Reduction of two-loop Feynman integrals

Rob Verheyen

July 3, 2012

Contents

Chapter 1

The Fundamentals at One Loop

1.1 Introduction

Experimental particle physics is currently a very active field of research. Particle colliders produced or are producing large amounts of data to be analysed. To interpret these data, comparison with highly accurate theoretical results is required. In particular, in order to calculate cross-sections, many contributions from different Feynman diagrams have to be computed. The corresponding Feynman integrals become progressively complex as more loops appear in the associated Feynman diagrams. Many authors have attempted to reduce several of these integrals to more manageable forms. One of these methods will be studied in this thesis.

We start by looking at Feynman diagrams with a single loop. An example of such a diagram is given in Figure 1.1.

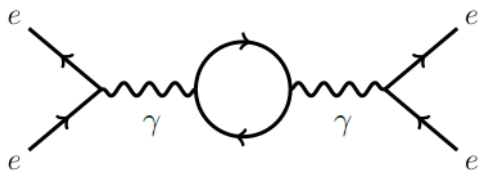

Figure 1.1: A general one-loop Feynman diagram. The particles in the loop can be several things, such as an electron-positron pair

The general form of the relevant integral is

$$\int \frac{A}{D_1 D_2 ... D_n} d^4 q \tag{1.1}$$

The numerator A can have many different forms. For our reduction, we will only focus on the denominator and leave A the same. The typical shape of D_i is $D_i = (q+p_i)^2 - m_i^2$. The integration variable q^μ is the loop momentum. As this momentum is unknown within a loop, it has to be accounted for by integrating over all possible values. The constants p_i^μ and m_i correspond to the external momentum[1] and mass of particle i. D_i is then what is called the *propagator* of particle i. From now on, we refer to n (the total number of propagators) as the *order* of the integrand.

These integrals do not necessarily correspond with a single Feynmann diagram. Instead, we will consider the reduction in the most general way possible. To visualize the integral, we use an *integrand graph* (or iGraph). Figure 1.2 is an example of such an iGraph.

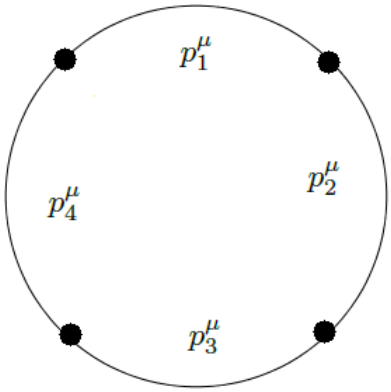

Figure 1.2: iGraph with 4 propagators

In general, there is no momentum conservation along the loop of the iGraph. For instance, our integral may be a combination of several integrals corresponding to different diagrams. In that case, the resulting integral no longer corresponds with any physical event, but it can still be reduced.

1.2 Reducing the One-loop Case

Our goal is to reduce the above integrals into a number of integrals that are all easier to compute. We would therefore like to split up the integrand into several smaller fractions. Consider an iGraph of order n. A reduction of the corresponding integral would look like

$$\frac{1}{D_1 D_2 ... D_n} = \frac{f_1(q)}{D_2 D_3 ... D_n} + \frac{f_2(q)}{D_1 D_3 ... D_n} + ... + \frac{f_n(q)}{D_1 D_2 D_3 ... D_{n-1}} \tag{1.2}$$

[1] Another, fixed momentum

If we can indeed find functions $f_1(q^\mu), f_2(q^\mu), \ldots, f_n(q^\mu)$ that satisfy equation (1,2), we have found a reduction of the original integral. The reduction is just a sum of integrands corresponding with iGraphs of order n-1. Rewriting (1,2), we get

$$D_1 f_1(q) + D_2 f_2(q) + ... + D_n f_n(q) = 1 \tag{1.3}$$

Where we multiplied by $D_1 D_2 \ldots D_n$. We can attempt to reduce the integrand trivially by just setting $f_1(q)$, $f(q)$, ..., $f(q)$ to constants.

$$f_i(q) = c_i \tag{1.4}$$

Filling in and substituting $D_i = (q + p_i)^2 - m_i^2$ in equation (1,3), we find

$$(q^2 + p_1^2 + 2q_\mu p_1^\mu - m_1^2) \cdot c_1 + \ldots + (q^2 + p_n^2 + 2q_\mu p_n^\mu - m_1^2) \cdot c_n = 1 \tag{1.5}$$

Writing out equation (1.5)

$$q^2 \sum_{i=1}^{n} c_i + 2q_\mu \sum_{i=1}^{n} c_i p_i^\mu + \sum_{i=1}^{n} c_i \cdot (p_i^2 - m_i^2) = 1 \tag{1.6}$$

We can now easily see that this equation actually has the general form

$$a \cdot q^2 + b^\mu q_\mu + d = 1 \tag{1.7}$$

Where a and d are just constants, and b^μ is a constant four-vector. We now require this equation to hold for any q^μ. This immediately implies that $a = 0$, $b^\mu = 0$ and $d = 1$. To see this, consider a situation where for some q^μ, equation (1,7) holds for nonzero a. We can now always choose a different q^μ such that $a \cdot q^2 + d = 0$, violating equation (1.7). The same argument can be used for the individual components of b^μ, and even for combinations of nonzero a and b^μ, as we can always change and scale $q^\mu$to make (1.6) vanish. Thus, equation (1.7) is in fact a system of equations

$$a = \sum_{i=1}^{n} c_i = 0 \tag{1.8}$$

$$b^\mu = \sum_{i=1}^{n} c_i p_i^\mu = 0 \tag{1.9}$$

$$d = \sum_{i=1}^{n} c_i \cdot (p_i^2 - m_i^2) = 1 \tag{1.10}$$

These six[2] equations yield constraints that apply to the values of c_i. As they are all linear in c_i, we can fulfill them if we have six or more c_i, or iGraphs of order 6 or higher. In general dimension d, we can thus reduce the iGraph if it has order $d + 2$ or higher. Of course, if we want to reduce iGraphs of order

[2]Equation (1.9) actually represents 4 equations

larger than $d+2$, we can just keep reducing untill we hit the situation where every iGraph has $d+1$ propagators.

It should be noted that this process does not yet solve the coefficients c_i. For that, one has to calculate them through a process we will see later on and check if they hold true for any value of q^μ. This test is called the '1 = 1 test', which refers to filling in equation (1.3) with random values for q^μ, and checking if it adds up to 1. We can use random values for q^μ because the reduction should be valid for every value of q^μ.

We will now consider how we can improve this method by expanding the functions $f_1(q)$, $f(q)$, ..., $f(q)$ with terms of higher order in q$^\mu$. However, before expanding the functions with larger and larger terms, one might wonder what the limit of this reduction is.

1.3 Unitarity

We have seen that we can already reduce iGraphs in dimension d to integrals of order $d+1$ by only using constant functions $f_1(q)$, $f(q)$, ..., $f(q)$. One would expect the reduction to go further if we expand the functions. Unfortunately, there is a fundamental limit on how far we can carry reductions with this method, related to unitarity.

Consider a Feynman integral of order d. To further reduce this integral, the relevant equation to solve is

$$D_1(q)f_1(q) + D_2(q)f_2(q) + \ldots + D_d(q)f_d(q) = 1 \tag{1.11}$$

Where D_i has explicitly been written as a function of q. But q^μ was the integration variable of the Feynman integral. Therefore, we can freely choose every component q^μ without changing the content of the integral. For an iGraph of order d in particular, we can choose q^μ in such a way that every D_i vanishes. This would violate equation (1.11), and as such the conclusion is that no reduction further than order d can be possible.

1.4 Expanding $f_i(q)$

We now expand the functions $f_1(q)$, $f(q)$, ..., $f(q)$ by adding linear terms

$$f_i(q^\mu) = c_i + \sum_{j=1}^{d} c_{i,j} q^j \tag{1.12}$$

The additional terms are linear in every component of q. Instead of n, there are now $(1+d)\cdot n$ coefficients to fulfill equations (1.8) through (1.10). Unfortunately, these equations now also recieve extra terms.

$$(q^2+p_1^2+2q_\mu p_1^\mu-m_1^2)\cdot(c_1+\sum_{j=1}^{d}c_{1,j}q^j)+\ldots+(q^2+p_n^2+2q_\mu p_n^\mu-m_1^2)\cdot(c_n+\sum_{j=1}^{d}c_{n,j}q^j)=1 \tag{1.13}$$

In a similar fashion as before, equation (1.13) splits up into several parts governed by the powers of q.

$$q^2\sum_{i=1}^{n}c_i=0 \tag{1.14}$$

$$2q_\mu\sum_{i=1}^{n}(c_i p_i^\mu+c_i^\mu\cdot(p_i^2-m_i^2))=0 \tag{1.15}$$

$$q^2q^\mu\sum_{i=1}^{n}c_{i,\mu}=0 \tag{1.16}$$

$$2q_\mu q^\nu\sum_{i=1}^{n}c_{i,\nu}p_i^\mu=0 \tag{1.17}$$

$$\sum_{i=1}^{n}c_i\cdot(p_i^2-m_i^2)=1 \tag{1.18}$$

Every one of the equations determines a value of our coefficients. Therefore, to count how many restrictions on the coefficients there are, we just have to count the amount of tensor structures. In this situation, we see that equation (1.14) is already contained in equation (1.17), and should therefore not be counted.[3] . In total, we then have $0+d+d+\frac{d\cdot(d+1)}{2}+1=\frac{1}{2}d^2+\frac{5}{2}d+1$ independent tensor structures.[4]

Reduction with trivial terms was possible until iGraphs of order $d+1$, and unitarity restricts reduction of order d. The only relevant reduction left is that of order $d+1$ to order d. Filling in $n=d+1$ in the expression for the amount of coefficients, we find $(d+1)\cdot(d+1)=d^2+2d+1$. Comparing this with the amount of tensor structures, it is clear that for $d>1$ the amount of coefficients surpasses the amount of tensor structures. It would seem that the reduction is possible.

However, we have to consider that we may be unable to fulfill equations (1.14) through (1.18) with the coefficients we have. The reason for this is that it might happen that coefficients only appear in certain combinations. Consider a system of equations

[3] $q^2$is in fact spread out over the equations in (1.17) where $\mu=\nu$

[4] Equation (1.17) only accounts for $\frac{d\cdot(d+1)}{2}$ tensor structures because it is symmetric.

$$x + 2y - z = 3$$

$$-2x - 4y + 3z = 5$$

If we rewrite the second equation as $2(x+2y)+3z = 5$, it is clear that variables x and y only appear in a set combination, and should be considered as a single variable. If this happens, one of the coefficients is called *dependent* on the other. The same may happen to the system (1.14)-(1.18), resulting in dependent c's. In fact, it *has* to happen, because any coefficients that would be left over after applying (1.14)-(1.18) can be chosen freely, and are therefore dependent. We conclude that the amount of independent coefficients has to be equal to the amount of independent tensor structures. If this is the case, the reduction is possible.

In [1] the amount of independent coefficients is calculated using a method called *cancellation probing*. The results presented there show that reduction of iGraphs of order $d+1$ is indeed possible with linear terms. This completely solves the reduction of Feynman integrals in a single loop. We will be able to reproduce this result with a method used for reduction of integrals with two or more loops.

Chapter 2

Two and more loops

2.1 The Two-Loop Case

Now that we finished decomposition of iGraphs with a single loop, we are ready to consider diagrams with two or more loops. While these diagrams yield higher order terms, making them smaller than those of the one-loop diagrams, they still represent contributions that are not negligible. The general form of the Feynman integral for two loops is similar to equation (1.1)

$$\int \frac{1}{D_1 D_2 ... D_n} d^4 l_1 d^4 l_2 \tag{2.1}$$

There are now two separate loop momenta l_1 and l_2 that both have to be integrated over. The expression for the propagator changes to $D_i = (l_1 + p_i)^2 - m_i^2$, $D_i = (l_2 + p_i)^2 - m_i^2$ or $D_i = (l_1 + l_2 + p_i)^2 - m_i^2$. That is, a propagator can have momenta l_1 , l_2 or both l_1 and l_2. A two-loop iGraph must have at least one of each of these propagators, If one is missing, we can consider the iGraph as a combination of two one-loop iGraphs. We can uniquely represent the corresponding iGraphs with the expression (n_1, n_2, n_3), where n_1 represents the amount of propagators with l_1, n_3 those with l_2 and n_2 the amount of shared propagators. Then $n = n_1 + n_2 + n_3$ is the total number of propagators. Figure 2.1 displays an example of an iGraph with two loops.

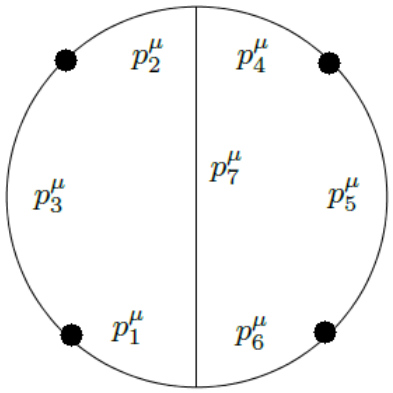

Figure 2.1: An iGraph with configuration (3,2,3)

When we try to reduce two-loop iGraphs, we will generally use functions $f_1(l_1, l_2)$, $f_2(l_1, l_2)$, ..., $f_n(l_1, l_2)$ that have terms depending on both l_1 and l_2. For example, if we would try reduction with linear terms, the functions would consist of terms that are linear in l_1 and other terms that are linear in l_2.

2.2 The Relevant iGraphs

Fortunately, the amount of different reductions to consider is limited. First of all, we notice that the integral in equation (2.1) is just a product of the inverse of propagators. Therefore, if either n_1 or n_3 is $r > d$, we can simply write the integral as

$$\int \frac{1}{D_1 D_2 ... D_r} \cdot \frac{1}{D_{r+1} \dots D_n} d^4 l_1 d^4 l_2 \tag{2.2}$$

We now easily see that because the first r propagators only depend on one loop momentum, we can simply apply our knowledge of the reduction of one-loop integrals, reducing the expression to

$$\int \left\{ \frac{f_1(l_i^\mu)}{D_2 D_3 ... D_r} + \frac{f(l_i^\mu)}{D_1 D_3 ... D_r} + \dots + \frac{f_n(l_i^\mu)}{D_1 D_2 D_3 ... D_{r-1}} \right\} \cdot \frac{1}{D_{r+1} \dots D_n} d^4 l_1 d^4 l_2 \tag{2.3}$$

When the relevant loop momentum is l_i. In fact, because we can freely choose the path of our loop momentum along the iGraph[1], we can use the same procedure in case $n_2 > d$ by just redefining how l_1 and l_2 move along the iGraph. These simple observations immediately tell us that any two-loop iGraph of order $n > 3d$ is reducable.

[1] As long as it's a closed loop

Furthermore, we should consider unitarity for the two-loop iGraphs. Rewriting equation (1.11), we find

$$\begin{aligned}1 = D_1(l_1)f_1(l_1) + \ldots + D_{n_1}(l_1)f_{n_1}(l_1) + \\ D_{n_1+1}(l_1+l_2)f_{n_1+1}(l_1+l_2) + \ldots + D_{n_1+n_2}(l_1+l_2)f_{n_1+n_2}(l_1+l_2) + \\ D_{n_1+n_2+1}(l_2)f_{n_1+n_2+1}(l_2) + \ldots + D_n(l_2)f_n(l_1) \quad (2.4)\end{aligned}$$

If all $n_i \leq d$, we can again shift l_1 and l_2 such that $2d$ of the propagators vanish. This implies that equation (2.4) cannot be true for all l_1 and l_2 if we have $n = 2d$. Therefore, reduction is not possible with this method. We conclude that we are looking to reduce two-loop iGraphs between orders $3d$ and $2d$.

2.3 Trivial Terms and Beyond

We again start with functions that are just constants

$$f_i(l_1 + l_2) = c_i \quad (2.5)$$

Analogously to the previous process that resulted in equation (1.3), we find

$$\sum_{i=1}^{n_1} c_i D_i(l_1) + \sum_{i=n_1+1}^{n_1+n_2} c_i D_i(l_1+l_2) + \sum_{i=n_1+n_2+1}^{n} c_i D_2(l_2) = 1 \quad (2.6)$$

Again, because equation (2.6) has to hold for every l_1 and l_2 , we can count the number of tensor structures to find the amount of restrictions on the coefficients. The possibilities are l_1^μ, l_2^μ, l_1^2, l_2^2, $l_{1\mu}l_2^\mu$ and the constant term. In total, this adds up to $d+d+1+1+1+1 = 2d+4$ conditions for the coefficients c_i. Therefore, for every configuration, an iGraph of order $2d+4$ can be decomposed to iGraphs of order $2d+3$.

In [1], it is shown that functions with linear terms

$$f_i(l_1 + l_2) = c_i + \sum_{j=1}^{d} a_{i,j} l_1^j + \sum_{j=1}^{d} b_{i,j} l_2^j \quad (2.7)$$

can be used to reduce iGraphs up to order $2d+1$. Additionally, quadratic terms

$$f_i(l_1+l_2) = c_i + \sum_{j=1}^{d} a_{i,j} l_1^j + \sum_{j=1}^{d} b_{i,j} l_2^j + \sum_{j,k=1}^{d} d_{i,jk} l_1^j l_1^k + \sum_{j,k=1}^{d} e_{i,jk} l_1^j l_2^k + \sum_{j,k=1}^{d} f_{i,jk} l_2^j l_2^k \quad (2.8)$$

allow for reduction to order $2d$ exclusively for $d = 2$. When adding cubic terms

$$f_i(l_1+l_2) = c_i+\sum_{j=1}^{d} a_{i,j}l_1^j+\sum_{j=1}^{d} b_{i,j}l_2^j+\sum_{j,k=1}^{d} d_{i,jk}l_1^j l_1^k+\sum_{j,k=1}^{d} e_{i,jk}l_1^j l_2^k+\sum_{j,k=1}^{d} f_{i,jk}l_2^j l_2^k+$$
$$\sum_{j,k,m=1}^{d} g_{i,jkm}l_1^j l_1^k l_1^m+\sum_{j,k,m=1}^{d} h_{i,jkm}l_1^j l_1^k l_2^m+\sum_{j,k,m=1}^{d} i_{i,jkm}l_1^j l_2^k l_2^m+\sum_{j,k,m=1}^{d} j_{i,jkm}l_2^j l_2^k l_2^m \tag{2.9}$$

it was shown that reduction to order $2d$ was possible for $d = 3$ as well. Additionally, it was suspected that cubic terms are also sufficient to reduce $d = 4$ to $2d$. This was not proven, because the numerical method used in [1] met its limits.

2.4 Solving the Coefficients

In order to tackle the problem of reducing iGraphs of order $2d+1$ for $d > 3$, we first introduce the general method of finding the required coefficients. We have already seen that we should take into account that equation (2.4) has to hold for any l_1 and l_2. To do this, we randomly generate a vector l_1^μ and l_2^μ for every coefficient. For $d = 4$ and reduction of an iGraph of order $2d + 1$ with cubic terms, this requires 1485 randomly generated l_1 and l_2. We then use random, but fixed values for m_i and p_i^μ to fill in equation (2.4) .[2] We use random values for m_i and p_i^μ because this eliminates any chance of picking values that might lead to special cases. If that occurs, reduction may be possible, while it is not in the general case. Of course, we are just trying to show that a reduction is possible. If one is trying to calculate the coefficients for specific values of m_i and p_i^μ, those would have to be used.

It is now possible to formulate the problem as a system of linear equations that have to be solved.

$$Ax = b \tag{2.10}$$

Where x represents the vector with all coefficients, A is the matrix contains the randomly generated set of linear equations we produced, and b is just a column vector with 1 everywhere. In order to prove reductibility, several conditions have to be met.

2.4.1 Solvability

First of all, the system of equations has to be solvable. To check the solvability of a system of equations, one usually computes matrix ranks. A matrix is solvable if and only if

$$rk(A) = rk(A|b) \tag{2.11}$$

[2] The same values of m_i and p_i^μ for every different value of l_1 and l_2

where $A|b$ means the matrix $\begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \end{pmatrix}$

To see this, note that when computing the product Ax, we are just making linear combinations of the columns of A. That is, the columns of A generate the image of A. If $Ax = b$ is solvable, the image of A contains b and it can be expressed as linear combination of columns of A. But then, if one adjoins b as a column of A, the rank of the matrix does not change. If we are using numerical methods to solve systems of equations, this condition can be used to check if there is a point in attempting to find a solution.

2.4.2 Matrix Rank and Tensor Structures

The second condition confirms the reductibility of the iGraph. It was mentioned before that the number of independent coefficients has to be equal to the number of tensor structures. If this is the case, equation (2.4) is solvable and the reduction is possible. In our matrix notation, the notion of independent coefficients relates to the rank of the matrix. If every single one of our coefficients would be independent, it would mean the system A fully determines them. If there are dependent coefficients, this has to be reflected in A in the form of linear dependance in the matrix rows. It just means that some of the equations in A are linear combinations of others, and therefore give no additional information about the solution of the coefficients. Every dependent row therefore corresponds with a dependent coefficient, and to check reductibility, we need to compare the matrix rank with the amount of tensor structures.

One might now wonder what would happen if we added even more equations than we already have by just generating more random values for l_1 and l_2. For every single l_1 and l_2, the new equation would just be linearly dependent on those we already had. This has to be the case, because if it was not, we could keep adding equations untill the system is overdetermined, and no reduction would be possible at all.

Counting the amount of tensor structures is a straightforward exercise. One just writes down all possible combinations[3] and substracts all structures that are contained in others. A formula for general dimension d is given in [1] for the case of cubic terms with $d > 2$.[4]

$$T(d) = \frac{2}{3}d^4 + \frac{22}{3}d^3 + \frac{71}{6}d^2 + \frac{1}{6}d + 1 \tag{2.12}$$

[3] Similar to what we did in (1.7)

[4] For $d = 2$ the amount of tensor structures turned out to be slightly lower than expected. However, this only occurs for $d = 2$

2.4.3 Solving the Coefficients

Obviously, the last requirement is to solve and test the coefficients. Several methods of solving systems of linear equations exist, which we will deal with in chapter 3. To test our solution, we need to require that the coefficients obey equation (2.4) for any value of l_1 and l_2. Numerically, we can test this by generating new random values for l_1 and l_2 and filling them in in equation (2.4) together with the solved coefficients. If the result is equal to one for several randomly generated l_1 and l_2, we can safely say we have found a reduction.

Chapter 3

Methods of solving linear systems

3.1 Gaussian Elimination

The first and most straightforward method of solving systems of linear equations we try is Gaussian elimination. We write down the matrix A|b as

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & 1 \end{pmatrix}$$

which is just the matrix corresponding to the system of equations we introduced earlier. Adjointed is the column of solutions, which is just 1 for every equation. We now perform elementary row operations to reduce the matrix to upper triangular form

$$\begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} & 1 \\ 0 & \alpha_{22} & \cdots & \alpha_{2n} & \alpha_{2(n+1)} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & \alpha_{nn} & \alpha_{n(n+1)} \end{pmatrix}$$

where α_{ii} indicates changed values from a_{ii}. We can see this process as systematically adding and substracting multiples of equations from others.

When the matrix is singular, as is the case here, pivoting has to be used to fully reduce the matrix. The process of elimination includes division by the elements on the diagonal. If these values are zero, the elimination fails. We can prevent this by interchanging rows when necessary. If there is no available row with a nonzero element on the relevant diagonal, the row should just be skipped.

When the matrix is in upper triangular form, we can aquire the solution to our coefficients by backward substitution. One uses the last row to determine the last coefficient, then substitutes it to solve the second to last equation. This can be repeated untill all coefficients are solved.
Additionally, the matrix rank is equal to the amount of nonzero elements on the diagonal. This becomes apparent when performing backward substitution. At any time, when a diagonal element is zero, the corresponding coefficient is not determined by the equation, and is therefore linearly dependent.

While Gaussian elimination is one of the fastest ways to solve a system of equations, it has turned out to be numerically unstable. Reasons for this include the large amount of divisions that have to be successively computed on the elements of the matrix. Our algorithm needs a way to determine if a diagonal element is zero. Inevitably, because of stacking rounding errors, this requires a limit on a low number below which we consider numbers as zero. This is a slippery slope for matrices, as more rounding errors requires a higher limit, which induces more errors. We will use a different method to both calculate the matrix rank and solve the system of equations.

3.2 QR Decomposition

3.2.1 Existence

Instead, we look at a process called QR decomposition. It allows us to decompose any matrix A as

$$A = QR \tag{3.1}$$

where Q is an orthogonal matrix and R is an upper triangular matrix. We first prove that this decomposition always exists.

Consider an arbirary matrix A. For any vector y, call $y^\star = Ay$. Then $y^T A^T Ay = y^{\star T} y^\star = |y^\star|^2 > 0$. Therefore the matrix $A^T A$ is always positive definite. Additionally, $A^T A$ is symmetric, because $(A^T A)^T = A^T A^{T,^T} = A^T A$. We now need to prove that we can write the matrix $A^T A = R^T R$ where R is upper triangular and has positive diagonal elements.

This proof is by induction in the order of the matrix $A^T A$, which we call B. If the order is 1, $A = R$ works as an upper triangular matrix. Let's now assume we can write $B^\star = R^{\star T} R^\star$ for $B^\star$ of order $n-1$. Then, we create the matrix B of order n

$$B = \begin{pmatrix} B^\star & b \\ b^T & \beta \end{pmatrix} \tag{3.2}$$

In this notation, $B^\star$ is the previous matrix of order $n-1$. Therefore, b is a vector and β is a scalar. This notation is possible because B is symmetric.
Given the upper triangular matrix $R^\star$, we can similarly choose R as

$$R = \begin{pmatrix} R^\star & r \\ 0 & \rho \end{pmatrix} \tag{3.3}$$

which is still upper triangular. Now, we need to show that R exists and is unique as a decomposition for B. We write out $B = R^T R$ explicitly.

$$\begin{pmatrix} B^\star & b \\ b^T & \beta \end{pmatrix} = \begin{pmatrix} R^{\star T} & 0 \\ r^T & \rho \end{pmatrix} \begin{pmatrix} R^\star & r \\ 0 & \rho \end{pmatrix} \tag{3.4}$$

This results in the following 4 equations

$$B^\star = R^{\star T} R^\star \tag{3.5}$$

$$b = R^{\star T} r \tag{3.6}$$

$$b^T = r^T R^\star \tag{3.7}$$

$$\beta = r^T r + \rho^2 \tag{3.8}$$

We do not have to worry about equation (3.5), as it is covered by our initial inductive assumption. Obviously, equation (3.7) is just the transpose of equation (3.6). Additionally, it follows that $r = R^{T^{-1}} b$, which uniquely determines r. Equation (3.8) yields $\rho = \sqrt{\beta - r^T r}$ which is unique and real[1] as a consequence of the positive definitiveness of the matrix B. Therefore, through induction, writing $A^T A = B = R^T R$ with R upper triangular is possible for any A.

Now, we can choose $Q = AR^{-1}$. Q is then orthogonal as $Q^T Q = R^{-1^T} A^T A R^{-1} = R^{-1^T} R^T R R^{-1} = I$. Obviously, $QR = AR^{-1}R = A$, and therefore, we have found a decomposition $A = QR$.

When the matrix is decomposed, the matrix rank is again equal to the amount of nonzero diagonal elements in R. Filling in the substitution for the matrix equation (2.13)

$$Rx = Q^T b$$

we can again find the solution for x by backward substitution.

There are several ways to obtain matrices Q and R, including the well-known Gram-Schmidt process. Numerically, the most stable method uses Householder transformations[2].

[1] And obviously positive

[2] Also called Householder reflections

3.2.2 Householder Transformations

The method of Householder transformations uses a series of matrices P_i to gradually reduce the matrix A to upper triangular shape R. The product of all these P_i eventually form the orthogonal matrix Q. Individually, every P_i reduces the elements below the diagonal in ith column to zero. For P_1

$$P_1 \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ 0 & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix}$$

Repeating this process, we have

$$P_n P_{n-1} \dots P_2 P_1 A = Q^T A = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ 0 & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{nn} \end{pmatrix} = R \tag{3.9}$$

Such that $QR = A$ with R upper triangular. To perform this process, we need to find the shape of P_i and have to prove all these matrices are orthogonal.[3]

We define a *Householder* matrix H as

$$H = I - \pi^{-1} v v^T \tag{3.10}$$

where v is a vector and $\pi = \frac{1}{2}|v|^2$. The normalization is normally written this way because it skips taking a square root. A Householder matrix is symmetrical

$$H^T = (I - \pi^{-1} v v^T)^T = I^T - \pi^{-1} v^{T^T} v^T = I - \pi^{-1} v v^T = H$$

Then, orthogonality follows as

$$\begin{aligned} H^T H = (I - \pi^{-1} v v^T)^2 = I + \pi^{-2}(v v^T)(v v^T) - 2\pi^{-1} v v^T = I + \pi^{-2} v 2\pi v^T - 2\pi^{-1} v v^T \\ = I + 2\pi^{-1} v v^T - 2\pi^{-1} v v^T = I \end{aligned}$$

Householder matrices work on a corresponding vector x, if v is chosen correctly, as

$$Hx = -\sigma e_1 \tag{3.11}$$

where e_1 is the unity vector and $\sigma = \pm|x|$. To show this, we take $v = x + \sigma e_1$. Then

$$\pi = \frac{1}{2}(x + \sigma e_1)^T (x + \sigma e_1) = \frac{1}{2}(x^T x + 2\sigma x_1 + \sigma^2) = \sigma^2 + \sigma x_1 \tag{3.12}$$

[3] Because products of orthogonal matrices are orthogonal

Because $x^T x = \sigma^2$ and x_1 is just the first component of x. We now calculate Hx using equation (3.12)

$$\begin{aligned} Hx &= x - \pi^{-1} u u^T x = x - (\sigma^2 + \sigma x_1)^{-1}(x + \sigma e_1)(x + \sigma e_1)^T \\ &= x - (\sigma^2 + \sigma x_1)^{-1}(x + \sigma e_1)(x^T x + \sigma e_1) = x - (x + \sigma e_1) = -\sigma e_1 \end{aligned}$$

Proving (3.11). Returning to our decomposition, we consider the k-th step where we have

$$A_k = \begin{pmatrix} R_k & r_k & B_k \\ 0 & c_k & D_k \end{pmatrix} \tag{3.13}$$

R_k and r_k are already part of the final matrix R.[4] B_k and D_k are just arbitrary matrices, and c_k is the vector we are looking to transform. To do that, we choose

$$P_k = \begin{pmatrix} I_k & 0 \\ 0 & H_k \end{pmatrix} \tag{3.14}$$

With H_k defined as before with corresponding vector c_k. Notice that P_k is also orthogonal. Now

$$\begin{aligned} A_{k+1} = P_k A_k = \begin{pmatrix} I_k & 0 \\ 0 & H_k \end{pmatrix} \begin{pmatrix} R_k & r_k & B_k \\ 0 & c_k & D_k \end{pmatrix} = \begin{pmatrix} R_k & r_k & B_k \\ 0 & H_k c_k & H_k D_k \end{pmatrix} \\ = \begin{pmatrix} R_k & r_k & B_k \\ 0 & \sigma e_1 & H_k D_k \end{pmatrix} \end{aligned} \tag{3.15}$$

bringing the reduction one step further. Eventually, $A_n = R$ is upper triangular and $A_1 A_2 \dots A_n = Q$. Aditionally, since products of orthogonal matrices are orthogonal, Q is orthogonal because P_i is orthogonal.

The large advantage of this method is that we do not actually have to calculate a matrix product of order n every step. From equation (3.15), it is clear that the only product to be calculated is $H_k D_k$, which becomes progessively smaller as k increases. Additionally, the transformation matrices $H_1, H_2, \dots H_n$ do not have to be calculated and stored individually, because for any matrix A of the size of H_i

$$H_i A = (I - \pi^{-1} v v^T) A = A - 2\pi^{-1} v (A^T v)^T$$

Showing that we only need to store v to compute any of the transformations.

3.3 Singular Value Decomposition

Singular value decomposition (SVD) is a more advanced method of decomposing a matrix. It has the possiblity to produce much higher accuracy, but requires

[4] Capitals stand for matrices, lower case are vectors

more computing power. Although QR decomposition is sufficient in most cases, SVD is preferred for very large matrices.[5] If computing power is not a problem, SVD is superior.

3.3.1 Existence

Again, we first have to show the singular value decomposition actually exists. For any matrix A, the singular value decomposition is

$$A = U^T S V \tag{3.16}$$

where both U and V are orthogonal matrices and S is a diagonal matrix with nonnegative, so called singular values on it's diagonal. As will be proven, singular value decomposition is similar to eigenvalue decomposition, but is not limited to only matrices that are diagonizable.

As we have seen before, for any arbitrary matrix A, the matrix $A^T A$ is symmetrical and positive definite. We now show it also has nonnegative eigenvalues. Let u be an eigenvector and λ the corresponding eigenvalue of $A^T A$. Additionally, call $Au = b$. Now

$$|b|^2 = u^T A^T A u = u^T \lambda u = \lambda |u|^2$$

Then $|b|^2 \geq 0$ and $|u|^2 \geq 0$ implies $\lambda \geqslant 0$. We now have all the tools to prove the existence of the singular value decomposition. Rewriting the decomposition

$$V^T A U = \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} \tag{3.17}$$

Where Σ is the diagonal matrix with the nonzero singular values.

We now rename the eigenvalues of $A^T A$ as σ^2, which is possible because they are positive. We label them in descending order $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_n^2$. Additionally, let $U_1 = (u_1, \ldots, u_r)$ be the matrix of eigenvectors corresponding with eigenvalues $\sigma_i^2 > 0$, and let $U_2 = (u_{r+1}, \ldots u_n)$ be the matrix of eigenvectors corresponding with $\sigma_i^2 = 0$. Note that U_2 is not unique, and therefore the decomposition will not be unique. Σ is then defined as $diag(\sigma_1, \ldots, \sigma_r)$.

Now $U_1^T A^T A U_1 = \Sigma^2$ per definition. Rewriting, $\Sigma^{-1} U_1^T A^T A U_1 \Sigma^{-1} = I$. Furthermore, $U_2^T A^T A U_2 = 0$. We now choose $V_1 = A U_1 \Sigma^{-1}$. Then V_1 is orthogonal, because $V_1^T V_1 = \Sigma^{-1} U_1^T A^T A U_1 \Sigma^{-1} = I$. Finally, choose V_2 such that the matrix (V_1, V_2) is orthogonal, which we can always do. Filling in equation (3.17)

[5] For 3 or more loops, or dimensionality of 8 or higher

$$V^T AU = (V_1, V_2)^T A(U_1, U_2) = \begin{pmatrix} (V_1^T)AU_1 & V_1^T(AU_2) \\ V_2^T(AU_1) & V_2^T(AU_2) \end{pmatrix} =$$
$$\begin{pmatrix} (\Sigma^{-1}U_1^T A^T)AU_1 & V_1^T(0) \\ V_2^T(V_1\Sigma) & V_2^T(0) \end{pmatrix} = \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix}$$

Proving the decomposition exists. In the last two steps, we have used some of the identities showed earlier, orthogonality and eigenvalue relations.

3.3.2 Computation

We will only briefly sketch the current method of computing the SVD of a matrix. In reality, the process of computation is much more complex, because of the existence of many different techniques and transformations that can be applied to the initial matrix that make the computation both faster and more accurate. We will only describe some of the more prevalent of these techniques.

We have previously determined that computing the SVD of a matrix is just an eigenvalue problem for the matrix $A^T A$. There are different methods to calculate eigenvalues of large matrices, some faster or more accurate than others. We decribe a popular one based on QR decomposition. The method is usually called the *QR algorithm* and relies on successive applications of QR decompositions.

Consider a arbitrary matrix B, and call it B_0. On step k, the QR algorithm decomposes $B_k = Q_k R_k$ by regular QR decomposition. Then we define $B_{k+1} = R_k Q_k$ and start by decomposing again. Note that

$$B_{k+1} = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T B_k Q_k$$

Because Q_k is orthogonal, all B_k are similar, meaning they have the same eigenvalues. The shape of B_k converges to a so-called Shur form, where the matrix is upper triangular with it's eigenvalues on the diagonal. When n iterations are completed, we can write $B_0 = Q^T B_n Q$ with $Q = Q_0 Q_1 \dots Q_n$. With the eigenvalues known, calculating the eigenvectors of the matrix B_n is quite easy, as it is upper triangular. If x_i are eigenvectors of B_n, the eigenvectors of the original matrix B_0 are Qx_i.

Practically, this algorithm is quite expensive. There are some techniques to reduce the amount of calculations by manipulating the matrix before the algorithm is applied. First, the matrix is usually transformed into upper Hessenberg shape as

$$H = \begin{pmatrix} h_{11} & \cdots & \cdots & h_{1n} \\ h_{21} & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ 0 & & h_{n(n-1)} & h_{nn} \end{pmatrix}$$

This is achieved with Householder transformations similar to what is done when computing QR decomposition. It turns out that this transformation significantly speeds up the QR algorithm.

The more the QR algorithm is applied, the more accurate the eigenvalues and eigenvectors can be calculated. However, it can mathematically be proven that the eigenvalues converge faster when there is a larger separation between them. Therefore, several shifting methods are used to shift eigenvalues in order to compute them more easily. Once the shifted eigenvalues are computed, they can be shifted back to find their actual values.

Chapter 4

Results

Having described some methods to calculate solutions to large systems of equations, we are now ready to tackle our problem. We will discuss all points dealt with in section 2.4

4.1 Solvability

To check solvability of the relevant systems of equations, we use the condition

$$rk(A) = rk(A|b) \tag{4.1}$$

In our case, the matrix A is the matrix of equations described in chapter 2, and the solution vector b is just a column of ones.

For confirmation, both QR and SVD were used to calculate the results listed in Table 4.1. The program was written in Python. Numpy, which is a library within Python, allowed for easier handling of large array-like object such as the matrices and vectors required. Withing Numpy, the linalg package was used to calculate both QR and SVD of the matrix. This package is a wrapper around LAPACK[1], which was originally written for Fortran. The package includes many different state-of-the-art methods of various calculations within the field of numerical linear algebra, including extremely fast and precise algorithms for QR and SVD. The matrix rank was distinctly visible with both methods, even with the largest matrices.

[1] Linear Algebra PACKage

Dimension	2			3			4			5			6		
Propagators	3	4	5	5	6	7	7	8	9	9	10	11	11	12	13
Order 1	✔	✔	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Order 2	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Order 3	✖	✖	✔	✖	✖	✔	✖	✖	✔	✖	✖	✔	✖	✖	✔

Table 4.1: Solvability for several dimensions, amounts of propagators and order of terms

Note that only the number, and not the configuration of propagators is a relevant coefficient. All configurations of propagators have been tested, and it never made any difference. From these results we can already draw some significant conclusions. In [1], it was conjectured one would need higher order terms for higher dimensions. Instead, these results indicate that $d = 2$ is a special case. For higher dimensions, there is a strict pattern where the last possible reduction by unitarity is solvable, but only with terms of order 3.

4.2 Matrix Rank and Tensor Structure

To confirm that we can indeed reduce iGraphs of order $2d+1$ with cubic terms for $d > 2$ as the above results suggest, we compare the amount of tensor structures with the matrix ranks calculated with both QR and SVD. To calculate the tensor structures, equation (2.12) is used. Results are listen in Table 4.2

Dimension	3	4	5	6
Tensor Structures	360	831	1631	2876
Matrix Rank	360	831	1631	2876

Table 4.2: Comparison of matrix rank and tensor structures for order of terms 3 and propagators $2d + 1$

In every case, the matrix rank is equal to the amount of tensor structures. This indicates that terms of order 3 are enough to reduce any reducible iGraph with two loops. Because we now know these reductions are possible, we can explicitly solve them.

4.3 Solving the coefficients

To definitively confirm the reduction of iGraphs of order $2d+1$ with cubic terms, we solve the corresponding linear systems. In order to check if the calculated coefficients apply globally to all values of $l_1$and l_2, we check them against randomly generated l_1 and l_2 as described in section 2.4.3. Table 4.3 shows the result of a numerical computation of the coefficients which are checked against

random l_1 and l_2.[2] These were done in $d = 4$ with cubic terms.

Propagators	8	9
Random l_1 and l_2	0.999999997206	-0.599001394349
	0.999999999578	3.458581682780
	0.999999999534	0.827390263649
	1.000000000169	0.406886263765
	1.000000000652	1.323267241981
	1.000000000233	0.184702651816
	0.999999999523	0.719414617007
	1.000000000559	0.157197961494
	0.999999999522	1.341094285732
	1.000000000093	1.218648200823
Previous l_1 and l_2	0.999999999534	0.733363900901

Table 4.3: Numerical checks of calculated coefficients against random l_1 and l_2 for 8 propagators (left) and 9 propagators (right)

Clearly, the reduction exists for 9 propagators, but fails with 8. Note that we also check against one of the equations from the original system in the last line. For 8 propagators, we see that the computed solution is not an actual solution to the equations in the system, which is consistent with the results from section 4.1 where we saw the 8 propagator case is not solvable.

4.4 Unusual behaviour in two dimensions

In our proces of reducing several iGraphs, there have been some unusual properties that only appeared in the case of $d = 2$. The first one shows in Table 4.1. There, we see that the pattern of solvability is very different in $d = 2$. This also reflects in the fact that we only need quadratic terms to fully reduce two-loop iGraphs, while in higher dimensions we need cubic terms. Additionally, in [1] it was mentioned that the number of tensor structures is lower than expected in $d = 2$, while for higher dimensions it is exactly as expected.

Figure 4.1 displays a Feynman diagram with five propagators.

[2] Shows results were calculated with QR decomposition. SVD obviously yields identical results, but with slightly higher accuracy

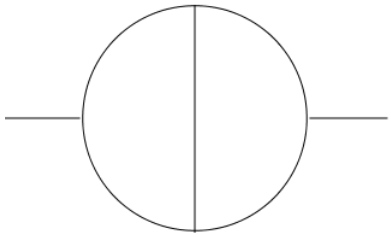

Figure 4.1: Feynman diagram of a special case in 2 dimensions

Normally, the corresponding iGraph should be reducable in $d = 2$ with quadratic terms. However, it turns out that in this specific case, it is possible to choose values for $l_1^\mu$and l_2^μ such that all propagators vanish. As such, for this specific case, reduction is actually not possible.

4.5 Conclusions

We have explored the possibilities of reduction of iGraphs with one and two loops. We have shown that the number of reducible iGraphs is limited by unitarity, and we have seen that we can carry out all possible reductions in two-loop iGraphs with cubic terms. To do this, we have used efficient programming to solve the associated large systems of equations with the described techniques of QR and SVD. We can now consider both the one-loop and two-loop cases solved.

Unfortunately, the techniques used here are not suitable for practical use. While our method of solving the coefficients is straightforward, it is also not fast enough to be used for actual calculations. Finding a way to calculate the associated coefficients for specific integrals is the next step that can lead to much faster computation of Feynman integrals.

Additionally, iGraphs with three or more loops have not yet been fully solved. While these iGraphs represent smaller contributions to the calculation of cross sections, they cannot be ignored. To show if reduction is possible, it is possible to use the same method as above, but one would expect to require more time and faster computers to solve the large systems of equations that result from these larger iGraphs.

Bibliography

[1] Ioannis Malamos, *Reduction of one and two loop Amplitudes at the Integrand level*

[2] G.W. Stewart, *Introduction to matrix computations*, Academic Press 1973

[3] Gene H. Golub, Charles F. Van Loan, *Matrix computations*, The Johns Hopkins University Press 1996

[4] The used Python script can be found at: http://codepad.org/mrlYkkIe