0.3141593E1;
    t10 = detM*detM;
    t11 = t10*t10;
```

```
    t14 = logf(SmuSq);
    t15 = t8*0.3141593E1/t11*SmuSq;
    return(t15*t14-t15);
  }
}
```

As the **FP** procedure will in general produce more than one parametric integrand term for any particular diagram, it necessary to combine them in the appropriate way. The **MakeCodeDriver** procedure takes the list of Feynman parameters and the maximum term number and produces a file named diagram.c. This file contains a call to each term up to the maximum term number and returns the sum of these terms. It is diagram.c which is called by the numerical integration routines.

The example of diagram.c for the 1-loop 2-point function for ϕ^3 in $d = 6$ is given below:

```
#include <math.h>
float diagram(float x100,float x101)
{
  float t1;
  float t2;
  extern float term1(float,float);
  extern float term2(float,float);
  {
    t1 = term1(x100,x101);
    t2 = term2(x100,x101);
    return(t1+t2);
  }
}
```

3.8 VEGAS Monte Carlo

In all but the most trivial of cases, the resulting parametric integrals produced by the Maple BPHZ Feynman diagram evaluation are best solved numerically. The most widely used class of algorithms for solving numerical integrals of this type are the Monte Carlo algorithms. The VEGAS Monte Carlo algorithm due to Lepage [96] in particular is often used. The VEGAS Monte Carlo is an adaptive Monte Carlo based on importance sampling, which takes samples of the probability distribution, which is approximated by a histogram, to find areas of the integration region which contribute the most to the integral function. The histogramming procedure is run several times, each time refining the regions which the Monte Carlo concentrates on.

The VEGAS Monte Carlo method employed here is due to the *Numerical recipes* group [97], but as the VEGAS Monte Carlo algorithm is well established any VEGAS Monte Carlo program is sufficient.

The main program for using the Numerical recipes VEGAS Monte Carlo routine is shown below:

Up until this point there has been an implicit δ function in the parametric integral $\delta\left(1-\sum_{k=1}^{N} x_k\right)$ and to evaluate the Feynman parameter integral by Monte Carlo it is sensible to explicitly eliminate one of the parameters using the δ function constraint. Doing this directly would mean that there is an introduction of the step function $\theta(x_N)$ and set $x_N = 1-\sum_{i=0}^{N} x_i$ but this leads to an integrand that vanishes over a large part of the unit hypercube. This problem can be solved by writing the integral as a sum over regions where each parameter is largest

$$\begin{aligned}&\int_0^1 dx_1 \cdots \int_0^1 dx_N \delta\left(1-\sum_{k=1}^{N} x_k\right) f(x_1, \ldots x_N) \\ &= \sum_{i=1}^{N} \int_0^1 dx_1 \cdots \int_0^1 dx_N \delta\left(1-\sum_{k=1}^{N} x_k\right) f(x_1, \ldots x_N) \prod_{j\neq i} \theta(x_i \geq x_j) \end{aligned} \quad (3.20)$$

Within each such region the variables are changed to $t_j \equiv x_j/x_i$ with Jacobian det $\frac{\partial x_j}{\partial t_j} x_i^{N-1}$

$$\begin{aligned}&\int_0^1 dx_1 \cdots \int_0^1 dx_N \delta\left(1-\sum_{k=1}^{N} x_k\right) f(x_1, \ldots x_N) \prod_{j\neq i} \theta(x_i \geq x_j) \\ &\int_0^1 dt_1 \cdots \int_0^1 dt_N \delta(t_i - 1) \int_0^1 dx_i\, x_i^{N-1} \delta\left(1 - x_i \sum_{k=1}^{N} t_k\right) f(t_1 x_i, \ldots t_N x_i) \end{aligned} \quad (3.21)$$

The definition $T \equiv \sum_{k=1}^{N} t_k$ is now used to evaluate the integral over x_i to obtain the representation

$$\begin{aligned}&\int_0^1 dx_1 \cdots \int_0^1 dx_N \delta\left(1-\sum_{k=1}^{N} x_k\right) f(x_1, \ldots x_N) \\ &= \int_0^1 dt_1 \cdots \int_0^1 dt_N \frac{f\left(\frac{t_1}{T}, \ldots, \frac{t_N}{T}\right)}{T^N} \sum_{i=1}^{N} \delta(t_i - 1) = \int_0^1 dt_1 \cdots \int_0^1 dt_{N-1} \frac{\bar{f}\left(\frac{t_1}{T}, \ldots, \frac{t_{N-1}}{T}\right)}{T^N} \end{aligned} \quad (3.22)$$

where

$$\begin{aligned}\bar{f}(z_1,\ldots,z_{N-1}) &\equiv f\left(z_1,\ldots,z_{N-1},\frac{1}{T}\right)+f\left(z_2,\ldots,\frac{1}{T},z_1\right)\\ &\quad+\cdots+f\left(z_{N-1},\frac{1}{T},z_1,\ldots,z_{N-2}\right)+f\left(\frac{1}{T},z_1,\ldots,z_{N-1}\right)\end{aligned} \tag{3.23}$$

with $T=1+\sum_{i=1}^{N-1} z_i$.

Chapter 4

Results from ϕ^3-Theory

In this section the results from the Forestry code are presented for 1-loop 2-point and 3-point functions in ϕ^3 theory for both $2\omega = 4$ and $2\omega = 6$ as well as some preliminary results for the 2-loop 2-point function in ϕ^3_4. The results are presented with independent checks, where possible, either derived by the author or from the literature. Finally the complete forest for a 5-loop 2-point function in ϕ^3_6 is generated from the Forestry code to demonstrate how such a calculation would be evaluated with the methods shown in this thesis.

4.1 1-Loop 2-Point Function

Using all the techniques developed thus far in this thesis, the 1-loop 2-point function for $\phi^3_{2\omega}$ will now be calculated.

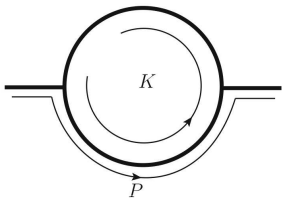


Figure 4.1: The 2-point 1-loop graph of ϕ^3-theory.

The Feynman graph figure 4.1 has the following form:

$$\begin{aligned} I_{4.1} &= \int d^{2\omega}k\, \Delta\left(p+k\right)\Delta\left(k\right), \\ &= \int d^{2\omega}k\, \frac{1}{\left(\left(p+k\right)^2+m^2\right)\left(k^2+m^2\right)}. \end{aligned} \tag{4.1}$$

Using the Feynman parameter formula eq.(2.29) and the dimensional regularizarization identity eq.(2.41) derived earlier, yields:

$$\begin{aligned} I_{4.1} &= \Gamma(2)\int_0^1 dx \int d^{2\omega}k \, \frac{1}{\left[\left(x((p+k)^2+m^2\right)+(1-x)\left(k^2+m^2\right)\right]^2}, \\ &= \Gamma(2)\int_0^1 dx \int d^{2\omega}k \, \frac{1}{\left[k^2+2xk\cdot p+xp^2+m^2\right]^2}, \\ &= \Gamma(2)\int_0^1 dx \int d^{2\omega}k \, \frac{1}{\left[\left(k+xp\right)^2+x\left(1-x\right)p^2+m^2\right]^2}, \\ &= \Gamma(2)\int_0^1 dx \int d^{2\omega}k' \, \frac{1}{\left[k'^2+\mu^2\right]^2}, \\ &= \pi^{\omega}\Gamma(2-\omega)\int_0^1 dx \, \left(\mu^2\right)^{\omega-2}. \end{aligned} \tag{4.2}$$

where $k \equiv k' + k_0$ with $k_0 = -xp$ and $\mu^2 \equiv x\left(1-x\right)p^2 + m^2$.

For the massless case this can simplified yet further by setting $m = 0$ and factoring out p^2 in the last step in the above equation.

$$I_{4.1} = \left(p^2\right)^{\omega-2}\Gamma(2-\omega)\int_0^1 dx \, \left(x\left(1-x\right)\right)^{\omega-2} \tag{4.3}$$

4.1.1 The case for $\omega = 2 - \varepsilon$

Using the power counting methods, it is seen that in $\omega = 2$ dimensions eq.(4.2) has a $\deg(\mathcal{G}) = 0$, and as such will need to have the resulting divergence subtracted. Making a Taylor series expansion about $p^2 = 0$ of eq.(4.2) to leading order yields the necessary subtraction term, to give:

$$\begin{aligned} \overline{I} = I - TI &= \pi^{\omega}\Gamma(2-\omega)\int dx \, [x(1-x)p^2+m^2]^{\omega-2} \\ &\quad -[m^2]^{\omega-2} \end{aligned} \tag{4.4}$$

putting in the value for ω

$$\begin{aligned} \omega = 2-\varepsilon: \quad \overline{I} &= \pi^2\Gamma(\varepsilon)\int dx \, \left\{[x(1-x)p^2+m^2]^{-\varepsilon}\right. \\ &\quad \left.-[m^2]^{-\varepsilon}\right\} \\ &= -\pi^2\Gamma(\varepsilon)\int dx \, \varepsilon \ln[x(1-x)p^2+m^2] - \varepsilon\ln(m^2) \\ &= -\pi^2\int dx \, \ln[x(1-x)p^2+m^2] - \ln(m^2) + O(\varepsilon) \end{aligned} \tag{4.5}$$

using $\Gamma(\varepsilon) = \frac{1}{\varepsilon} - \gamma + O(\varepsilon)$ in the last step.

This corresponds exactly with the result of the Forestry:

```
fp0:=
-Pi^2*(ln((pp*x100*x101+x100^2*m^2+2*x100*m^2*x101+x101^2*m^2)/(x100+x101))
-ln(m^2*(x100+x101)))/(x100+x101)^2
```

upon using the Kronecker δ function condition $x101 = 1 - x100$.

Eq.(4.5) can be evaluated by making the substitution $q = \frac{p}{m}$ to give:

$$\overline{I}_{4.1}(p, m) = -\pi^2 \int_0^1 dx \, \ln\left(x(1-x)q^2 + 1\right) \tag{4.6}$$

The outcome of the resulting integral is shown in fig. 4.7

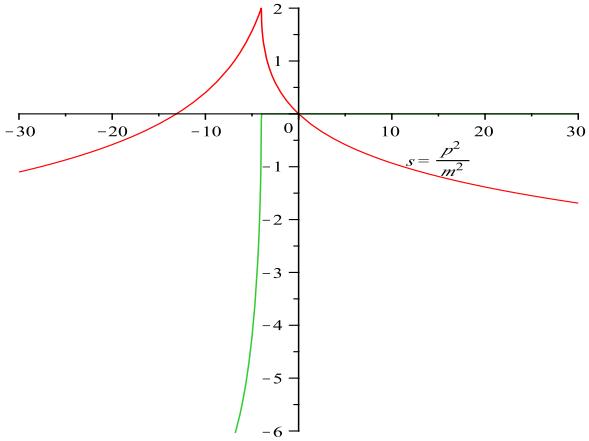


Figure 4.2: The real (red) and imaginary (green) parts of $\overline{I}(p, m)/\pi^2$ for ϕ_4^3 at subtraction point of $p^2 = 0$.

For the massless case, performing the ε expansion directly on eq.(4.3), it is easily seen that the necessary subtraction term is $KI = -\frac{\pi^2}{\varepsilon} + \pi^2\gamma$ to cancel the pole and the Euler-Mascheroni constant, which then leads the renormalized result of:

$$\overline{I}_{4.1}(p) = -\pi^2 \ln p^2 \tag{4.7}$$

Using the PolePart command in the Forestry code, the code picks up the correct term necessary to cancel the pole in the Γ function.

```
PolePart(rg0);
   -Pi^2/(x100+x101)^2
```

4.1.2 The case for $\omega = 3 - \varepsilon$

Turning now to the 2-point 1-loop function in ϕ_6^3, the degree of divergence for eq.(4.2) becomes $\deg(\mathcal{G}) = 2$, so quadratic divergences are now present within the integral. Making the usual Taylor series expansion of eq.(4.2) about $p^2 = 0$ up to $O(p^4)$ results in the following integral:

$$\begin{aligned}\overline{I} = I - TI &= \pi^\omega \Gamma(2-\omega) \int dx\, [x(1-x)p^2 + m^2]^{\omega-2} \\ &\quad -[m^2]^{\omega-2} - (\omega-2)[m^2]^{\omega-3} \cdot x(1-x)p^2\end{aligned}$$

putting in the value for ω

$$\begin{aligned}\omega = 3-\varepsilon: \quad \overline{I} &= \pi^3 \Gamma(\varepsilon-1) \int dx\, \left\{[x(1-x)p^2 + m^2]^{1-\varepsilon}\right. \\ &\quad \left. -[m^2]^{1-\varepsilon} - (1-\varepsilon)[m^2]^{\varepsilon} \cdot x(1-x)p^2\right\} \\ &= \pi^3 \frac{\Gamma(\varepsilon)}{\varepsilon - 1} \int dx\, \left\{[x(1-x)p^2 + m^2]\left[1 - \varepsilon \ln[x(1-x)p^2 + m^2]\right]\right. \\ &\quad \left. -m^2[1 - \varepsilon \ln m^2] - (1-\varepsilon)(1 - \varepsilon \ln m^2)x(1-x)p^2\right\} \\ &= -\frac{\pi^3}{\varepsilon} \int dx\, \left\{[x(1-x)p^2 + m^2] - m^2 - x(1-x)p^2\right. \\ &\quad -\varepsilon \left([x(1-x)p^2 + m^2]\ln[x(1-x)p^2 + m^2] - m^2 \ln m^2 - x(1-x)p^2\right. \\ &\quad \left.\left. -[x(1-x)p^2]\ln m^2\right)\right\} \\ &= \pi^3 \int dx\, \left\{\left([x(1-x)p^2 + m^2]\ln[x(1-x)p^2 + m^2] - m^2 \ln m^2 - x(1-x)p^2\right.\right. \\ &\quad \left.\left. -[x(1-x)p^2]\ln m^2\right)\right\} \qquad (4.8)\end{aligned}$$

which again matches with result of the Forestry code once the δ-function constraint is invoked:

```
fp0:=
Pi^3*(x100^2*pp*ln(m^2*(x100+x101))-x100^2*pp*ln(x100+x101)
+ln((pp*x100*x101+x100^2*m^2+2*x100*m^2*x101+x101^2*m^2)
/(x100+x101))*pp*x100^2*x101+ln((pp*x100*x101+x100^2*m^2
+2*x100*m^2*x101+x101^2*m^2)/(x100+x101))*pp*x100*x101^2
+3*ln((pp*x100*x101+x100^2*m^2+2*x100*m^2*x101+x101^2*m^2)
/(x100+x101))*x100^2*m^2*x101+3*ln((pp*x100*x101+x100^2*m^2
+2*x100*m^2*x101+x101^2*m^2)/(x100+x101))*x100*m^2*x101^2
+ln(x100+x101)*pp*x100^2*x101-3*m^2*ln(m^2*(x100+x101))*x100^2
*x101-3*m^2*ln(m^2*(x100+x101))*x100*x101^2-2*x100^2*pp
*ln(m^2*(x100+x101))*x101-x100*pp*ln(m^2*(x100+x101))*x101^2
```

```
-pp*x100^2*x101-pp*x100*x101^2+ln((pp*x100*x101+x100^2*m^2
+2*x100*m^2*x101+x101^2*m^2)/(x100+x101))*x100^3*m^2
+ln((pp*x100*x101+x100^2*m^2+2*x100*m^2*x101+x101^2*m^2)
/(x100+x101))*x101^3*m^2-m^2*ln(m^2*(x100+x101))*x100^3
-m^2*ln(m^2*(x100+x101))*x101^3-x100^3*pp*ln(m^2*(x100+x101))
+x100^3*pp*ln(x100+x101))/(x100+x101)^5
```

To evaluate the eq.(4.8), the usual substitution of $q = \frac{p}{m}$ is made to get:

$$\overline{I}_{4.1}(p,m) = \int_0^1 \pi^3 m^2 \left\{ [x(1-x)q^2+1] \ln[x(1-x)q^2+1] - x(1-x)q^2 \right\} \tag{4.9}$$

with the evaluated results show in fig. 4.3.

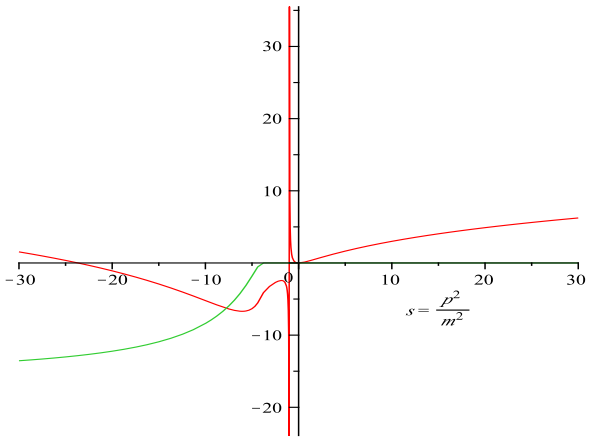


Figure 4.3: The real (red) and imaginary (green) parts of $\overline{I}(p,m)/\left(\pi^2(p^2+m^2)\right)$ for ϕ_6^3 at subtraction point of $p^2 = 0$.

Once again for the massless case, performing the ε expansion directly on eq.(4.3) and using $\Gamma(\varepsilon - 1) = \left(\frac{1}{\varepsilon} - \gamma + 1 + (\varepsilon)\right)$, it is seen that the necessary subtraction term is $KI = -\frac{\pi^3 p^2}{6\varepsilon} + \frac{\pi^3 p^2 \gamma}{6}$ to cancel the pole and the Euler-Mascheroni constant, which then leads to the renormalized result of:

$$I\overline{I}_{4.1}(p) = \frac{\pi^3}{6} \ln p^2 \tag{4.10}$$

The Forestry code returns the following for the pole part of eq.(4.8):

```
PolePart(rg0)
Pi^3*(m^2*x100^3+3*m^2*x100^2*x101+3*m^2*x100*x101^2+m^2*x101^3
-x100^2*pp+pp*x100^3+2*pp*x100^2*x101+pp*x100*x101^2)/(x100+x101)^5
m^2*x100^3+3*m^2*x100^2*x101+3*m^2*x100*x101^2+m^2*x101^3
-x100^2*pp+pp*x100^3+2*pp*x100^2*x101+pp*x100*x101^2))
```

which when the mass is set to zero and the δ function constraint is used, reduces to:

```
pp0 := subs({m = 0, x101 = 1 - x100}, x100 = x, PolePart(rg0));
            pp0 := Pi^3*(-x^2*pp+pp*x^3+2*pp*x^2*(1-x)+pp*x*(1-x)^2)
```

performing the resulting integral returns:

```
factor(int(pp0, x = 0..1));
            (1/6)*Pi^3*pp
```

which is, as seen earlier, the correct factor to cancel the pole in the integral.

4.1.3 For a Taylor series expansion around $p^2 = -m^2$

To compare the results from the Forestry code, and the earlier derivation to some of the results in the literature [98], the renormalized integral for the 2-point function in ϕ_6^3 is recalculated with the subtractions obtained from a Taylor series expansion of eq.(4.2) about $p^2 = -m^2$ up to $O((p^2+m^2)^2)$.

$$\begin{aligned}\bar{I} = I - KI &= \pi^\omega \Gamma(2-\omega) \int dx \left\{ [x(1-x)p^2 + m^2]^{\omega-2} - \left[m^2\left(x(x-1)+1\right)\right]^{\omega-2} \right. \\ &\left. +(\omega-2)x(x-1)(p^2+m^2)\left[m^2\left(x(x-1)+1\right)\right]^{\omega-3} \right\}\end{aligned}$$

Putting in the value of ω:

$$\begin{aligned}\omega = 3-\varepsilon: \quad \overline{I} &= \pi^3\Gamma(\varepsilon-1)\int dx\ \left\{ [x(1-x)p^2+m^2]^{1-\varepsilon} \right. \\ &\left. -\left[m^2\left(x(x-1)+1\right)\right]^{1-\varepsilon} + (1-\varepsilon)x(x-1)(p^2+m^2)\left[m^2\left(x(x-1)+1\right)\right]^{-\varepsilon} \right\} \\ &= \pi^3\Gamma(\varepsilon-1)\int dx\ \left\{ [x(1-x)p^2+m^2]\left[1-\varepsilon\ln[x(1-x)p^2+m^2]\right] \right. \\ &+\left[m^2\left(x(x-1)+1\right)\right]\left[1-\varepsilon\ln\left[m^2\left(x(x-1)+1\right)\right]\right] \\ &\left. +(1-\varepsilon)x(x-1)(p^2+m^2)\left[m^2\left(x(x-1)+1\right)\right]\left[1-\varepsilon\ln\left[m^2\left(x(x-1)+1\right)\right]\right] \right\}\end{aligned}$$

which results in the finite integral:

$$\begin{aligned}\overline{I}_{4.1}(p,m) &= -\pi^3\int dx\ \left\{ x(1-x)p^2 - [x(1-x)p^2+m^2]\ln[x(1-x)p^2+m^2] \right. \\ &\left. +x(1-x)m^2 + [x(1-x)p^2+m^2]\ln[m^2-x(1-x)m^2] \right\} \qquad (4.11)\end{aligned}$$

and making the usual substitution of $q = \frac{p}{m}$ to obtain:

$$\begin{aligned} \overline{I} &= -\pi^2 m^2 \int_0^1 dx \, \{x(1-x)q^2 - [x(1-x)p^2 + 1] \ln[x(1-x)p^2 + 1] \\ &\quad + x(1-x) + [x(1-x)q^2 + 1 \ln[1 - x(1-x)]\} \end{aligned} \tag{4.12}$$

which gives the solution:

$$\frac{\overline{I}_{4.1}}{\pi^3} = \frac{m^2}{6u} \left(u(c_1 s + c_2) + 2(4+s)^2 v\right) \tag{4.13}$$

where $c_1 = 3 - \pi\sqrt{3}$, $c_2 = c_1 = 3 - 2\pi\sqrt{3}$, $u = \sqrt{s(4+s)}$ and $v = \text{arctanh}\left(\frac{s}{u}\right)$

with the results of the above equation plotted in fig. 4.4.

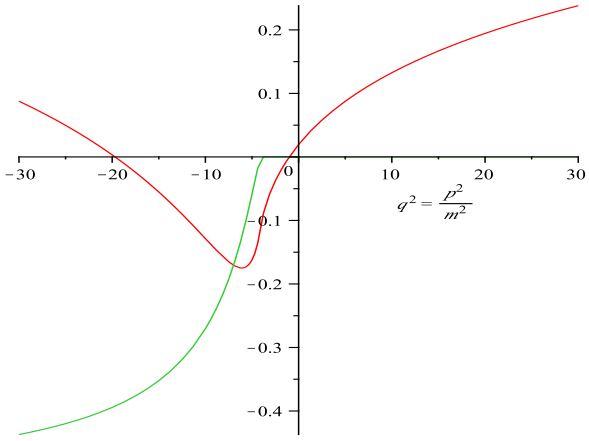


Figure 4.4: The real (red) and imaginary (green) parts of $\overline{I}(p,m)/\pi^2$ for ϕ_6^3 at subtraction point of $p^2 = -m^2$, which is in agreement with literature results [98].

4.2 1 Loop 3-Point Function

Turning now to the 1-loop 3-point function in $\phi_{2\omega}^3$, fig. 4.5 gives:

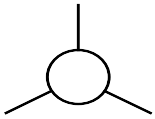


Figure 4.5: The 3-point 1-loop graph of ϕ^3-theory.

$$\begin{aligned} I(\mathcal{G}) &= \int d^{2\omega}k\,\Delta(p_1+k)\Delta(p_1+p_2+k)\Delta(k) \\ &= \int d^{2\omega}k\,\frac{1}{\left((p_1+k)^2+m^2\right)\left((p_1+p_2+k)^2+m^2\right)(k^2+m^2)} \end{aligned} \tag{4.14}$$

Applying the Feynman parameter formula eq.(2.29) to the above equation:

$$\begin{aligned} I(\mathcal{G}) &= \Gamma(3)\int_0^1 dx_1\,dx_2\,dx_3\,\delta\left(1-\sum_{i=1}^{3}x_i\right) \\ &\int d^{2\omega}k\,\frac{1}{\left[x_1\left((p_1+k)^2+m^2\right)+x_2\left((p_1+p_2+k)^2+m^2\right)+x_3\left(k^2+m^2\right)\right]^3} \end{aligned} \tag{4.15}$$

from this, the Feynman parameter matrices are formed along the lines of eq.(2.47), where the matrices M, B, C and D are given by:

$$M=(x_1+x_2+x_3)\,;\quad B=\begin{pmatrix} -x_1-x_2 & -x_2 \end{pmatrix};\quad C=\begin{pmatrix} x_1+x_2 & x_2 \\ x_2 & x_2 \end{pmatrix}$$

$$D=m^2(x_1+x_2+x_3);\quad P=\begin{pmatrix} p_1 \\ p_2 \end{pmatrix};\quad k'=k. \tag{4.16}$$

.

using the above matrices and manipulating the denominator of eq.(4.15) to be in the form of $Q(k)=k''^2+\mu^2$, to get for μ^2

$$\begin{aligned} \mu^2 &= P^T\left[C-B^T.M^{-1}.B\right]P+D, \\ &= p_1^2\left(x_1x_3+x_2x3\right)+2p_1\cdot p_2x_2x_3+p_2^2\left(x_2x1+x_2x3\right), \\ &\quad +m^2\left(x_1^2+2x_1x_2+2x_1x_3+x_2^2+2x_2x_3+x_3^2\right), \\ &= m^2+p_1^2\left(x_1-x_1x_2-x_1^2\right)+p_2^2x_1x_2+p_3^2\left(x_2-x_2x_1-x_2^2\right). \end{aligned} \tag{4.17}$$

using the momentum conservation $p_1+p_2+p_3=0$ to get $p_1\cdot p_2=\frac{p_1^2+p_2^2+p_3^3}{2}$, and the Feynman paramater constraint $\delta(1-x_1-x_2-x_3)$. The Feynman parameter constraint also leads to $\det M=1$ and consequently $k''=k'$.

This reduces eq.(4.15) to:

$$I_{4.5} = \Gamma(3) \int_0^1 dx_1\, dx_2 \int d^{2\omega}k'' \left[k''^2 + \mu^2\right]^{-3}. \tag{4.18}$$

Finally using the general form for Euclidean integrals in 2ω dimensions eq.(2.52) on the above equation, yields:

$$I_{4.5}(p_1,p_2,p_3,p_4) = \frac{\Gamma(3)\Gamma\left(\frac{1}{2}\right)^{2\omega}\Gamma(3-\omega)}{\Gamma(3)} \int_0^1 dx_1\, dx_2 \left(\mu^2\right)^{\omega-3}. \tag{4.19}$$

4.2.1 The case of $\omega = 2$

The 1-loop 3-point function in ϕ_4^3 is both UV and IR convergent and as such is not necessary to make an ε expansion on the integrand as there are no subtractions necessary.

As such, after putting in the value of $\omega = 2$ into eq.(4.19), the resulting integral is simply:

$$\begin{aligned} I_{4.5}(p_1,p_2,p_3,p_4) &= \pi^2 \int_0^1 dx_1\, dx_2 \left[\left(m^2 + p_1^2\left(x_1 - x_1x_2 - x_1^2\right)\right.\right. \\ &\quad \left.\left. + p_2^2 x_1 x_2 + p_3^2\left(x_2 - x_2x_1 - x_2^2\right)\right]^{-1} \end{aligned} \tag{4.20}$$

which is the same result as the Forestry code after the δ function constraint is used and the substitutions $p_1 \cdot p_2 = \frac{p_1^2+p_2^2+p_3^3}{2}$.

```
fp1 :=
Pi^2/((x1+x2+x3)*(p1p1*x1*x3+p1p1*x2*x3+2*p1p2*x2*x3+p2p2*x2*x1+p2p2*x2*x3
+x1^2*m^2+2*x1*m^2*x2+2*x1*m^2*x3+x2^2*m^2+2*x2*m^2*x3+x3^2*m^2))
```

Looking at the massless case first, μ^2 in the above equation can be simplified by using the change of variables $P_1 = \frac{p_1^2}{p_3}$ and $P_2 = \frac{p_2^2}{p_3}$ in eq.(4.17) and setting m=0 to give:

$$\mu^2 = P_1\left(x_1 - x_1x_2 - x_1^2\right)p_3^2 + P_2 x_1 x_2 p_3^2 + \left(x_2 - x_2x_1 - x_2^2\right)p_3^2. \tag{4.21}$$

which becomes

$$\begin{aligned} I_{4.5}(p_1,p_2,p_3,p_4) &= \pi^2 \int_0^1 dx_1\, dx_2 \left[p_3^2\left(P_1\left(x_1 - x_1x_2 - x_1^2\right)\right.\right. \\ &\quad \left.\left. + P_2 x_1 x_2 + \left(x_2 - x_2x_1 - x_2^2\right)\right)\right]^{-1} \end{aligned} \tag{4.22}$$

An analytic result is given by Davydychev [99] for $\omega = 2$, using $x = \frac{p_1^2}{p_3^2}$ and $y = \frac{p_2^2}{p_3^2}$, as $I(x,y) = \frac{\pi^2\Phi(x,y)}{p_3^2}$ where

$$\Phi(x,y) = \int_0^1 d\xi \, \frac{\ln\left((1-\xi(x+\xi y) - \ln(\xi) - \ln(1-\xi)\right)}{(1-\xi)x + \xi y_\xi (1-\xi)} \tag{4.23}$$

referred to as the α representation (named as such due to the use of the Schwinger parameters to calculate it) and this can have a dilogarithm representation

$$\begin{aligned}\Phi(x,y) &= \frac{1}{\lambda}\left(2\ln\left(\frac{1}{2}+\frac{1}{2}x-\frac{1}{2}y-\frac{1}{2}\lambda\right)\ln\left(\frac{1}{2}-\frac{1}{2}x+\frac{1}{2}y-\frac{1}{2}\lambda\right)\right. \\ &\quad -\ln(x)\ln(y) - 2\text{dilog}\left(\frac{1}{2}-\frac{1}{2}x+\frac{1}{2}y+\frac{1}{2}\lambda\right) \\ &\quad \left.-2\text{dilog}\left(\frac{1}{2}+\frac{1}{2}x-\frac{1}{2}y+\frac{1}{2}\lambda\right)+\frac{1}{3}\pi^2\right) \end{aligned} \tag{4.24}$$

where $\lambda = \sqrt{(1-x-y)^2 - 4xy}$.

The results of three representations after performing the integral of eq.(4.22), are shown in table 4.1 for random value of P_1, P_2 between 0 and 1, in which it is shown that the result obtained by the code matches that of Davydychev.

Table 4.1: Results for the massless 1-Loop 3-Point functionin ϕ_4^3

$\Phi(P,Q)$	α representation	dilog	Forestry Code
$\Phi(0.1292870000, 0.4058620000)$	5.692470643,	5.69247066,	5.692470664,
$\Phi(0.9029870000, 0.6880622000)$	2.737419762,	2.737419762,	2.737419776,
$\Phi(0.4861374000, 0.1739097000)$	4.979965912,	4.979965912,	4.979965926,
$\Phi(0.8078809000, 0.6832282000)$	2.847660859,	2.847660858,	2.847660874,
$\Phi(0.9911880000, 0.5344486000)$	2.863214641,	2.863214640,	2.863214657,

Returning once more to the massive case, μ^2 in eq.(4.19) becomes

$$\mu^2 = m^2\left(1 + P_1(x_1 - x_1x_2 - x_1^2) + P_2x_1x_2 + P_3(x_2 - x_1x_2 - x_2^2)\right) \tag{4.25}$$

after making the substitutions $P_1 = \frac{p_1^2}{m^2}$, $P_2 = \frac{p_2^2}{m^2}$ and $P_3 = \frac{p_3^2}{m^2}$.

Factoring out powers of m^2 and π, eq.(4.20) obtains:

$$\begin{aligned} I_{4.5}(p1,p2,p3,m) &= \frac{m^2}{\pi^2}\Phi(P_1,P_2,P_3) \\ &= \left[(x_1+x_2+x_3)\left(P_1x_1x_3 + x_2x_3P_3 + P_2x_2x_1 + x_1^2 + 2x_1x_2\right.\right. \\ &\qquad \left.\left. +2x_1x_3 + x_2^2 + 2x_2x_3 + x_3^2\right)\right]^{-1} \end{aligned} \tag{4.26}$$

which when integrated over the three Feynman variables gives the results in table 4.2 for random values of P_1, P_2 and P_3 between 0 and 1.

Table 4.2: Henge code results for the massive 1-Loop 3-Point function in ϕ_4^3 for specific values of momentum P_1, P_2 and P_3

$\Phi(P_1, P_2, P_3)$	Henge Code Result
$\Phi(0.4297752000, 0.5286489000, 0.0663355000)$	0.4611178497,
$\Phi(0.4525242000, 0.2385551000, 0.8242048000)$	0.4446666771,
$\Phi(0.7181350000, 0.1292870000, 0.4058620000)$	0.4533542067,
$\Phi(0.8547723000, 0.4861374000, 0.1739097000)$	0.4447944519,
$\Phi(0.0784967000, 0.2977395000, 0.8724206000)$	0.4538351962,

4.2.2 For the case of $\omega = 3 - \varepsilon$

For $\omega = 3 - \varepsilon$, the integral eq.(4.19) has a degree of divergence of $\deg(\mathcal{G}) = 0$, so requires a constant subtraction term to render it finite.

Performing a Taylor expansion of equation (4.19) about $p_3^2 = 0$, leads to the constant subtraction term $\Gamma\left(\frac{1}{2}\right)^{2\omega} \Gamma(3-\omega) \left(m^2\right)^{\omega-3}$

Leading to the finite integral:

$$\begin{aligned}
(1-\bar{K})I_{4.5} &= \Gamma\left(\frac{1}{2}\right)^{2\omega} \Gamma(3-\omega) \left\{ \int_0^1 dx_1\, dx_2\, dx_3\ \left(\mu^2\right)^{\omega-3} - \left(m^2\right)^{\omega-3} \right\}, \\
&= \pi^{3-\varepsilon}\Gamma(\varepsilon) \left\{ \int_0^1 dx_1\, dx_2\, dx_3\ \left(\mu^2\right)^{-\varepsilon} - \left(m^2)\right)^{-\varepsilon} \right\} \\
&= \pi^3 \left(1 - \varepsilon \ln(\pi) + O(\varepsilon^2)\right) \left(\frac{1}{\varepsilon} - \gamma + O(\varepsilon)\right) \\
&\quad \times \left\{ \int_0^1 dx_1\, dx_2\, dx_3\ \left(1 - \varepsilon \ln\left(\mu^2\right) + O(\varepsilon^2)\right) - \left(1 - \varepsilon \ln\left(m^2\right) + O(\varepsilon^2)\right) \right\} \\
&= -\pi^3 \left(\int_0^1 dx_1\, dx_2\, dx_3\ \ln\left(\mu^2\right) - \ln\left(m^2\right) \right) \\
&= -\pi^3 \left(\int_0^1 dx_1\, dx_2\, dx_3\ \ln\left(m^2\left(1 + P_1(x_1 - x_1x_2 - x_1^2) + P_2 x_1 x_2\right)\right.\right. \\
&\quad \left.\left. + P_3(x_2 - x_1x_2 - x_2^2)\right) - \ln\left(m^2\right)\right)
\end{aligned} \tag{4.27}$$

which makes use of the substitutions $P_1 = \frac{p_1^2}{m^2}$, $P_2 = \frac{p_2^2}{m^2}$ and $P_3 = \frac{p_3^2}{m^2}$ and matches with the Forestry code output after the momentum substitutions and the usual δ function constraint.

```
-Pi^3*(ln((p1p1*x1*x3+p1p1*x2*x3+2*p1p2*x2*x3+p2p2*x2*x1+p2p2*x2*x3
+x1^2*m^2+2*x1*m^2*x2+2*x1*m^2*x3+x2^2*m^2+2*x2*m^2*x3+x3^2*m^2)/(x1+x2+x3))
-ln(m^2*(x1+x2+x3)))/(x1+x2+x3)^3
```

The evaluations of resulting integral are shown in table 4.3 for random values of P_1, P_2, P_3 between 0 and 1.

Table 4.3: Henge code results for the massive 1-Loop 3-Point function in ϕ_6^3 for specific values of P_1, P_2 and P_3

$\Phi(P_1, P_2, P_3)$	Henge Code Result
$\Phi(0.8009118000, 0.5153566000, 0.7831792000)$	-0.08018132941
$\Phi(0.3764154000, 0.1790694000, 0.05266570000)$	-0.02461244957
$\Phi(0.4294795000, 0.8996528000, 0.1619033000)$	-0.05804637582
$\Phi(0.2751522000, 0.2478862000, 0.07873530000)$	-0.02439842244
$\Phi(0.1684579000, 0.5090165000, 0.7950231000)$	-0.05745049421

4.3 2 Loop 2-Point Function

For the final calculation, the overlapping contribution to the 2-loop 2-point function in ϕ_4^3 fig.4.6 is examined.

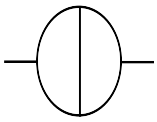


Figure 4.6: The 2-point 2-loop graph of ϕ^3-theory with and overlapping divergence.

This graph is given by the integral:

$$I_{4.6} = \int d^{2\omega}k_1\, d^{2\omega}k_2\, \frac{1}{((p+k_1+k_2)^2+m^2)}\frac{1}{((k_1+k_2)^2+m^2)}\frac{1}{\left(k_1^2+m^2\right)} \times \frac{1}{((p+k_2)^2+m^2)}\frac{1}{\left(k_2^2+m^2\right)} \tag{4.28}$$

Taking the lead from Smirnov [75], the massless case can be calculated using 1-loop insertions to construct a 2-loop graph and the general result is given by:

$$I_4.6 = \frac{\Gamma\left(\frac{1}{2}\right)^{4\omega}\Gamma(2-\omega)\Gamma(\omega-1)^2\Gamma(\omega-2)}{(\omega-2)\Gamma(2\omega-2)\Gamma(3-\omega)\Gamma(3\omega-5)\Gamma(2\omega-3)} \times \left[\Gamma(\omega-1)\Gamma(3-\omega)^2\Gamma(3\omega-5) - \Gamma(5-2\omega)\Gamma(2\omega-3)^2\right]\left(p^2\right)^{2\omega-5} \tag{4.29}$$

which in $\omega = 2$ gives the result:

$$I_{4.6} = \frac{6\Pi^4\zeta(3)}{p^2} \approx \frac{702.5476218}{p^2} \tag{4.30}$$

The output for the Forestry code is given by:

```
1/(((x7+x6+x8)*x4+(x5+x7+x8)*x6+x5*x7+x5*x8)*(((x8+x5)*x6+(x5+x7)*x8+x5*x7)*x4
+(x7*x8+x5*x7)*x6+x7*x5*x8))
```

which when the δ function is used and the resulting integral evaluated gives:

$$I(p) = \frac{702.6903137}{p^2} \tag{4.31}$$

as required.

The massive case proves to be the limit of what Maple can evaluate numerically. The Forestry code produces the following output:

```
Pi^4/
((x4*x7+x4*x8+x5*x7+x5*x8+x6*x4+x6*x5+x6*x7+x6*x8)
*(p3p3*x4*x5*x7+p3p3*x7*x4*x8+p3p3*x4*x6*x8+p3p3*x4*x6*x5
+p3p3*x4*x5*x8+2*x4*m^2*x5*x8+2*x4*m^2*x5*x7+p3p3*x7*x6*x8
+p3p3*x7*x6*x5+p3p3*x7*x5*x8+3*x4*m^2*x6*x8+3*x4*m^2*x6*x7
+2*x4*m^2*x6*x5+3*x5*m^2*x6*x8+3*x5*m^2*x6*x7+2*x7*m^2*x5*x8
+2*x7*m^2*x4*x8+2*x7*m^2*x6*x8+x5^2*m^2*x8+x5^2*m^2*x7
+x4^2*m^2*x6+x8^2*m^2*x6+x4^2*m^2*x8+x8^2*m^2*x5+x6^2*m^2*x4
+x5^2*m^2*x6+x7^2*m^2*x6+x6^2*m^2*x5+x7^2*m^2*x4+x7^2*m^2*x5
+x6^2*m^2*x8+x8^2*m^2*x4+x6^2*m^2*x7+x4^2*m^2*x7))
```

which corresponds with:

$$\begin{aligned} I(p,m) &= \frac{m^2}{\pi^4} I'\left(s = \frac{p^2}{m^2}\right) \\ &= [(((x_7+x_6+x_8)x_4 + (x_5+x_7+x_8)x_6 + x_5x_7 + x_5x_8)\,((((x_8+x_5)x_6 \\ &\quad +(x_5+x_7)x_8 + x_5x_7)\,x_4 + (x_7x_8 + x_5x_7)x_6 + x_7x_5x_8)\,s + (x_7+x_6+x_8)x_4^2 \\ &\quad + \left(x_6^2 + (2x_5+3x_8+3x_7)x_6 + x_8^2 + (2x_5+2x_7)x_8 + x_7(x_7+2x_5)\right)x_4 \\ &\quad +(x_5+x_7+x_8)x_6^2 + \left(x_8^2 + (2x_7+3x_5)x_8 + x_7^2 + 3x_5x_7 + x_5^2\right)x_6 \\ &\quad +x_8^2x_5 + x_5(x_5+2x_7)x_8 + x_5x_7(x_5+x_7))\big)\big]^{-1} \end{aligned} \tag{4.32}$$

A sample of results are shown in table 4.4, and plot in fig. 4.7.

Table 4.4: Results for the massive 2-Loop 2-Point function

$\Gamma(s)$	Henge Code Result
$\Gamma(0.1791140000)$	0.7608870973,
$\Gamma(0.5744387000)$	0.7198412702,
$\Gamma(0.9996601000)$	0.6809054032,
$\Gamma(0.3485546000)$	0.7426587014,
$\Gamma(-0.2546720000)$	0.8125957761,

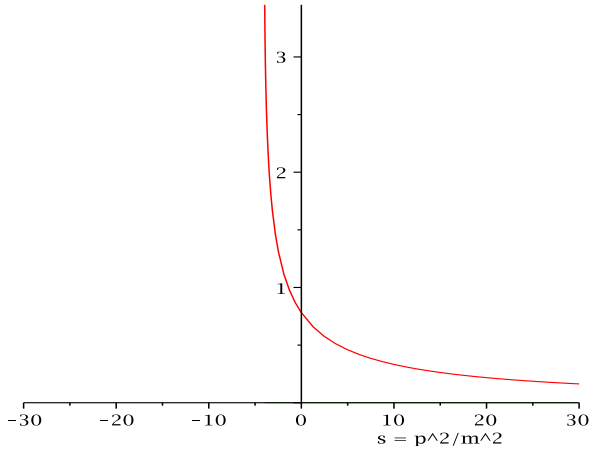


Figure 4.7: The real (red) parts of $\overline{I}(p,m)/\left(\pi^2(p^2+m^2)\right)$ for ϕ_6^3 at subtraction point of $p^2=0$.

4.4 5 Loop 2-Point Function Forest and code limitations

There a several possible limiting factors, for this code. The first is numerical integration of the results in the Maple enviroment with anything beyond 2-point 2-loop ϕ_4^3 impossible. Problematically these are the interesting case where overlapping divergences will occur.

However the calculation of the Forestry output is not the bottle neck, as demonstrated below:

This code scrap is the edges for the 5-loop 2-point function graph in ϕ_6^3:

```
table(edge, [I30 = edge(-k1+k2-k3-k4-k5, -2, NumeratorMomenta(0)),
             I29 = edge(p7-k1+k2-k3-k4-k5, -2, NumeratorMomenta(0)),
             I31 = edge(-k1+k2-k3-k4, -2, NumeratorMomenta(0)),
             I28 = edge(p7+k2-k3-k4-k5, -2, NumeratorMomenta(0)),
             I32 = edge(k2-k3-k4, -2, NumeratorMomenta(0)),
             I27 = edge(p7+k2-k3-k4, -2, NumeratorMomenta(0)),
             I33 = edge(k2-k3, -2, NumeratorMomenta(0)),
             I34 = edge(k2, -2, NumeratorMomenta(0)),
             I25 = edge(p7+k2, -2, NumeratorMomenta(0)),
             E13 = edge(-p7, -2, NumeratorMomenta(0)),
             I36 = edge(k4, -2, NumeratorMomenta(0)),
             I35 = edge(k3, -2, NumeratorMomenta(0)),
             I26 = edge(p7+k2-k3, -2, NumeratorMomenta(0)),
             E12 = edge(p7, -2, NumeratorMomenta(0)),
             I38 = edge(k1, -2, NumeratorMomenta(0)),
```

```
I37 = edge(k5, -2, NumeratorMomenta(0))])

EXT(E12, E13),
INT(I25, I26, I27, I28, I29, I30, I31, I32, I33, I34, I35, I36, I37, I38)
```

which easily generates the following forest (shown graphically)

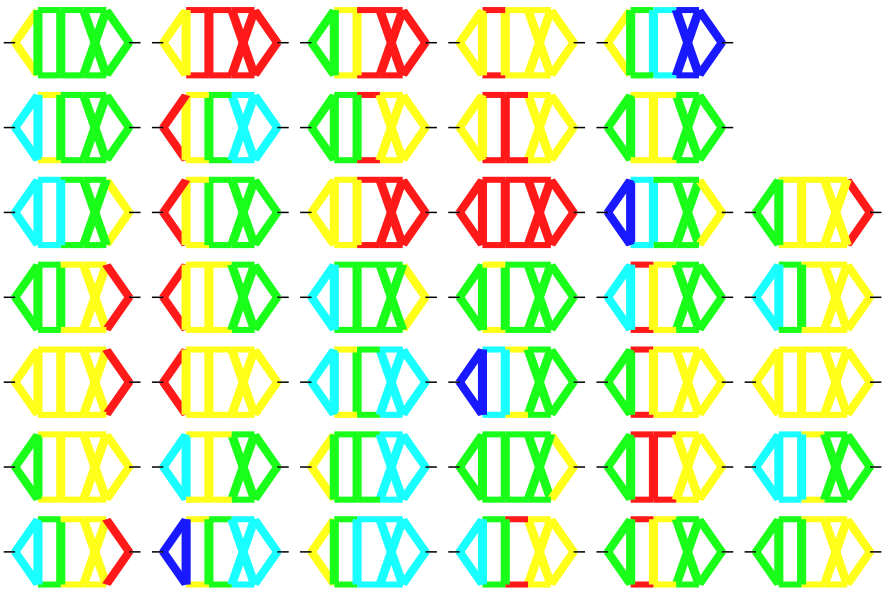


Figure 4.8: The complete forest of a 5-loop graph contribution to the 2-point function of ϕ_6^3-theory. Red lines are the line of the graph, yellow is the first henged layer followed by green, light blue and dark blue for the final recursively henged layer

However, it is currently impossible to evaluate this graph as it is beyond the capabilities of Maple.

Exporting the output to a dedicated numerical integrator such as VEGAS seem to have problems with them, even for manifestly finite graphs as the ones in the previous section.

This leads to the conclusion that the problem is one of four areas:

- Although the Forestry is producing the correct number and type of subtractions, the combinatorics have gone wrong somewhere.
- The momentum-space integration is not canceling the divergences as it should
- There is a problem with Maples C code generator
- The problem lies with VEGAS, and cleverer ways will need to be found to manipulate the output functions such that they can be evaluated.

Chapter 5

Conclusion

5.1 Summary

With the ever increasing need for the calculation of higher loop order calculation to match the accuracy of current and future colliders, different approaches have been attempted to tame the difficulty in performing such calculations but so far very have proved to be reliable methods beyond a few specific examples and those that do are still based on Feynman graphs. The disadvantage of the Feynman graph approaches is factorial increase in the number of integrals needed to be calculated, which suggests the need for automation.

The method discussed in this thesis lends itself well to the process of automation of Feynman graph calculation. The method of henge decomposition [86, 87] makes all the UV divergences present in many Feynman graph integrals manifestly disentangled and as such allows the treatment of these divergences in the most straightforward manner. The use of BPHZ theorem allows the calculation of processes on a graph by graph basis and guarantees that these subtractions are the equivalent of counter terms in the lagrangian and as such eases the burden on any computation as the perturbative processes are more easily and naturally split up in terms of Feynman diagrams. The link between a fully henge decomposed Feynman graph and an ordered sector in the method of sector decomposition [100] provides an important link between these two powerful techniques, and allows the method of henge decomposition and the use of the good properties of parametric integrals such as guaranteed locality and the ability to combine propagators in a systematic way; without having to worry about the process of parameterization spoiling the disentangled UV divergence. For sector decomposition [70, 71, 72, 73, 85], the link to henge decomposition provides further confirmation that the subtractions made in the process are valid due to the henge decompositions' tie to the BPHZ theorem [90].

5.2 Outlook

The code presented in this thesis provides a good foundation for the implementation of a parameterized henge decomposition procedure into a working code. There are several ways to proceed from here; the most simple is to eliminate the last remaining bugs within the code which prevent the output from being analyzed via numerical integrators such as VEGAS. Further than that, the code presented in this thesis was written in Maple for the use of its symbolic manipulation, over several iterations of the code; the parts that needed the use of symbolic manipulation were replaced with routines based on the Galler-Fischer algorithm [95] and as such there is no longer any need for it to be in a language such as Maple. So a rewrite to a more suitable language would improve the performance of the code and allow the code to be written in more efficient way to remove some of the repetition in the code. Beyond theses things, the code could be extended to deal with IR divergences, tensor integrals with more than two external legs and incorporate methods for dealing with theories with γ_5's present.

Appendix A

Proof of BPHZ theorem using the Caswell-Kennedy method

Now that the equivalence between the henge decomposition representation of the BPHZ theorem has been shown to be the same as Bogoliubov's representation, a proof is needed that $RI(\mathcal{G})$ is made finite by local subtractions. The conjecture for this is

$$|RI_\lambda(\Theta)| \leq c \times \max(m,\lambda)(\lambda)^{\deg(\times)+0_+}$$

where 0_+ is an arbitrary small positive real constant, such that $x^{0_+} > (\ln x)^n$ for any n for sufficiently large x.

In order to show this it needs to be proved for two cases: the overall convergent and the overall divergent cases. Looking to the convergent case first, using the henge decomposition definition of $\bar{R}$ yields:

$$\left|\bar{R}I_\lambda(\mathcal{G})\right| = \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dk \left|i_k\left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right)\right| \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} |RI_k(\Theta)|$$

inserting the induction hypothesis derived earlier

$$\begin{aligned}
\left|\bar{R}I_\lambda(\mathcal{G})\right| &\leq c \times \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dk k_\ell^{\deg(\mathcal{D}/\mathcal{H})-2\omega} \prod_{\Theta\in\mathcal{H}(\mathcal{G},\ell)} k_\ell^{\deg(\Theta)+0_+} \\
&\leq \text{constant} \times \sum_{\ell\in\mathcal{G}} \int_\lambda^\infty dk k_\ell^{\deg(\mathcal{G})-2\omega+0_+} \\
&\leq \text{constant} \times \lambda^{\deg(\mathcal{G})+0_+} \qquad \text{(A.1)}
\end{aligned}$$

As this is for an overall convergent integral, through linearity $RI_\lambda(\mathcal{G}) = \bar{R}I_\lambda(\mathcal{G})$ and thus the conjecture has been established for the convergent case. Turning now to

the case for an integral with an overall divergent degree. As seen before, the Feynman integral $I(\mathcal{G})$ can be decomposed into a IR sensitive and a UV sensitive region $\int_0^\infty I_0(\mathcal{G}) = \int_0^\lambda I_\lambda(\mathcal{G}) + \int_\lambda^\infty I_\lambda(\mathcal{G})$. In order to prove the bounds on $|RI_\lambda(\mathcal{G})|$ the bounds have to be found on $|RI_0(\mathcal{G})|$ and on $|RI_0(\mathcal{G}) - RI_\lambda(\mathcal{G})|$.

The bound on the IR sensitive region is straight forward and is analogous to the method for the convergent Feynman integral

$$\begin{aligned}
|RI_0(\mathcal{G}) - RI_\lambda(\mathcal{G})| &= \left| \sum_{\ell \in \mathcal{G}} \int_0^\lambda d^{2\omega} k i_k \left(\mathcal{G}/\mathcal{H}(\mathcal{G},\ell)\right) \prod_{\Theta \in \mathcal{H}(\mathcal{G},\ell)} RI_k(\Theta) \right| \\
&\leq c \times \sum_{\ell \in \mathcal{G}} \int_0^\lambda d^{2\omega} k_\ell m^{\deg(\mathcal{G}/\mathcal{H})-2\omega} \prod_{\Theta \in \mathcal{H}} \max(m,\lambda)^{\deg(\Theta)+0_+} \\
&\leq c \times \sum_{\ell \in \mathcal{G}} \int_0^\lambda d^{2\omega} k_\ell m^{\deg(\mathcal{G})-2\omega+0_+} \\
&\leq c \times \lambda^{\deg(\mathcal{G})+0_+} \qquad \text{(A.2)}
\end{aligned}$$

for $\mathcal{G} < 0$

Recalling the Taylor expansion subtraction operator introduced earlier eq.(2.65), it is seen that an interesting property of the subtraction operator K is that it commutes with the differential operator ∂, although this property is not in general true for all sensible choices of subtraction (in this case minimal subtraction and Taylor series) it does hold. For Taylor series it is seen that

$$\partial T^{\deg(\mathcal{G})-1} \partial I(\mathcal{G}) = T^{\deg(\partial\mathcal{G})} \partial I(\mathcal{G})$$

as $\deg(\partial\mathcal{G}) = \deg(\mathcal{G}) - 1$, therefore $[\partial, T^{\deg}] = 0$. For minimal subtraction the operation acts on parameter space to isolate the pole, which means that $[\partial, K] = 0$ trivially. Both of these could have been guessed at from looking at the form of the Bogoliubov's representation of R eq.(2.67) and $\bar{R}$ eq.(2.68).

With this final property of subtraction operator K established, the bound on $|RI_0(\mathcal{G})|$ can be calculated by applying Taylor's theorem (2.65) on $\bar{R}I(\mathcal{G})$ to give

$$\begin{aligned}
\bar{R}I_0(\mathcal{G}) &= T^{\deg(\mathcal{G})} \bar{R}I_0(\mathcal{G}) + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \partial^{\deg(\mathcal{G}+1)} \bar{R}I_0(P_{\deg(\mathcal{G})+1}) \\
&= T^{\deg(\mathcal{G})} \bar{R}I_0(\mathcal{G}) + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \bar{R}\partial^{\deg(\mathcal{G}+1)} I_0(P_{\deg(\mathcal{G}+1)}) \\
&\qquad \text{(A.3)}
\end{aligned}$$

The sum of graphs in the second term in the above above equation $\partial^{\deg(\mathcal{G})+1} I_0(\mathcal{G})$

are convergent as $\deg\left(\partial^{\deg(\mathcal{G})+1} I_0(\mathcal{G})\right) = -1$ and as such this part of the integral is finite. As the integration is over a compact region this means that any divergence has to be in the polynomial of the first term. Using the henge decomposition definition of the R-operation (2.82), the first term is replaced by a finite polynomial in the external momenta as it is exactly $K\bar{R}I_0(\mathcal{G})$, so $\bar{R}$ will remove the subdivergences.

$$\begin{aligned} RI_0(\mathcal{G}) &= \bar{R}I_\lambda(\mathcal{G}) - K\bar{R}I_0(\mathcal{G}) \\ &= Q(p) + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} R\partial^{\deg(\mathcal{G}+1)} I_0(P_{\deg(\mathcal{G})+1}) \end{aligned} \tag{A.4}$$

The second term in above equation has its form due to $K\bar{R}\partial^{\deg(\mathcal{G})+1} I_0(\mathcal{G}) = 0$. Applying the inductive bound for the overall convergent integrand, where the polynomial $Q(p)$ satisfies the tree level bound, yieldst:

$$\begin{aligned} |RI_0(\mathcal{G})| &\leq Q(p) + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \left| R\partial^{\deg(\mathcal{G}+1)} I_0(P_{\deg(\mathcal{G}+1)}) \right| \\ &\leq c \times \max(m,p)(0)^{\deg(\mathcal{G})} + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \left| R\partial^{\deg(\mathcal{G}+1)} I_0(P_{\deg(\mathcal{G})+1}) \right| \\ &\leq c \times \max(m,p)(0)^{\deg(\mathcal{G})} + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \max(m,p)(0)^{-1+0_+} \\ &\leq c \times \max(m,p)(0)^{\deg(\mathcal{G})} + \int_{p_0}^{p} dp_1 \cdots \int_{p}^{\deg(\mathcal{G})} dp_{\deg(\mathcal{G})+1} \max(m,p)(0)^{-1+0_+} \\ &\leq c \times \max(m,p)(0)^{\deg(\mathcal{G})+0_+} \end{aligned} \tag{A.5}$$

Using the bounds for $|RI_0(\mathcal{G})|$ and $|RI_0(\mathcal{G}) - RI_\lambda(\mathcal{G})|$ the bounds for the overall divergent $|RI_\lambda(\mathcal{G})|$ is obtained.

$$\begin{aligned} |RI_\lambda(\mathcal{G})| &\leq |RI_0(\mathcal{G})| + |RI_0(\mathcal{G}) - RI_\lambda(\mathcal{G})| \\ &\leq c \times \max(m,p)(0)^{\deg(\mathcal{G})+0_+} + c \times \max(m,\lambda)(\lambda)^{\deg(\mathcal{G})+0_+} \\ &\leq c \times \max(m,\lambda)(\lambda)^{\deg(\mathcal{G})+0_+} \qquad \text{for} \quad \deg\mathcal{G} \geq 0 \end{aligned} \tag{A.6}$$

Appendix B

Common command calls in Maple

This appendix contains a few of the important and commonly used Maple commands in the Henge decomposition code in the main body of the thesis. An important type of functional programming used is the **functional operator**, which is called with $->$. This is special kind of procedure where the argument, which is on the left of the arrow is applied to an equation defined on the right of the arrow. The most basic example of this is:

```
f := x -> x^2;

f(2);
                                    4
```

where the arguments of the function "f" are squared.

Much of the data is in terms of lists, which is defined to be an ordered sequence and designated with square braces [], and sets, an unordered sequence with duplicate entries removed designated by curly braces {}. It is necessary to sometimes access individual elements, or operands, of these sequences and in order to do so the **op** command is used. For the set $S := \{1, 2, \ldots, n\}$, the first element is accessed via the command **op(1,S)**, likewise the second element is accessed with **op(2,S)**. This can be done for every element within the set S, however in the cases where the number of elements within the set are not constant, the end elements of the set can accessed with **op(-1,S)** for the nth operand, **op(-2,S)** for the $(n-1)$th term etc. Another useful command is that of **nops**, this counts the number of elements within the set or list.

The **map** command is widely used throughout the code and at its most basic level, applies a function to each operand of an expression. In the example below we have a

function f which takes the arguments of each operand of the expression $x^2 + y$ and the additional arguments a and b.

```
map(f,x^2+y,a,b);
```

```
f(x^2 , a, b) + f(y, a, b)
```

A second way in which the map command is used in this code is used in conjunction with the functional operator:

```
map(x->x^2+y,[a,b]);
```

```
[a ^2 + y, b^2  + y]
```

This technique takes the expression of $x^2 + y$ and constructs a sequence of expressions with the argument of x taking the value of the elements of the sequence $[a, b]$.

Debugging is done with the use of the $userinfo$ command, which is invoked with a non-negative integer, the procedure to which it belongs and the action which is to be taken. The most common way this is used is of the form:

```
userinfo(4, procname,
            sprintf("The argument of  %a is %a", f, g));
```

which when the $infolevel$ of the procedure is set to 4, it will print out the information in $sprintf$ whenever the procedure it is in is called.

Bibliography

[1] O. Greenberg, Phys. Rev. Lett. **13**, 598 (1964).

[2] Y. Han and Y. Nambu, Phys. Rev. B **139**, 1006 (1965).

[3] Y. Nambu, *A Systematics of Hadrons in Subnuclear Physics. In Preludes in Theoretical Physics, ed. A. de Shalit, H. Feshbach and L. Van Hove* (North Holland, Amsterdam,, 1966).

[4] D. Gross and F. Wilczek, Phys. Rev. Lett. **30**, 1343 (1973).

[5] H. Politzer, Phys. Rev. Lett. **30**, 1346 (1973).

[6] S. L. Glashow, Nucl. Phys. **22**, 579 (1961).

[7] A. Salam, Originally printed in *Svartholm: Elementary Particle Theory, Proceedings Of The Nobel Symposium Held 1968 At Lerum, Sweden*, Stockholm 1968, 367-377.

[8] S. Weinberg, Phys. Rev. Lett. **19**, 1264 (1967).

[9] P. W. Higgs, Phys. Lett. **12**, 132 (1964).

[10] P. W. Higgs, Phys. Rev. Lett. **13**, 508 (1964).

[11] P. W. Higgs, Phys. Rev. **145**, 1156 (1966).

[12] Dyson, phys. rev., 75 (1949) 486, 1736.

[13] Feynman, phys. rev., 74 (1948) 1430, 76 (1949) 769.

[14] Feynman, phys. rev., 74 (1948) 1430, 76 (1949) 769.

[15] Tomonaga, progr. theoret. phys. (kyoto), i (1946) 27; koba, tati and tomonaga, ibid., 2 (1947) ioi, 198; kanesawa and tomonaga, ibid., 3 (1948) i, ioi.

[16] Ito, koba and tomonaga, progr. theoret. phys. (kyoto), 3 (1948) 276; koba and takeda, ibid., 3 (1948) 407.

[17] Koba and tomonaga, progr. theoret. phys. (kyoto), 3 (1948) 290; tati and tomonaga, ibid., 3 (1948) 391.

[18] Particle Data Group, C. A. et al., Phys. Lett. **B667**, 1 (2008).

[19] ALEPH, , Phys. Rept. **427**, 257 (2006), hep-ex/0509008.

[20] EWWG, Plots forwinter 2009, http://lepewwg.web.cern.ch/LEPEWWG/plots/winter2009/.

[21] WMAP, D. S. et al., Astrophys. J. Suppl. **148**, 175 (2003), astro-ph/0302209.

[22] D. G. Bertone and J. Silk, Phys. Rept. **405**, 279 (2005), hep-ph/0404175.

[23] Workshop on monte carlo generator physics for run ii at the tevatron, 18-20 apr 2001, batavia, illinois.

[24] Workshop: Physics at tev colliders, les houches, 2-20 may 2005, http://lappweb.in2p3.fr/conferences/LesHouches/Houches2005/.

[25] C. B. et al., (2006), hep-ph/0604120.

[26] Physics at tev colliders, les houches, 11-29 june 2007, http://lappweb.in2p3.fr/conferences/LesHouches/Houches2007/.

[27] NLO Multileg Working Group, Z. e. a. Bern, (2008), 0803.0494.

[28] T. Gehrmann, Tools for NNLO QCD calculations, http://ilcagenda.linearcollider.org, LCWS/ILC Workshop, Desy 2007.

[29] S. Heinemeyer, Prepared for International Conference on Linear Colliders (LCWS 04), Paris, France, 19-24 Apr 2004.

[30] T. Binoth, (2009), 0903.1876.

[31] A. Denner, S. Dittmaier, M. Roth, and L. H. Wieders, Phys. Lett. **B612**, 223 (2005), hep-ph/0502063.

[32] S. D. A. Bredenstein, A. Denner and S. Pozzorini, **08**, 108 (2008), 0807.1248.

[33] T. Binoth *et al.*, (2008), 0807.0605.

[34] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov, and G. Zanderighi, JHEP **01**, 012 (2009), 0810.2762.

[35] C. F. e. a. Berger, (2009), 0902.2760.

[36] T. Reiter, (2009), 0903.0947.

[37] S. Catani and M. H. Seymour, Nucl. Phys. **B485**, 291 (1997), hep-ph/9605323.

[38] S. Catani, S. Dittmaier, M. H. Seymour, and Z. Trocsanyi, Nucl. Phys. **B627**, 189 (2002), hep-ph/0201036.

[39] D. A. Kosower, Phys. Rev. **D71**, 045016 (2005), hep-ph/0311272.

[40] A. Daleo, T. Gehrmann, and D. Maitre, JHEP **04**, 016 (2007), hep-ph/0612257.

[41] A. G.-D. Ridder and M. Ritzmann, (2009), 0904.3297.

[42] G. Passarino and M. J. G. Veltman, Nucl. Phys. **B160**, 151 (1979).

[43] O. V. Tarasov, Phys. Rev. **D54**, 6479 (1996), hep-th/9606018.

[44] R. E. Cutkosky, J. Math. Phys. **1**, 429 (1960).

[45] Z. Bern, L. J. Dixon, D. C. Dunbar, and D. A. Kosower, Nucl. Phys. **B425**, 217 (1994), hep-ph/9403226.

[46] Z. Bern, L. J. Dixon, D. C. Dunbar, and D. A. Kosower, Nucl. Phys. **B435**, 59 (1995), hep-ph/9409265.

[47] E. Witten, Commun. Math. Phys. **252**, 189 (2004), hep-th/0312171.

[48] Z. Bern, L. J. Dixon, and D. A. Kosower, Annals Phys. **322**, 1587 (2007), 0704.2798.

[49] T. van Ritbergen, J. A. M. Vermaseren, and S. A. Larin, Phys. Lett. **B400**, 379 (1997), hep-ph/9701390.

[50] M. Czakon, Nucl. Phys. **B710**, 485 (2005), hep-ph/0411261.

[51] S. Laporta and E. Remiddi, Phys. Lett. **B379**, 283 (1996), hep-ph/9602417.

[52] T. Kinoshita and M. Nio, Phys. Rev. **D70**, 113001 (2004), hep-ph/0402206.

[53] T. Kinoshita and M. Nio, Phys. Rev. **D73**, 053007 (2006), hep-ph/0512330.

[54] T. Aoyama, M. Hayakawa, T. Kinoshita, and M. Nio, PoS **RADCOR2007**, 025 (2007).

[55] T. Aoyama, M. Hayakawa, T. Kinoshita, and M. Nio, Phys. Rev. **D78**, 113006 (2008), 0810.5208.

[56] P. A. Baikov, K. G. Chetyrkin, and C. Sturm, Nucl. Phys. Proc. Suppl. **183**, 8 (2008), 0807.1646.

[57] S. Moch, J. A. M. Vermaseren, and A. Vogt, Nucl. Phys. **B688**, 101 (2004), hep-ph/0403192.

[58] A. Vogt, S. Moch, and J. A. M. Vermaseren, Nucl. Phys. **B691**, 129 (2004), hep-ph/0404111.

[59] K. G. Chetyrkin, Nucl. Phys. **B710**, 499 (2005), hep-ph/0405193.

[60] K. G. Chetyrkin and F. V. Tkachov, Nucl. Phys. **B192**, 159 (1981).

[61] V. A. Smirnov, Phys. Lett. **B460**, 397 (1999), hep-ph/9905323.

[62] J. B. Tausk, Phys. Lett. **B469**, 225 (1999), hep-ph/9909506.

[63] M. Czakon, Comput. Phys. Commun. **175**, 559 (2006), hep-ph/0511200.

[64] T. Gehrmann and E. Remiddi, Nucl. Phys. **B580**, 485 (2000), hep-ph/9912329.

[65] S. Laporta, Int. J. Mod. Phys. **A15**, 5087 (2000), hep-ph/0102033.

[66] A. V. Smirnov and V. A. Smirnov, JHEP **01**, 001 (2006), hep-lat/0509187.

[67] K. Hepp, Commun. Math. Phys. **2**, 301 (1966).

[68] N. N. Bogoliubov and O. S. Parasiuk, Acta Math. **97**, 227 (1957).

[69] M. Roth and A. Denner, Nucl. Phys. **B479**, 495 (1996), hep-ph/9605420.

[70] T. Binoth and G. Heinrich, Nucl. Phys. **B585**, 741 (2000), hep-ph/0004013.

[71] T. Binoth and G. Heinrich, Nucl. Phys. **B680**, 375 (2004), hep-ph/0305234.

[72] T. Binoth and G. Heinrich, Nucl. Phys. **B693**, 134 (2004), hep-ph/0402265.

[73] T. Binoth, G. Heinrich, T. Gehrmann, and P. Mastrolia, Phys. Lett. **B649**, 422 (2007), hep-ph/0703311.

[74] V. A. Smirnov, *Renormalization and asymptotic expansions* (Birkhaeuser(Progress in physics 14, 380 p.) Basel, Switzerland, 1991).

[75] V. A. Smirnov, *Evaluating Feynman integrals* (Springer Tracts Mod. Phys. 211, 2004).

[76] J. Zinn-Justin, *Quantum Field Theory and Critical Phenomena* (Oxford University Press, 1989).

[77] A. D. Kennedy, Unpublished notes.

[78] J. Collins, *Renormalization* (Cambridge University Press, 1984).

[79] W. Pauli and F. Villars, Rev. Mod. Phys. **21**, 434 (1949).

[80] H. J. Rothe, *Lattice Gauge Theories: An Introduction* (World Scientific Publishing Co Pte Ltd, 2005).

[81] G. 't Hooft and M. J. G. Veltman, Nucl. Phys. **B44**, 189 (1972).

[82] G. 't Hooft, Nucl. Phys. **B61**, 455 (1973).

[83] P. Cvitanović, Group theory: Birdtracks, lie's, and exceptional groups, http://birdtracks.eu/.

[84] N. Nakanishi, *Graph Theory and Feynman Integrals* (Gordon and Breach, New York, 1971).

[85] G. Heinrich, Int. J. Mod. Phys. **A23**, 1457 (2008), 0803.4177.

[86] W. E. Caswell and A. D. Kennedy, Phys. Rev. **D25**, 392 (1982).

[87] W. E. Caswell and A. D. Kennedy, Phys. Rev. **D28**, 3073 (1983).

[88] A. D. Kennedy, Phys. Rev. **D26**, 1936 (1982).

[89] A. D. Kennedy, A simple proof of the bph theorem, http://www2.ph.ed.ac.uk/ adk/bph-slides.ps.gz.

[90] A. D. Kennedy, (1996), hep-th/9612113.

[91] C. Bogner and S. Weinzierl, Comput. Phys. Commun. **178**, 596 (2008), 0709.4092.

[92] K. Geddes and G. Gonnet, Maple 11, http://www.maplesoft.com/products/maple/.

[93] P. Nogueira, Qgraf, http://cfif.ist.utl.pt/ paulo/qgraf.html.

[94] D. Knuth, *The Art of Computer Programming I* (Addison-Wesley, 1997).

[95] B. Galler and M. Fischer, Commun. ACM **7(5)**, 301 (1964).

[96] G. Lepage, Journal of Comput. Phys. **27**, 192 (1978).

[97] Various, Numerical recipes, http://www.nr.com/ .

[98] M. Srednicki, *Quantum field theory* (Cambridge University Press, 2007).

[99] A. I. Davydychev, J. Phys. **A25**, 5587 (1992).

[100] T. B. A.D. Kennedy and T. Rippon, (2007), 0712.1016.